Discovering Network Topology by Correlating End-to-end Delay Measurements

by

Jason Ivan Howard Baron

B. S., Computer Science and Engineering
Massachusetts Institute of Technology (2000)

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science
at the

Massachusetts Institute of Technology

February 6, 2001

Author……

Department of Electrical Engineering and Computer Science
Februrary 6, 2001

Certified by……

Mark W. Garrett
VI-A Company Thesis Supervisor
Telcordia Technologies

Certified by……

Hari Balakrishnan
Assistant Professor
M.I.T. Thesis Supervisor

Accepted by……

Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Discovering Network Topology by Correlating End-to-end Delay Measurements

by

Jason Ivan Howard Baron

Submitted to the Department of Electrical Engineering and Computer Science

February 6, 2001

In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Conventional methods of discovering network topology require the cooperation of network elements. We present a method of network topology discovery based solely upon end-to-end delay measurements that requires only the cooperation of *end* systems. Previous work using end-to-end measurements has focused on discovering tree topologies; the method here discovers more general networks. The discovery method is based on two algorithms: the matroid algorithm and the correlation algorithm. This work develops and validates the correlation algorithm for three increasingly sophisticated network models using simulation. We also develop an indicator of the quality of a particular result.

VI-A Company Thesis Supervisor: Mark W. Garrett
Title: Director, Internet Performance Research Group, Telcordia Technologies, Inc.

Thesis Supervisor: Hari Balakrishnan
Title: Assistant Professor Electrical Engineering and Computer Science

Acknowledgements

Writing an acknowledgement section is difficult for three reasons. First, a thesis is only possible because of the efforts of many individuals. Therefore, it is incumbent upon the author to not forget anyone worthy of credit. Second, the acknowledgement section is the only section of the thesis where the author can freely express personality. Thus, the author must be sure to be witty and funny, or at least make some semblance of wit and humor, because there are no other opportunities to do so. Finally, an acknowledgement section is written last, immediately after the thesis supervisor has approved the work[*]. Therefore, the author must make haste before the supervisor has a change of heart. With an understanding of the difficulties of the acknowledgement section in hand and a little more serious tone, we proceed.

Dr. Mark Garrett was the most involved, enthusiastic, and insightful supervisor that one could have. Over two years ago, when I was deciding what to focus my masters research on, Mark presented the Felix project to me in such a way that I knew that I had to work for him. Mark's sense of humor helped keep the research fun. For instance, my progress was sometimes measured by Mark holding up the stack of papers that constituted my current version of the thesis and exclaiming: "Feels heavier than before, good!".

When I introduced my thesis topic to Professor Hari Balakrishnan, he joked that I should work with him at MIT, instead of for Mark at Telcordia. Despite the distance between Telcordia and MIT, I often felt like Hari was in the next room because he would reply immediately to my e-mails.

Many ideas came out of conversations with the mathematician, Dr. David Shallcross. David's keen interest in my work was very refreshing. Special thanks to Professor Paul Seymour for developing the matroid algorithm that paved the way for this work. Many thanks to Dr. Brian Coan, who having gotten his doctorate at MIT, gave me perspective and insight into the thesis process. Thank you Dr. Will Leland for ensuring that my thesis did not reveal too many of Telcordia's secrets.

Fellow 6A'er, officemate, and lunchmate Gina Yip was an amazing friend. We had great conversations about all of the major events that transpired during the summer and fall semesters, including the Olympics, the subway series, and the improbable presidential election, that we showed was not so improbable after all[**].

Finally, I must thank all of my friends who through the following biweekly exchange made me finish my thesis in a timely fashion. Friend: "So, you finished yet?" Me: "Perhaps, another two weeks, it looks like now." Friend: "That's what you said two weeks ago." Me: "Yea, and I'm probably going to say it again in another two weeks."

I guess I must now thank family, since family get-togethers are tough enough as it is. I thank my cousins Brant and Joel, and my Aunt Sarah for providing constant laughter and love.

I am forever grateful to my parents for their love and support. They have taught me the value of hard work as Vince Lombardi aptly once said: "The dictionary is the only place that 'success' comes before 'work'."

Finally, I would like to thank Dr. Kenneth Calvert, Dr. Matthew Doar, and Dr. Ellen Zegura for the freely available *tiers* software that was used in our simulations.

---

[*] Although we do not have sufficient empirical evidence to support this claim, we nevertheless believe it to be the case.

[**] Proof is left as an exercise for the reader.

Contents

List of Figures **9**

1 Introduction **11**

1.1 Motivation 11

1.2 Correlation and Matroid Algorithms 12

1.3 Related Work 13

1.4 Outline 15

2 Background **16**

2.1 Terminology 16

2.2 Network Topology Generation and Reduction 18

2.3 Implementation 25

3 Non-Overlapping Congestion **27**

3.1 Network Model 27

3.2 Applying a Threshold to the Delay Time Series 29

3.3 Algorithm 31

3.3.1 First Correlation Algorithm 31

3.3.2 Second Correlation Algorithm 32

3.3.3 Time and Space Complexity 38

3.3 Results and Discussion 39

4 Overlapping Congestion in Discrete Time **42**

4.1 Network Model 42

4.2 Probability Analysis 43

4.2.1 Observation Probability 43

4.2.2 Trigger Probability 45

4.2.3 Relating Trigger Probabilities to Observation Probabilities 47

4.2.3.1 3-Path Equations 51

4.2.3.2 n-Path Equations 53

4.3 Algorithm 56

4.3.1 *Pruning_threshold* 62

4.3.2 Time and Space Complexity 63

4.4 Results and Discussion 65

4.4.1 *Pruning_threshold* and η 65

4.4.2 Running Time 67

4.4.3 Varying the Link Congestion Rates 69

4.4.4 Largest Solved Networks 71

4.4.5 Summary 74

5 Overlapping Congestion 75
5.1 Network Model 75
5.2 Algorithm 76
5.3 Results and Discussion 78
6 Conclusions 81
Appendix 84
References 88

List of Figures

Figure 1-1: The topology discovery method. 12

Figure 2-1: Illustration of network terminology. 17
Figure 2-2: Example of a path's end-to-end delay time series. 18
Figure 2-3: Series links.. 20
Figure 2-4: Consecutive equivalent links.. 21
Figure 2-5: Eliminating consecutive equivalent links 22
Figure 2-6: Non-consecutive equivalent links.. 23
Figure 2-7: Original network topology generated by tiers. 24
Figure 2-8: Visible network.. 24
Figure 2-9: Reduced network. 24
Figure 2-10: Code tool-chain……..…….25

Figure 3-1: Applying a threshold. 29
Figure 3-2: Binary functions. 30
Figure 3-3: Circling algorithm.. 31
Figure 3-4: 2-path trigger probabilities. 33
Figure 3-5: Pseudocode of the correlation algorithm. 36
Figure 3-6: Reduced network. 40
Figure 3-7: Discovered network.. 40

Figure 4-1: Calculating an observation probability. 44
Figure 4-2: Relating trigger probabilities to observation probabilites. 47
Figure 4-3: Pseudocode of the calculate-trigger-probability procedure. 58
Figure 4-4: Psuedocode of the correlation algorithm. 59
Figure 4-5: Sample network topology for demonstrating the correlation algorithm 60
Figure 4-6: Path-link matrix for the network shown in Figure 4-5. 60
Figure 4-7: Binary tree constructed by the correlation algorithm. 61
Figure 4-8: Distribution of the true and false leads.. 62
Figure 4-9: Simulation length as a function of the number links. 67
Figure 4-10: Running time of the correlation algorithm. 68
Figure 4-11: Varying the link congestion rates. 69
Figure 4-12: Reduced network used for varying the link congestion rates.. 70
Figure 4-13: Reduced network……..71
Figure 4-14: Discovered network.. 72
Figure 4-15: Reduced network. 73
Figure 4-16: Discovered network.. 73

Figure 5-1: Sample of several paths' binary functions. 76
Figure 5-2: Setting of the *pruning_threshold* and the *final_pruning_threshold*. 77
Figure 5-3: Running time of the correlation algorithm. 78
Figure 5-4: Histogram of separated fully specified trigger probabilities.. 79
Figure 5-5: Histogram of overlapping fully specified trigger probablities.. 80

Chapter 1

Introduction

1.1 Motivation

Knowledge of network topology is essential for network management and network provisioning. However, conventional methods of network discovery (e.g., SNMP-based autodiscovery) rely on the cooperation of network elements and therefore have several potential shortcomings. For instance, network elements may be uncooperative due to heavy loading that disables autodiscovery in order to reduce processor load. Or, the owner of the network elements might restrict their access rights.

In addition, network elements may not have the functionality that is desired. Traditional autodiscovery tools may use outdated protocols or protocols that have not yet been implemented by network elements. Autodiscovery tools may also be limited because they can only detect the *logical* connectivity of a particular network layer. For example, ICMP and traceroute can not detect ATM switches.

A discoverer may not want network elements to be aware of the discovery process. In a military context, we might want to make a map of the enemy's network. Or, an ISP might want to verify the connectivity of a carrier network in order to verify service agreements.

This work responds to these concerns and provides a starting point for a system that can discover a broad class of network topologies from end-to-end delay measurements. The work grows out of a 1997 DARPA proposal by Christian Huitema at Telcordia Technologies. The proposal received funding and became the Felix Project [6, 7]. The

Felix Project focused on using non-invasive end-to-end measurements to determine network characteristics.

End-to-end measurements only require that the *end* systems be centrally controlled. Any interior network elements may be uncooperative to the extent that they do not respond to direct probing, such as ICMP, but they do forward probe packets. The topology discovery method presented here is based on two algorithms: the correlation algorithm and the matroid algorithm.

1.2 Correlation and Matroid Algorithms

Monitors collect the end-to-end delay measurements that are the starting point for discovering network topology. Monitors are placed at certain nodes in the network and collect delay measurements by time-stamping packets that are sent to other monitors. The links in a network that are traversed when a monitor sends a packet to another monitor constitute a path. The collected data is organized as end-to-end delay time as a function of time for each path.

The collected data is then used to discover the topology of the network in two steps. First, the correlation algorithm determines the links in the network by correlating the times when paths exhibit common end-to-end delay characteristics. The output of the correlation algorithm is a path-link matrix that identifies which links are on which paths. The matroid algorithm (invented by P. D. Seymour) [7] then reconstructs the network topology based upon the path-link matrix. The solution method is illustrated in Figure 1-1.

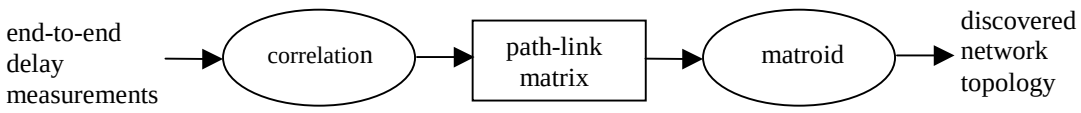


Figure 1-1: The topology discovery method.

The matroid algorithm uses linear algebra and graph theory to construct the network topology from the path-link matrix. The path-link matrix does not necessarily completely determine the network topology. For example, the matroid algorithm may produce a network that contains *split nodes*, which are a set of nodes in the discovered network that correspond to the same node in the original network. The matroid algorithm can also output localized uncertainties in the network topology or *clouds*. For the purposes of this paper, the matroid algorithm is treated as a black box.

The contribution of this thesis is the development and validation of topology discovery algorithms based upon the idea of correlating common end-to-end delay time characteristics that occur on network paths. Three correlation algorithms that are based upon accompanying network models form the core of the work. We start with relatively simplistic network models and move towards more complex ones. Validation of the correlation algorithms is done using simulation. We use simulation instead of real-world traffic data because exploring the ideas of the correlation algorithm would initially be too difficult with real-world data.

1.3 Related Work

A number of related papers have been published recently. We classify this related work into three areas: end-to-end measurement systems, analysis and modeling of real-world end-to-end measurements, and analysis and modeling of simulated end-to-end measurements. The present work belongs in the third area, analysis and modeling of simulated end-to-end measurements.

Measurement systems are primarily concerned with how end-to-end measurements can be made accurately and what type of infrastructure is required to support these measurements. These projects are concerned with many systems related issues such as scaling, distributed computing, and security. The National Internet Measurement Infrastructure (NIMI) project [13] proposes to build a large-scale measurement infrastructure on top of the existing Internet, similar to the Domain Name System. NIMI is primarily concerned with scalability and security. Another significant measurement

system is Surveyor [20], consisting of 38 machines spread around the world. One-way delay and packet loss is measured between these machines. Additional projects in this area are: Skitter [19], Internet Performance Measurement and Analysis (IPMA) at Merit [11], and Internet Monitoring & PingER at Stanford Linear Accelerator Center (SLAC) [10].

The second area of related work is analysis and modeling of real world end-to-end measurements. This work uses end-to-end measurements from real-world network traces to infer characteristics about the network. Most of the work focuses on analyzing delay and loss models. For example, Yajnik *et al*. [23] present a method of estimating the loss rates on links of a known multicast tree topology using the receivers' loss patterns. Further references in this area include [1, 12, 14, 24].

The third broad area of related work that we consider is analysis and modeling of simulated end-to-end measurements. A number of simulation studies have been done on inferring characteristics of multicast tree networks based upon the loss patterns of receivers. Caceres *et al*. [3] and Ratnasamy *et al*. [17] have developed maximum likelihood estimations of multicast tree topologies and the packet loss rates on these links. The basic insight is that if a multicast packet is lost along a link in the tree, all intended recipients lower in the tree will be affected. Intuitively, the *closer* the loss pattern of two receivers, the more likely it is that they are *closely* related in the multicast tree.

Moving from multicast to unicast streams but still considering only tree topologies, Rubenstein *et al*. [18] and Harfoush *et al*. [9] suggest two different probing packet techniques for characterizing the internals of a network. Rubenstein *et al*. propose analyzing two Poisson probe flows in order to detect if two co-located receivers or two co-located senders have shared points of congestion. Harfoush *et al*. suggest a method of identifying shared loss by using Baysian probes. Harfoush *et al*. are able to reconstruct the tree topology and loss rates between a server and a set of clients using these probes. Further references in this area include [2, 5, 16].

In the present work, we address the problem of inferring network topology for a mesh of paths. We are not limited to tree topologies in any way. In addition to discovering the network topology, we provide an estimate of individual link congestion rates.

1.4 Outline

We present background material and the network topology generation process in Chapter 2. Chapters 3, 4, and 5 delve into the three network models that form the core of the work. Each of these chapters contains an explanation of the network model, a correlation algorithm, results, and discussion. Finally, we present conclusions and future work in chapter 6.

Chapter 2

Background

Before we examine the three network models that form the core of this paper in chapters 3, 4 and 5, we present background material. In section 2.1, we introduce terminology that will be used extensively throughout the paper. Section 2.2 explains how we generate the network topologies that are used for testing the topology discovery algorithms. Finally, in section 2.3, we present an overview of how the various pieces of code fit together.

2.1 Terminology

We define a *network* as a set of *nodes* and directed edges, or *links*. Special nodes in the network are *monitors*. Nodes that are not monitors are *interior nodes*. A *path* is the set of nodes and links that are traversed if data is sent from one monitor to another monitor. Paths are *simple*, having no repeated nodes or links and consequently do not have loops. Paths are not necessarily *symmetric*. That is, the set of nodes that is traversed on the path between monitor M_1 and monitor M_2 need not be the same as the set of nodes that is traversed by the path from monitor M_2 to M_1. A path that is not symmetric is *asymmetric*. Paths are also *stable*. A stable path is a path does not change over time. Figure 2-1 illustrates these concepts.

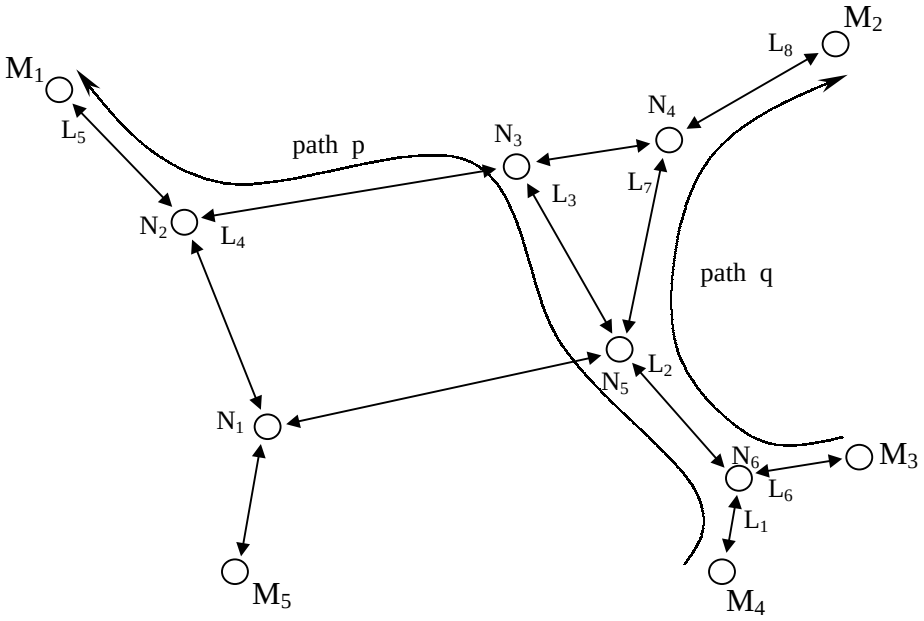


Figure 2-1: Illustration of network terminology.

The network in Figure 2-1 has eleven nodes. Five of these nodes are monitors and they are labeled: M_1, M_2, M_3… The six interior nodes are labeled: N_1, N_2, N_3… Some links are labeled next to arrowheads that indicate the direction of the link. The links that constitute path *p* are: L_1, L_2, L_3, L_4, and L_5, while the links that constitute path *q* are: L_2, L_6, L_7, and L_8.

All of the network models are based upon packet-switched networks. By time-stamping the packets that are sent from one monitor to another, the end-to-end delay time for a path is recorded. If time-stamped packets are sent over time, then the end-to-end delay time series from one monitor to another or a path's end-to-end delay time series is constructed. Figure 2-2 is an example of such a time series.

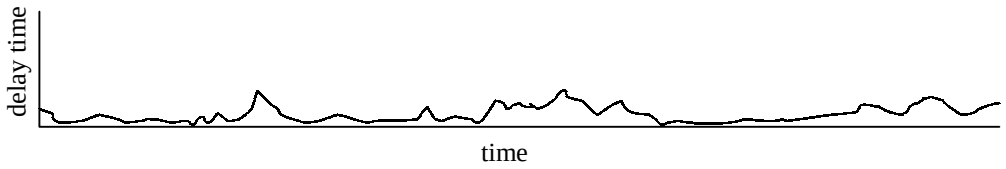


Figure 2-2: Example of a path's end-to-end delay time series. Note that this time series is only illustrative and is not meant to represent a recorded time series. In practice, the time series tends to be more punctuated and not as smooth as what is illustrated.

Finally, we formalize the path-link matrix that was introduced in chapter 1. One axis of the matrix is indexed by a path name and the other axis is indexed by a link name. We put a 1 in position (i, j), if link *j* is on path *i* and a 0 otherwise. Constructing such a matrix for paths *p* and *q*, which are defined in Figure 2-1 yields:

	L_1	L_2	L_3	L_4	L_5	L_6	L_7	L_8	L_9
P	1	1	1	1	1	0	0	0	0
Q	0	1	0	0	0	1	1	1	0

The path-link matrix serves as the interface between the correlation algorithm and the matroid algorithm. Note that the path-link matrix does not indicate how many nodes are in the network, nor whether or not two different paths share common nodes. Unlike an adjacency-matrix representation of a graph, the path-link matrix does not necessarily completely specify a network's topology.

2.2 Network Topology Generation and Reduction

In this section, we discuss the methodology we use to create network topologies for testing the topology discovery algorithms. We wanted a method that could generate a wide range of realistic network topologies, while keeping in mind the limitations of the discovery algorithms. Therefore, we create an initial randomized, realistic network topology using the *tiers* program by Calvert *et al.* [4]. Next, we choose a fixed number of

leaf nodes at random as monitors. We remove nodes and links that are not traversed by any paths, creating the *visible* network. Finally, we simplify or *reduce* the visible network, creating the *reduced* network.

The tiers program generates randomized and realistic topologies according to a three tier hierarchical structure. The three tiers are: Wide Area Networks (WANs), Metropolitan Area Networks (MANs), and Local Area Networks (LANs). Several parameters control graph generation. The user can specify the number of nodes in each type of network as well as the number of MANs per WAN, and LANs per MAN. Additionally, the connectivity within a network and between different types of networks can be specified. We refer to the network generated by tiers as the *original* network.

Monitors are then selected randomly from among the leaf nodes of the original network. We choose the leaf nodes as monitors because they tend to be at the edge of the network and therefore much of the network is contained on the paths between them. If we chose monitors in the center of the network, then the network topologies would probably not be very complex or interesting.

Given the choice of monitors, we determine the paths based upon shortest hop routing. This is implemented using breadth first search from each monitor. This implementation lends itself to the generation of asymmetric paths. Empirical work by Vern Paxson [15] found that in one network approximately 30% of the paths exhibited asymmetry. We find paths generated using breadth first search to be asymmetric approximately 35% of time. We assume that the routing is stable and thus paths do not change over time. After removing the nodes and links that are not traversed by any paths in the original network, the visible network remains.

From the visible network, we proceed to reduce the graph. The correlation algorithm requires that each link in the network be traversed by a unique set of paths. The matroid algorithm assumes that all paths are simple, a monitor is not an interior node in the network, and that paths destined to the same monitor do not converge and then diverge. This final assumption of the matroid algorithm means that all paths into a monitor form a *sink tree* [21].

The visible network satisfies the requirements of the matroid algorithm. A sink tree into each monitor is assured by using breadth first search because the adjacency list that is used to represent the network is examined in the same order in determining all paths. We ensure that monitors are not in the interior of the network by choosing monitors from among the leaf nodes in the original network. Finally, shortest hop paths do not contain any repeated nodes or links and therefore the simple path constraint is satisfied.

The visible network does not necessarily satisfy the correlation algorithm's requirement that each link be traversed by a unique set of paths. We classify links that are traversed by the same set of paths as other links into two categories: *series* links and *equivalent* links. Series links can be found by identifying those nodes that have an in-degree and out-degree of one or an in-degree and out-degree of two. To illustrate this concept, consider Figure 2-3.

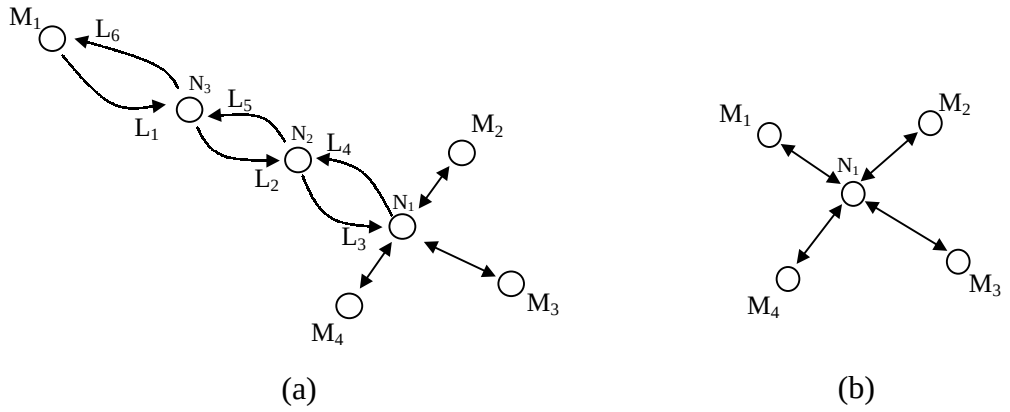
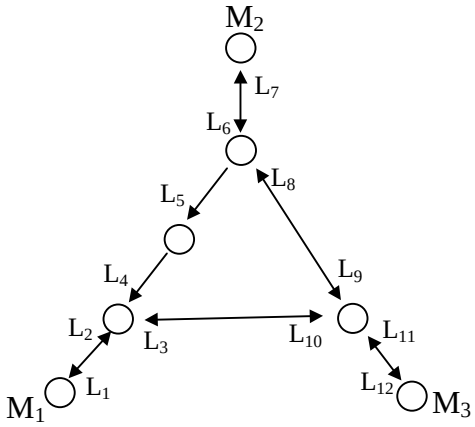


Figure 2-3: Series links. Figure 2-3(a) contains 6 series links. Links L_1, L_2, and L_3 are all traversed by paths M_1-M_2, M_1-M_3, and M_1-M_4, while links L_4, L_5, and L_6 are traversed by paths M_2-M_1, M_3-M_1, and M_4-M_1. Figure 2-3(b) modifies the network, eliminating the series links.

In Figure 2-3(a), nodes 2 and 3 both have an in-degree and out-degree of two. If we designate the path from M_1 to M_3 as M_1-M_3, then links L_1, L_2, and L_3 are traversed by paths M_1-M_2, M_1-M_3, and M_1-M_4, while links L_4, L_5, and L_6 are traversed by paths M_2-M_1, M_3-M_1, and M_4-M_1. We can remove these series links by removing nodes 2 and 3 and reconnecting the graph as shown in Figure 2-3(b).

An equivalent link is a link that is traversed by the same set of paths as at least one other link in the network, but it is not a series link. Equivalent links are further classified as *consecutive* equivalent links and *non-consecutive* equivalent links. A consecutive equivalent link has at least one node in common with another link that is traversed by the same set of paths as itself. A non-consecutive equivalent link is an equivalent link that does not have a common node with a link that is traversed by the same set of paths as itself.

In Figure 2-4, we present an example of consecutive equivalent links. The table in Figure 2-4(b) is an ordered set of the links that are traversed on each path. Notice that links L_7 and L_8 always appear together in the table and are found on paths M_1-M_2 and M_3-M_2. Looking at the network diagram, links L_7 and L_8 share a common node and are not series links. Therefore, links L_7 and L_8 are consecutive equivalent links. Links L_2 and L_{10} are also consecutive equivalent links, while links L_4 and L_5 are series links.



(a)

M_1-M_2: L_2, L_{10}, L_8, L_7
M_1-M_3: L_2, L_{10}, L_{12}
M_2-M_1: L_6, L_5, L_4, L_1
M_2-M_3: L_6, L_9, L_{12}
M_3-M_1: L_{11}, L_3, L_1
M_3-M_2: L_{11}, L_8, L_7

(b)

Figure 2-4: Consecutive equivalent links. Links L_7 and L_8 share a common node and are traversed by paths M_1-M_2 and M_3-M_2. Thus, L_7 and L_8 are consecutive equivalent links. Links L_2 and L_{10} are also consecutive equivalent links. The table in Figure 2-4(b) indicates the links that are traversed on the paths between the indicated monitors.

We modify the network that contains consecutive equivalent links as follows. We define a *group* of consecutive equivalent links as set of consecutive equivalent links where each link in the set is traversed by the same set of paths. We can move from the

begin node to the *end* node of a group of consecutive equivalent links by traversing only and all of those links that are in the group. We remove all of the consecutive equivalent links in a group from the network and then add to the network a single link from the begin node to the end node. After modifying the network in this manner, we remove nodes that are no longer traversed by any path. Applying this procedure to Figure 2-4 results in network found in Figure 2-5.

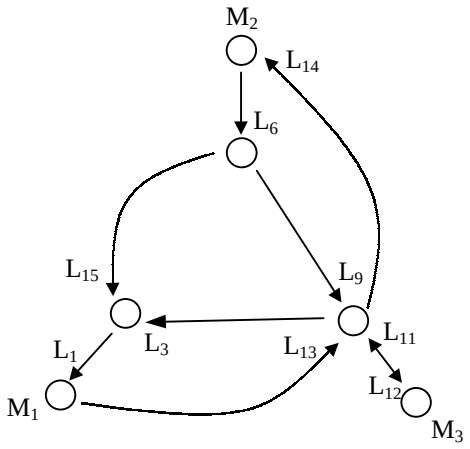


Figure 2-5: Eliminating consecutive equivalent links. Consecutive equivalent links L_2, L_7, L_8, and L_{10} are removed from the network in Figure 2-4(a). Links L_{13} and L_{14} are added to the network. In addition, the series links L_4 and L_5 are removed from Figure 2-4(a) and L_{15} is added.

An example of non-consecutive equivalent links is shown in Figure 2-6. In the figure, paths only traverse links that lie on the hexagon shape and links that are incident to a monitor, except for paths M_1-M_4, M_4-M_1, M_3-M_6, and M_6-M_3. These paths traverse links in the interior of the hexagon shape as shown. Links L_2 and L_4 are non-consecutive equivalent links, being traversed only by path M_6-M_3, while L_1 and L_3 are also non-consecutive equivalent links, being traverse only by path M_3-M_6. Paths M_1-M_4 and M_4-M_1 contain analogous non-consecutive equivalent links as well.

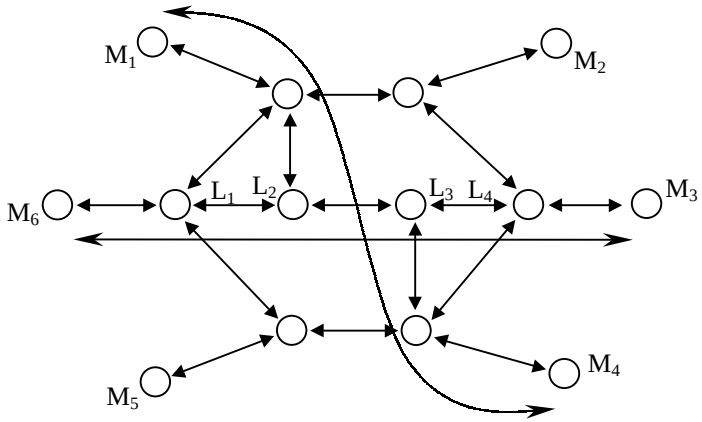


Figure 2-6: Non-consecutive equivalent links. Links L_2 and L_4 are only traversed by the path M_6-M_3 and are thus non-consecutive equivalent links. Similarly, links L_1 and L_3 are only traversed only by the path M_3-M_6 and are thus non-consecutive equivalent links as well. Analogous non-consecutive links occur on the path M_1-M_4 and M_4-M_1.

A network that has been checked for series and consecutive equivalent links and modified appropriately is a *reduced* network. The reduced network may seem very different from the visible network. However, a network that is modified because of series links does not change the graph in any structurally significant ways. Modifications to network topology that are due to consecutive equivalent links are localized in the network and the number of links in a group of consecutive equivalent links is typically small[*]. Finally, we do not use networks that contain non-consecutive equivalent links. However, networks with non-consecutive equivalent links are extremely rare in practice[**].

We show the evolution from the original network, to the visible network, and finally to the reduced network in Figures 2-7, 2-8, and 2-9, respectively.

[*] We found that most groups of consecutive equivalent links had 2 links.

[**] In generating hundreds of sample topologies, we have found only one case of non-consecutive equivalent links.

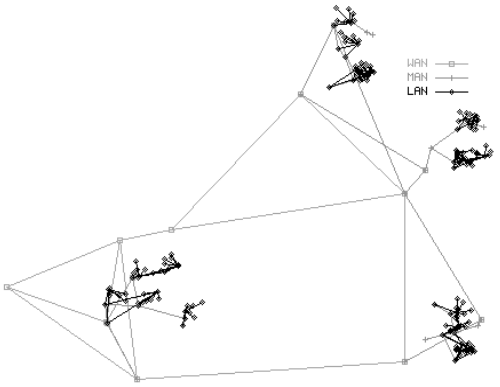


Figure 2-7: Original network topology generated by tiers. There are 180 nodes.

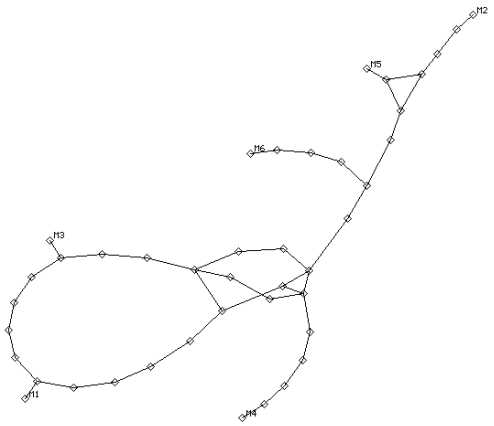


Figure 2-8: Visible network. Illustrates the network topology after monitors have been added and non-traversed nodes and links have been removed.

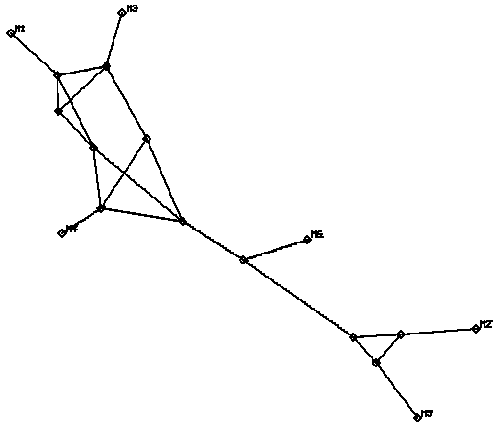


Figure 2-9: Reduced network. Series and consecutive equivalent links have been removed from the visible network.

2.3 Implementation

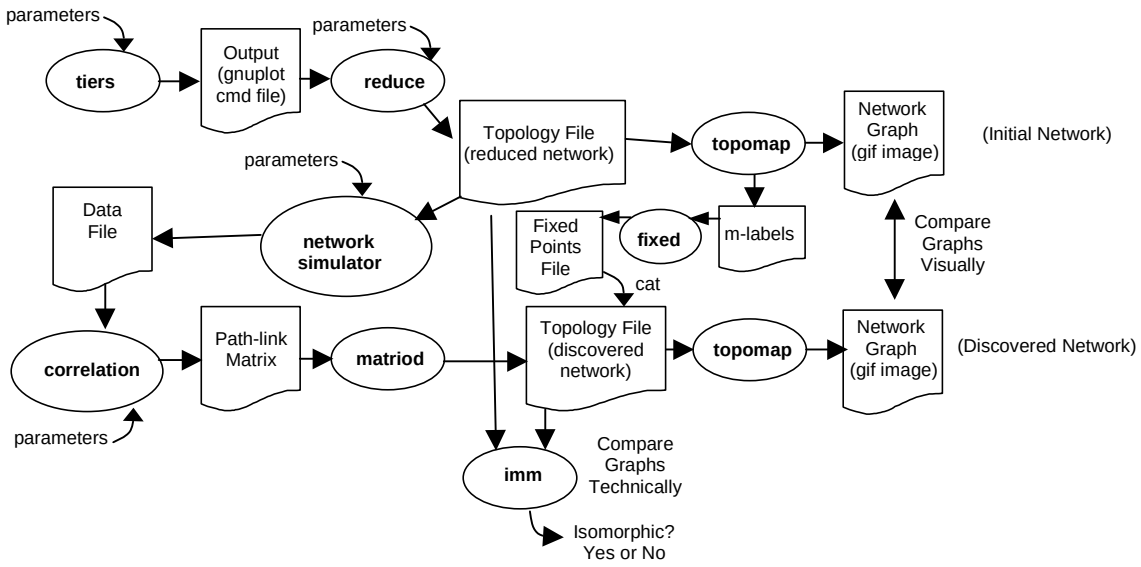


Figure 2-10: Code tool-chain. Ovals refer to programs and squares refer to files.

In Figure 2-10, we show how the various pieces of code interact. Ovals refer to programs and squares to files. Walking briefly through this chart, the *tiers* program generates the original network. This network is input to the *reduce* program that chooses monitors and removes non-traversed links and nodes, series links, and consecutive equivalent links. The *reduce* program captures the ideas of section 2.2. The output of the *reduce* program is a description of the reduced network, or a *topology file*. The *network simulator* then simulates a network model on the reduced network. End-to-end delay data is recorded by the network simulator and stored in a data file. This data file is input to the correlation algorithm, which attempts to solve for the path-link matrix. Finally, the matroid algorithm reconstructs the reduced network topology using the path-link matrix. The matroid algorithm outputs the discovered network as a *topology file*.

On the right-hand side of the figure, the reduced network is rendered and compared with the discovered network. *Topomap* is a graph drawing program that draws networks in a visually appealing manner. The purpose of *fix* is to place the monitors in the same locations in the diagrams of the reduced network and the discovered network. We can

compare the reduced network and the discovered network visually as well as with the graph isomorphism checker, *imm*.

These programs were all written in C. The primary authors of these programs are:

correlation- Jason Baron
fixed- Jason Baron
imm- Paul Seymour
matroid- Paul Seymour
network simulator- Jason Baron, Mark Garrett, Alex Poylisher
reduce- Jason Baron
topomap- Bruce Siegell

The algorithm timings referred to in the remainder of this paper were all performed on a Sun Ultra 5 running the SunOS 5.7 operating system with 64MB of RAM.

Chapter 3

Non-Overlapping Congestion

In this chapter, we consider the first network model. In section 3.1, we explain the assumptions and features of the network model. Section 3.2 introduces the idea of applying a threshold to the paths' end-to-end delay time series to produce a binary function. We find this to be a powerful technique and make use of it in solving this and subsequent network models. Section 3.3 presents two correlation algorithms—a simple algorithm that does not take some of the aspects of this network model into account and a more sophisticated algorithm that attempts to deal with these aspects. The results of testing the second algorithm are presented in section 3.4. Finally, we provide results and discussion in section 3.5.

3.1 Network Model

The network model consists of FIFO output queuing, no input queuing, and no processing delay at nodes. Therefore, there is a one-to-one correspondence between output queues and links. All links have identical bandwidth—1.55 Mbps.

The congestion aspects form the core of the model. Queues are congested by having packets of size 1250 bytes independently injected into them at a rate that is chosen from a heavy-tailed distribution[*]. After the injected packets have made their way to the front of the queue, they are simply discarded. The time for which a queue is injected with packets is fixed and this parameter is the *congestion_length*, which can be varied between simulation runs.

[*] We sample packet arrival rates from the VBR video trace used in [8]. This empirical data has a heavy-tailed distribution.

The key feature of this network model is that only one queue is being injected with packets at any given time. Queues are chosen to have packets injected into them at random. More precisely, if there are *k* queues and one queue has finished being injected with packets, then the probability that any one of the queues is chosen next for injection with packets is $1/k$. The time between the end of one queue being injected with packets and the start of a subsequent queue being injected with packets is the controlled by the *inter_congestion_length* parameter, which can be varied between simulation runs.

Monitors send time-stamped packets or *probes* of size 576 bytes to other monitors in a round-robin fashion. The frequency at which these packets are sent is inversely proportional to the number of monitors. Specifically, if monitor M_1 sends a probe packet to monitor M_2 at time *t*, then monitor M_1 will send its next probe packet to monitor M_3 at a time that is uniformly chosen between t and $t + (2/number\ of\ monitors)$. Packets are not sent at fixed intervals because we would like to detect events that may be happening periodically. As the size of the network grows, we increase the frequency that monitor packets are sent in order to maintain the rate at which probes measure network conditions. The consequence of generating probe packets in this manner is that if there are *m* monitors, then traffic may grow in some regions of the network by m^3.

Monitors that receive probes from other monitors calculate the end-to-end delay for a particular path by subtracting the time-stamp on the packet from the time that the packet was received. The receiving monitor then records the duple data: <time-stamp on received packet, end-to-end delay time>. Lost packets are ignored.

Two aspects of the model make finding the path-link matrix from the collected data difficult. If one could measure the underlying congestion aspects perfectly, then a *congestion event* or an increase in a path's end-to-end delay time would begin and end at almost the same time on those paths that contain the congested queue. However, since we are probing at somewhat random intervals, increases and decreases in end-to-end delay measurements do not line up between paths perfectly in time. We consider this to be an inherent measurement difficulty. A second difficulty arises if the *inter_congestion_time* is small. Since queues are being injected with packets according

to a heavy-tailed distribution, end-to-end delay times on paths that do not share any common queues may show correlation, despite the fact that only one queue is injected with packets at any given time.

Let us ignore the second difficulty for a moment. There is still a question of when to conclude that two paths share a common congestion event because of measurement uncertainty. We deal with this difficulty by applying a threshold to the end-to-end delay time series.

3.2 Applying a Threshold to the Delay Time Series

In order to decide if two paths share a common congestion event, we first apply a threshold to the end-to-end delay time series of each path. The result is a binary function that is 0 when the delay time series is below the threshold and 1 when the delay time series is above the threshold. To illustrate, the end-to-end delay time series for paths *p* and *q* are shown in Figure 3-1.

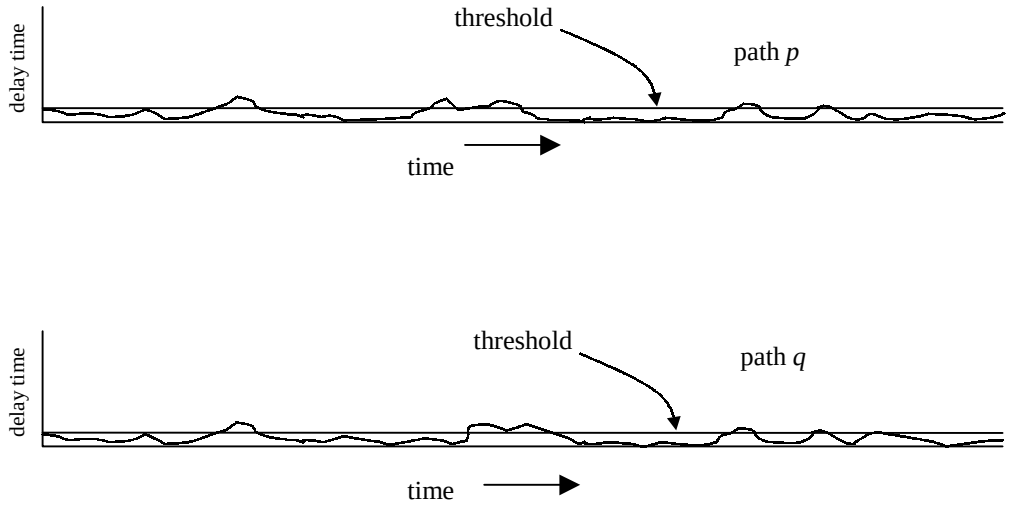


Figure 3-1: Applying a threshold. The end-to-end delay time series for paths *p* and *q* are shown with a threshold drawn on top. Note that this time series is only illustrative and is not meant to represent a recorded time series. In practice, the time series tends to be more punctuated and not as smooth as what is illustrated.

After applying a threshold, the binary functions that result are in Figure 3-2.

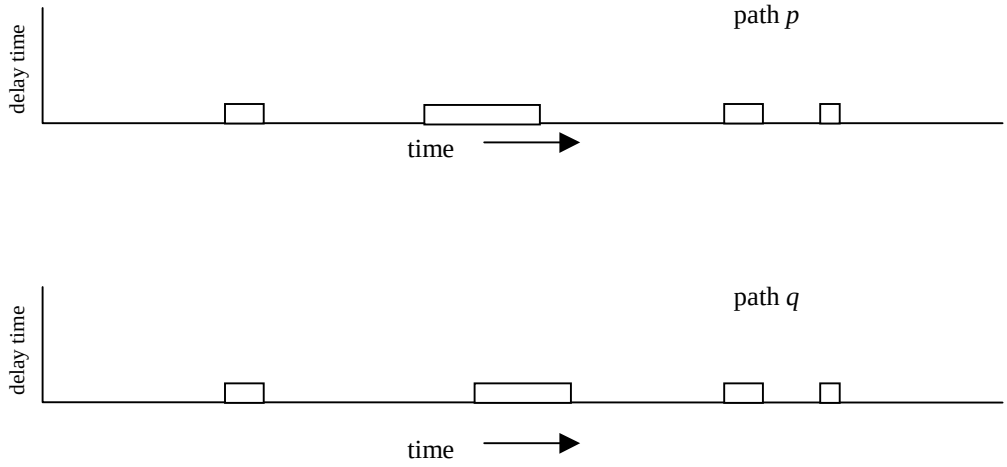


Figure 3-2: Binary functions. The binary functions that result from applying a threshold to the end-to-end delay time series of paths *p* and *q*.

In practice, we set the threshold at a level such that if no queues are congested on a path then the end-to-end delay time for that path should be below the threshold value. However, if at least one link congests on a path then the paths' end-to-end delay time should exceed the threshold value. We have set the threshold value to satisfy these properties by looking at the end-to-end delay time series. Setting the threshold in this manner certainly does not guarantee that the threshold will have its desired properties.

Based upon the binary functions, we can easily specify rules as to when a common congestion event is shared between paths. A rule might be: if the starting and ending times when two binary functions take on the value of 1 are similar, then this is a shared congestion event. Therefore, paths *p* and *q* share three congestion events. They might share a fourth event. However, the starting and ending times when the two binary functions are 1 are disparate due to measurement uncertainty.

There are certainly other techniques for identifying common congestion events on two paths. For example, one might consider using a continuous correlation function. We

have not explored the use of such techniques completely, but we show that applying a threshold to a delay time series is a powerful technique.

3.3 Algorithm

3.3.1 First Correlation Algorithm

Given the technique of applying a threshold to a path delay time series, a simple algorithm for determining the path-link matrix presents itself. Scan all paths over time and identify or *circle* the times when combinations of paths simultaneously take on the value 1. For example, a sample of the time series for the four paths *p*, *q*, *r*, and *s* is shown in Figure 3-3.

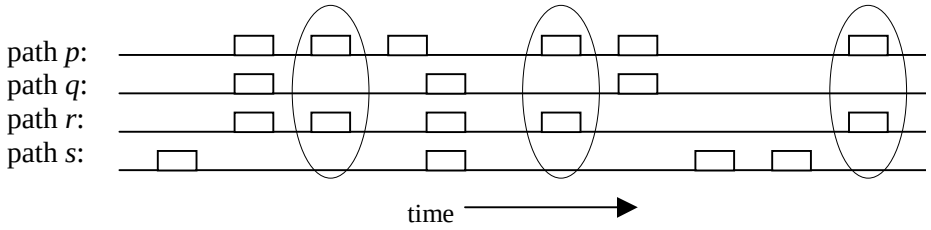


Figure 3-3: Circling algorithm. Times when only path *p* and path *r* share a congestion event are circled. If we continue circling all the unique sets of congestion events in this manner, we can construct the path-link matrix in a straightforward manner.

Here, we have circled the three instances when only paths *p* and *r* share a congestion event. If we continue to circle congestion events in this manner, then we can easily construct the path-link matrix. The columns in the path-link matrix correspond to the columns that we have circled. Paths in a circle that have a value of 1 are 1 in the corresponding position in the path-link matrix and 0 otherwise. The key assumption is that each link in the network is traversed by a unique set of paths and therefore any two circled sets of congestion events that have the same set of congested paths must have been caused by the same link in the network. It also follows that any two circles of

congestion events that have a different set of congested paths must have been caused by a different link in the network.

This algorithm, which we refer to as the *circling algorithm*, solves this congestion model to an extent. However, it does not address the two difficulties of the network model. First, if the *inter_congestion_length* approaches 0, it would start to become unclear as to what should be circled. Second, measurement uncertainty causes congestion events to not line up perfectly in time and thus circling becomes more difficult.

3.3.2 Second Correlation Algorithm

We now propose an algorithm that addresses the two difficulties of this model. The basic idea of the algorithm is to use pair-wise comparisons between paths that indicate whether or not two paths share a common cause of congestion. This pair-wise comparison takes many events into consideration, instead of just a single event as in the circling algorithm. The algorithm forms groups of paths that have at least one common link by adding one path at a time to a group through pair-wise comparisons. When the algorithm terminates, the groups correspond to columns in the path-link matrix. We must be careful in constructing these groups, since if there are *n* paths, then there are 2^n possible groups.

In deriving a pair-wise comparison test, we start with a model that contains two paths. Paths *p* and *q* are shown in Figure 3-4. We classify the links on these two paths as belonging to one of three categories: links on path *p* and not on path *q*, links on path *q* and not on path *p*, and finally links on both paths *p* and *q*. These three categories of links can be thought of as three causes of the congestion events that are observed on paths *p* and *q*. We represent the probability that at least one link congests in each of these three categories as: $P^t_{p\bar{q}}$, $P^t_{\bar{p}q}$, and P^t_{pq}. We refer to these three probabilities as *trigger* probabilities, since they can be thought of as triggering congestion events.

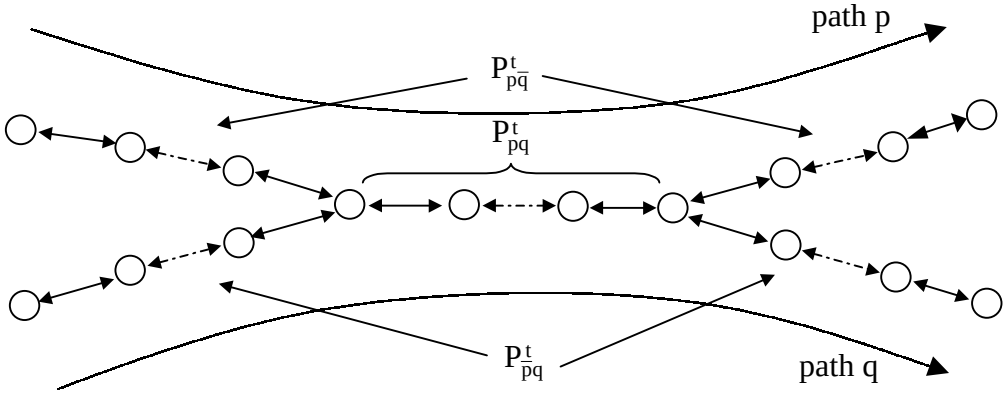


Figure 3-4: 2-path trigger probabilities. Links that are dashed are meant to represent zero or more links. $P^t_{p\bar{q}}$ is the probability that at least one of the links that is on path *p* and not on path *q* congests. $P^t_{\bar{p}q}$ is the probability that at least one of the links that is on path *q* and not on path *p* congests. Finally, P^t_{pq} is the probability that at least one link of the links common to both paths *p* and *q* congests.

There are four possible outcomes that we can observe, looking at the binary functions of both paths *p* and *q* at any instant in time. Path *p* might be 1 or 0 and path *q* might be 1 or 0. To calculate an estimate of the probability of any one of these four outcomes, one can divide the total time that each one of these four outcomes occur by the total observation time. We refer to the probability of each outcome as an *observation* probability. The notation that we use to designate the probability of each one of these four outcomes is: P^o_{pq}, if both paths' binary functions are 1, $P^o_{p\bar{q}}$, if path *p*'s binary function is 1 and path *q*'s binary function is 0, $P^o_{\bar{p}q}$, if path *p*'s binary function is 0 and path *q*'s binary function is 1, and $P^o_{\bar{p}\bar{q}}$, if both paths' binary functions are 0.

Assuming that the trigger probabilities are independent, the relationship between the observation probabilities and the trigger probabilities is the same here as in Ratnasamy *et al*. [17]. However, the underlying model that Ratnasamy uses is different from the model here. The Ratnasamy equations are:

$$P^{o}_{pq} = P^{t}_{pq} + (1 - P^{t}_{pq})P^{t}_{p\bar{q}}P^{t}_{\bar{p}q} \quad (3.1)$$

$$P^{o}_{p\bar{q}} = (1 - P^{t}_{pq})P^{t}_{p\bar{q}}(1 - P^{t}_{\bar{p}q}) \quad (3.2)$$

$$P^{o}_{\bar{p}q} = (1 - P^{t}_{pq})(1 - P_{p\bar{q}})P^{t}_{\bar{p}q} \quad (3.3)$$

The solution of P^{t}_{pq} from Ratnasamy is:

$$P^{t}_{pq} = \frac{P^{o}_{pq}P^{o}_{\bar{p}q} + P^{o}_{\bar{p}q}P^{o}_{p\bar{q}} + P^{o}_{p\bar{q}}P^{o}_{pq} + (P^{t}_{pq})^2 - P^{o}_{pq}}{P^{o}_{pq} + P^{o}_{\bar{p}q} + P^{o}_{p\bar{q}} - 1} \quad (3.4)$$

If the value of P^{t}_{pq} is significant, we conclude that two paths share a common congestion cause. We refer to testing two paths for a common cause as the P^{t}_{pq} test.

This pair-wise test addresses the two difficulties of the model as follows. First, it addresses the issue of queues being simultaneously congested because the model assumes that triggering events can happen simultaneously. Second, we assume that the measurement uncertainty is short relative to the duration of the *congestion_length* and thus the trigger probability estimates are not significantly affected. We have been a bit brief about this model because we will return to it in greater detail in chapter 4.

Assume that for two paths, *a* and *b*, the value of the P^{t}_{pq} test is significant and therefore we conclude that paths *a* and *b* share a common congestion cause. We form the group(*a*,*b*) and associate the *intersection* of path *a*'s binary function and path *b*'s binary function with this group. The intersection of two binary functions is 1 when both binary functions are 1 and 0 otherwise. The result of the intersection operation is a binary function that provides some information as to what congestion events the two paths have in common.

Now, we could test whether path *c* shares a common congestion cause with group(*a*,*b*) by applying the P^{t}_{pq} test again, treating the intersected binary function formed from paths *a* and *b* as one binary function and path *c*'s binary function as the other binary function.

If the P_{pq}^{t} test is not significant, then we continue to grow the group(*a*,*b*) by trying the P_{pq}^{t} test with additional paths. However, if the P_{pq}^{t} test is significant then we form the group(*a*,*b*,*c*) and associate the intersection of the binary functions of path *a*, *b*, and *c* with this group.

In deciding whether or not to keep the group(*a*,*b*), we *subtract* the binary function associated with group(*a*,*b*,*c*) from the binary function associated with the group(*a*,*b*). The subtraction operation is 1 when the minuend is 1 and the subtrahend is 0 and 0 otherwise. If we determine that there is a significant binary function left, meaning that there are a number of congestion events yet to be explained, then group(*a*,*b*) is kept. Otherwise, if this binary function is not significant, we delete the group(*a*,*b*). We also define a *unite* operation on two binary functions that is 1 when at least one binary function is 1 and 0 otherwise. Based upon the intersection, subtraction, and unite operation, and the two significance tests (the P_{pq}^{t} test and the test for detecting if significant congestion events remain), we construct the algorithm in Figure 3-5.

Input:

NUMBER_OF_PATHS; the number of paths
PATH_BINARY_FUNCTIONS : Array (Binary Function); PATH_BINARY_FUNCTION[i] is the binary function of path i

Output:

PATH_LINK_MATRIX: 2D Array; $x_{i,j} = \begin{cases} 1 & \text{if path i contains link j} \\ 0 & \text{otherwise} \end{cases}$

```
CORRELATE () {

   processed_events : Binary Function; initially contains no Events
   unexplained_events : Binary Function

   for path = 1 to NUMBER_OF_PATHS
      unexplained_events = PATH_BINARY_FUNCTIONS[path] subtract processed_events
      if unexplained_events is significant
         BREAK-UP(unexplained_events, path, (path+1))
      processed_events = processed_events unite PATH_BINARY_FUNCTIONS[path]

}


BREAK-UP(matching_events: Binary Function, group_paths : set of paths in this group,
lowest_candidate_path : minimum path that we try to add to this group) {

   intersect : Binary Function

   for path = lowest_candidate_path to NUMBER_OF_PATHS
      if P^t_pq test between matching_events and PATH_BINARY_FUNCTIONS[path] is significant
         intersect = matching_events intersect PATH_BINARY_FUNCTIONS[path]
         matching_events = matching_events subtract intersect
         BREAK-UP(intersect, group_paths ∪ path, (path+1))
         if matching_events is not significant
            return

   append 1 column to PATH_LINK_MATRIX
   for all i ∈ group_paths
      PATH_LINK_MATRIX[ i, numOfColumns(PATH_LINK_MATRIX)] = 1

}
```

Figure 3-5: Pseudocode of the correlation algorithm.

The input to the algorithm is the NUMBER_OF_PATHS, and the PATH_BINARY_FUNCTIONS. For convenience, we assume that the paths are numbered 1 through the NUMBER_OF_PATHS. Therefore, we index into the PATH_BINARY_FUNCTIONS array with a number between 1 and NUMBER_OF_PATHS.

The algorithm recursively forms a tree in a depth first manner. A group is associated with each node in the tree. Associated with a group are a set of paths, *group_paths*, a binary function, *matching_events*, and the *lowest_candidate_path*. We attempt to add all paths with a number greater than or equal to the *lowest_candidate_path* number to the group.

The procedure BREAK-UP is called on the node in the tree where we are progressing. The group associated with this node is the current group. The BREAK-UP procedure successively tries the P_{pq}^{t} test between the binary function associated with the current group, *matching_events*, and the binary function for all paths with a path number greater than the *lowest_candidate_path*. If this test is significant at any point, we form a *new* group.

The paths associated with the new group include all the paths in the current group and the path that was being tried when the P_{pq}^{t} test returned a significant result. Let us assume that we were trying to add path *i*, when the P_{pq}^{t} test returned a significant result. The binary function associated with the new group is formed via the intersection of the binary function that is associated with the current group and the binary function of path *i*. We also subtract the binary function that we have just associated with the new group from the current group's binary function. We call BREAK-UP next on this new group where we have set the new group's *lowest_candidate_path* to (i + 1).

Control is returned from the BREAK-UP procedure to the parent node in the tree in two cases. First, if the binary function associated with the current group is no longer significant. Second, when we have finished trying to add all paths greater than or equal to the *lowest_candidate_path* number to the current group and the congestion events

associated with the current group are significant. In this second case, we have identified a link and therefore we add a column to the path-link matrix.

The algorithm is initialized in the procedure CORRELATE by calling BREAK-UP on initial groups. We form one initial group for each path, initializing the set of paths that we associate with each initial group to contain that single path. The *lowest_candidate_path* is set to one more than the path number that is associated with each initial group. We initialize the binary function for the initial group that contains only the path *i* by subtracting all of the binary functions that have a path number that is less than *i* from path *i*'s binary function. This is analogous to subtracting the new group's binary function from the current group's binary function during a call to BREAK-UP. The root node of the tree can be thought of as containing all of the congestion events on all of the paths. By subtracting the binary functions of all paths that are numbered less than *i* from the binary function of path *i*, we are subtracting all congestion events that we have already explained.

3.3.3 Time and Space Complexity

We analyze the running time of the second correlation algorithm, assuming that the significant tests are always correct. Let:

M ≡ number of monitors
P ≡ number of paths = M ×(M – 1)
L ≡ number of links
N ≡ simulation length

The number of leaves in the tree that is formed is L and the number of nodes between the root and a leaf is at most P. Therefore, there are O(PL) nodes in the tree. We can also associate at most a single intersection and subtraction operation with each node in the

tree. The P^t_{pq} test takes time proportional to N. At each node in the tree we perform the P^t_{pq} test at most O(P) times. Therefore, we bound the running time of this algorithm as $O(P^2LN)$.

3.3 Results and Discussion

The algorithm performed well when the *inter_congestion_length* was large. However, as the *inter_congestion_length* was decreased towards zero, the algorithm did not necessarily find the correct path-link matrix. Here, we only present the largest network that was solved with the algorithm.

The reduced network has 30 monitors, 36 internal nodes, and 160 links. The correlation algorithm solved for the correct path-link matrix in 20 seconds using 80 minutes of simulated traffic. The *congestion_length* was 10 seconds and the *inter_congestion_length* was 5 seconds. The reduced network is shown in Figure 3-6 and the discovered network in Figure 3-7. The two networks are identical except for 8 split nodes that are mostly in the interior of Figure 3-7. When we show an example or data about the topology discovering algorithms, the correlation algorithm always finds the correct path-link matrix, unless otherwise noted. The split nodes are the result of the matroid algorithm.

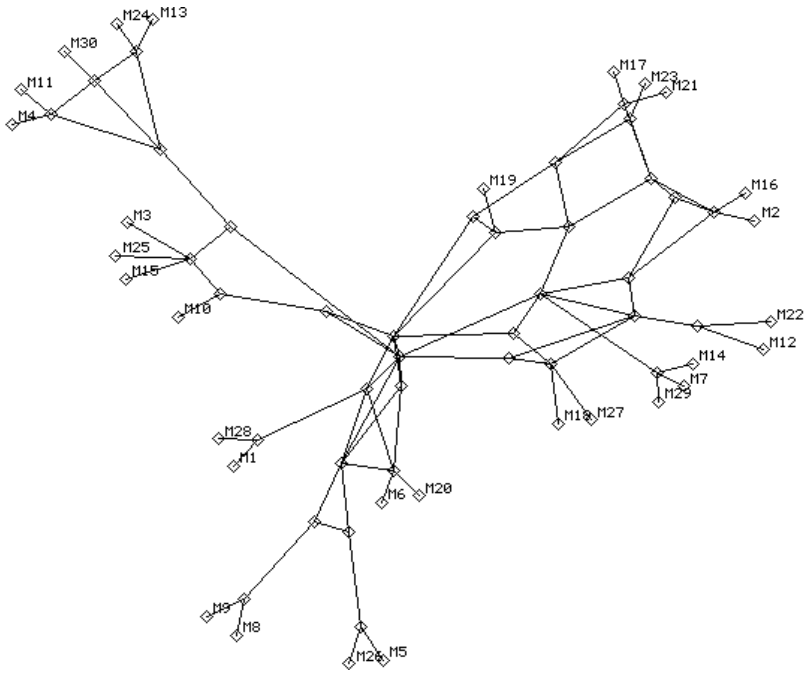


Figure 3-6: Reduced network. 30 monitors, 36 interior nodes, and 160 links.

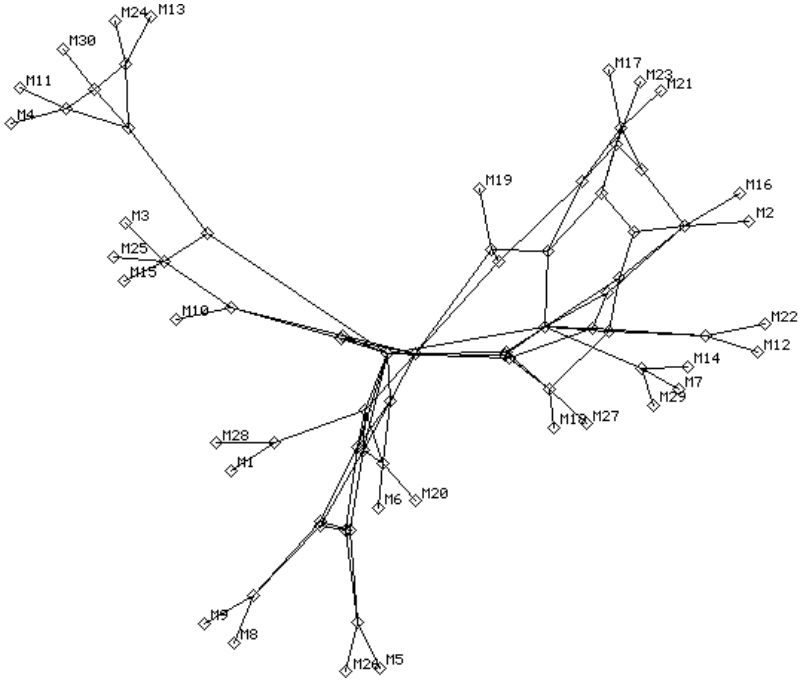


Figure 3-7: Discovered network. Network discovered by the matroid and correlation algorithms. This network is identical to the reduced network except for 8 split nodes. These split nodes appear as two nodes that are slightly offset. By comparing Figure 3-7 with Figure 3-6 the split nodes can be identified.

We have developed an algorithm that attempts to account for the difficulties of the non-overlapping congestion model. The algorithm performed well when the *inter_congestion_time* was large. However, as the *inter_congestion_time* approached 0 the correct path-link matrix was elusive. Despite the algorithm's shortcomings, the algorithm begins to capture ideas of how to deal with overlapping congestion events that are due to different queues and measurement uncertainty.

One problem with the algorithm is that the meaning of the P_{pq}^{t} test is unclear, when one of the binary functions is the result of intersected and subtracted binary functions. Another problem is deciding when the P_{pq}^{t} test is significant and when a significant number of congestion events remain in a binary function.

The subsequent network model has no measurement uncertainty. Instead, we focus on the difficulty that is caused when queues simultaneously congest. Thus, we isolate one of the difficulties of this network model.

Chapter 4

Overlapping Congestion in Discrete Time

The overlapping congestion in discrete time network model both simplifies and complicates aspects of the non-overlapping congestion network model. We introduce the overlapping congestion in discrete time network model in section 4.1. Section 4.2 presents a probability framework that forms the basis for the algorithm that is presented in section 4.3. Finally, in section 4.4, we provide results and discussion.

4.1 Network Model

In this network model, time is divided into discrete intervals. There is no input queuing at nodes, only output queuing. Therefore, there is a one-for-one correspondence between queues and links.

In each interval, each queue either does or does not congest. The probability that a queue congests in an interval is fixed at the same value for all intervals and is independent of all other queue congestion. Different queues may have different probabilities of congestion or *congestion rates*. We assume that the congestion rate on each link is greater than zero and that the congestion rate on the least congested link in the network is known. When a queue congests during a time interval *n*, all paths that traverse this queue are congested or *high* during time interval *n*. Therefore, a path *p* is congested during the time interval *n* if and only if at least one queue on path *p* congests during time interval *n*.

We can use a binary function to represent the congestion that occurs on a path over time. This binary function is 1 in time interval *n* if the path is congested and 0 otherwise. We refer to this binary function as a path's end-to-end delay time series.

This congestion model abstracts away some difficulties that we encountered in solving the non-overlapping model. Most importantly, all paths that contain a congested queue are all high simultaneously. There are no longer questions about whether or not congestion is temporally related. Thus, the measurement uncertainty of the non-overlapping model is no longer troublesome.

However, simultaneously congesting queues introduces new difficulties in determining the paths that traverse each link in a network. For instance, the sets of links on two separate paths in a network may be disjoint. However, queues on both of these paths might congest simultaneously. Consequently, there would appear to be a link in the network traversed by both paths. In order to begin to reason about this model, we introduce a probability framework in the next section.

4.2 Probability Analysis

4.2.1 Observation Probability

An observation probability is defined as the probability that specific paths are or are not congested. In order to formalize this notion, we introduce some random variables:

$$x_\lambda[n] = \begin{cases} 1 & \text{if link } \lambda \text{ is congested at time n} \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

Next, we define a random variable that is 1 if at least one link on path *p* is congested during interval *n* and 0 otherwise.

$$y_p[n] = \begin{cases} 1 & \text{if } \exists \lambda \in p \text{ such that } x_\lambda[n] = 1 \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

Now, we define a random variable that takes on the value 1 if from among a certain set of paths, some of these paths are congested, while the remaining paths in the set are not congested:

$$z_{p,\bar{q},r,\bar{s}}[n] = \begin{cases} 1 & y_p[n]=1, y_q[n]=0, y_r[n]=1, y_s[n]=0 \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

Note that a path without a bar above it refers to a path that is congested while a path with a bar above it refers to path that is not congested. We introduce the *observation* probability as:

$$P^{o}_{p,\bar{q},r,\bar{s}} = \Pr[z_{p,\bar{q},r,\bar{s}}[n]=1] \tag{4.4}$$

We let $P^{o}_{\bar{e}i}$ represent a general observation probability, where e is the set of *excluded* paths and *i* is the set of *included* paths. A path that is *excluded* from the observation probability is not congested and thus has a bar above it. A path that is *included* in the observation probability refers to a path that is congested and does not have a bar above it. A path that does not appear in the subscript of an observation probability is *ignored*.

In order to be clear about the meaning of an observation probability, we imagine that the end-to-end delay time series for paths *p*, *q*, *r*, and *s* are as shown in Figure 4-1.

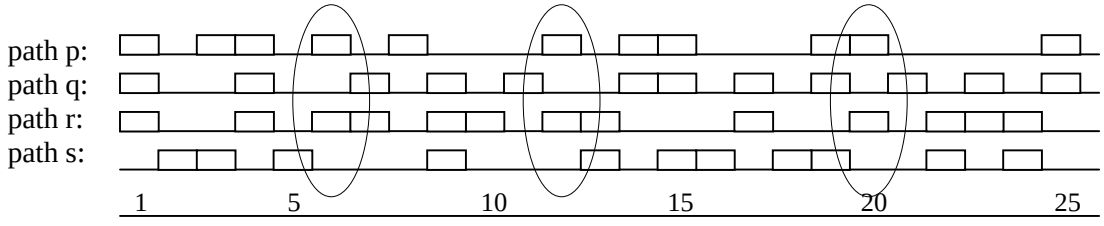


Figure 4-1: Calculating an observation probability. The binary functions of paths *p*, *q*, *r*, and *s* are shown. We have circled the times when paths *p* and *r* are congested and paths *q* and *s* are not congested. By dividing the number of times we have circled this event, 3, by the simulation length, 25, we estimate $P^{o}_{p,\bar{q},r,\bar{s}}$.

An estimator of $P^{o}_{p,\bar{q},r,\bar{s}}$ is:

$$\hat{P}^{o}_{p,\bar{q},r,\bar{s}} = \frac{\sum_{1}^{N} z_{p,\bar{q},r,\bar{s}}}{N} \tag{4.5}$$

Therefore, in the example of Figure 4-1, an estimate of $P^{o}_{p,\bar{q},r,\bar{s}}$ is $3/25$ or 0.12.

4.2.2 Trigger Probability

Estimates of the observation probabilities are calculated from the time series. However, these estimates are not informative as to individual link congestion rates. In order to estimate individual link congestion rates, we introduce the trigger probability.

A path *p* consists of a set of links. If all of the links in a given network are contained in the set U, then the complement of the set of links that constitute path *p* is defined as all of those links contained in the set U *and* not in the set *p*. We refer to this set of links, denoted $\bar{p}$, as path *p*'s *complementary* links. The intersection of path *p*'s links and path *q*'s complementary links is also a set of links. Therefore, thinking of a path as a set of links, we adopt the following notation:

$$p\bar{q}r\bar{s} = p \cap \bar{q} \cap r \cap \bar{s} \tag{4.6}$$

We define a random variable that is 1, if there is congestion on at least one link in the set that is defined by the intersection of certain paths' links and certain paths' complementary links.

$$z'_{p\bar{q}r\bar{s}}[n] = \begin{cases} 1 & \text{if at least one link in the set } p\bar{q}r\bar{s} \text{ is congested at time n} \\ 0 & \text{otherwise} \end{cases} \tag{4.7}$$

The trigger probability is defined as:

$$P^{t}_{p\bar{q}r\bar{s}} = \Pr[z'_{p\bar{q}r\bar{s}}[n]] = 1]] \tag{4.8}$$

We let $P^{t}_{\bar{e}i}$ denote a general trigger probability. A path that is *included* in the trigger probability does not have a bar above it and refers to the set of links on that path. A path that is *excluded* from the trigger probability has a bar above it and refers to that path's complementary links. A path that does not appear in a trigger probability subscript is *ignored*.

We assume that each link in the network is traversed by a unique set of paths. Therefore, we characterize a link by specifying whether or not each path traverses that link. Therefore, if there are *n* paths, then there are $(2^n\text{-}1)^*$ *possible* links. Some of the possible links are in the network and are *true* links, while some of the possible links are not in the network and are *false* links.

Through the intersection of the sets of links referred to by included and excluded paths, a trigger probability specifies a set of links that are *truly* in the network or the empty set. Assuming that we have *n* paths and we include or exclude each path in a trigger probability (we do not ignore any paths), then the trigger probability is *fully specified* and must refer to an individual link. This individual link could either be a true link or a false link. If it is a false link then the intersection of the sets of links referred to by the included and excluded paths is the empty set. There are $(2^n - 1)^{**}$ fully specified trigger probabilities that correspond to each of the possible links.

Assume that each link in the network congests with a probability greater than zero and that we observe the network for an infinite amount of time. Also, assume that we can calculate the value of all fully specified trigger probabilities. The trigger probabilities that refer to true links would have a value greater than zero, while the trigger probabilities that refer to false links would have a value of zero. The fully specified trigger

* We do not consider the link that is not traversed by any path.
** We do not consider the fully specified trigger probability that excludes all paths.

probabilities that refer to false links specify an empty set of links and therefore the probability of congestion is zero. In practice, trigger probabilities are not necessarily equal to their limiting value because of finite sample sizes. Although we do not observe the trigger probabilities directly, we can relate the trigger probabilities to the observation probabilities.

4.2.3 Relating Trigger Probabilities to Observation Probabilities

The sample network shown in Figure 4-2 has 5 monitors, 20 paths and 24 links. Let us relate the trigger probabilities to the observation probabilities for paths *p* and *q*, as shown.

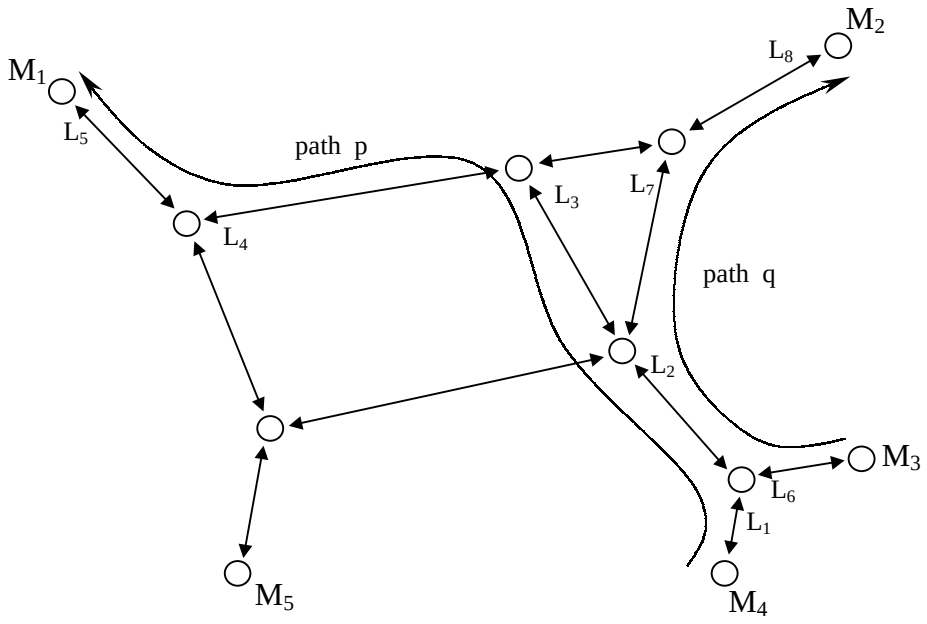


Figure 4-2: Relating trigger probabilities to observation probabilites for paths *p* and *q*.

Using the random variables that we have defined:

$$P^{o}_{\bar{p},\bar{q}} = \Pr[\, z_{\bar{p},\bar{q}}[n] = 1]$$

$$= \Pr[\, y_{p}[n] = 0, y_{q}[n] = 0]$$

The probability that there is no congestion on either path *p* or path *q* is:

$$= \Pr[x_{\lambda_1}[n] = 0, x_{\lambda_2}[n] = 0, x_{\lambda_3}[n] = 0, x_{\lambda_4}[n] = 0, x_{\lambda_5}[n] = 0, x_{\lambda_6}[n] = 0, x_{\lambda_7}[n], x_{\lambda_8}[n] = 0]$$

Making use of the fact that links congest independently:

$$= \Pr[x_{\lambda_2} = 0] \cdot \Pr[x_{\lambda_1} = 0, x_{\lambda_3} = 0, x_{\lambda_4} = 0, x_{\lambda_5} = 0] \cdot \Pr[x_{\lambda_6} = 0, x_{\lambda_7} = 0, x_{\lambda_8} = 0]$$

Finally, realizing that these three sets of links can be written as the following path intersections, we have:

$$= (1 - P^{t}_{pq})(1 - P^{t}_{p\bar{q}})(1 - P^{t}_{\bar{p}q})$$

This equation relates an observation probability to three trigger probabilities. For two paths, there are three trigger probabilities of interest: P^{t}_{pq}, $P^{t}_{\bar{p}q}$, and $P^{t}_{p\bar{q}}$. We are only interested in links that are on either path *p* or path *q* and therefore do not consider the trigger probability, $P^{t}_{\bar{p}\bar{q}}$. Expressing the observation probabilities $P^{o}_{\bar{p}}$ and $P^{o}_{\bar{q}}$ in terms of the three trigger probabilities of interest yields two more equations, giving us a total of three equations in three unknowns:

$$P^{o}_{\bar{p}\,q} = (1 - P^{t}_{pq})(1 - P^{t}_{p\bar{q}})(1 - P^{t}_{\bar{p}\,q}) \quad (4.9)$$

$$P^{o}_{\bar{p}} = (1 - P^{t}_{pq})(1 - P^{t}_{p\bar{q}}) \quad (4.10)$$

$$P^{o}_{\bar{q}} = (1 - P^{t}_{pq})(1 - P^{t}_{\bar{p}q}) \quad (4.11)$$

We refer to equations (4.9), (4.10), and (4.11) as the *2-path equations*. The form of the 2-path equations is similar to equations derived by Ratnasamy and McCanne [17]. Using the notation introduced here, the Ratnasamy equations can be written as:

$$P^{o}_{pq} = P^{t}_{pq} + (1 - P^{t}_{pq})P^{t}_{p\bar{q}}P^{t}_{\bar{p}q} \quad (4.12)$$

$$P^{o}_{p\bar{q}} = (1 - P^{t}_{pq})P^{t}_{p\bar{q}}(1 - P^{t}_{\bar{p}q}) \quad (4.13)$$

$$P^{o}_{\bar{p}q} = (1 - P^{t}_{pq})(1 - P_{p\bar{q}})P^{t}_{\bar{p}q} \quad (4.14)$$

The 2-path equations differ from the Ratnasamy equations in an important respect. Although, observation probabilities are only on the left-hand sides and trigger probabilities only on the right-hand sides of both sets of equations, the 2-path equations only involve observation probabilities that are the result of an absence of triggering events. In the Ratnasamy equations, observation probabilities reflect both the absence as well as a presence of triggering events. The right-hand sides of equations (4.9), (4.10), and (4.11) only multiply together *inverse* trigger probability terms that are of the form: $(1 - P^{t}_{\bar{e}i})$. On the other hand, the Ratnasamy equations contain terms of the form $(1 - P^{t}_{\bar{e}i})$ and $(P^{t}_{\bar{e}i})$ combined in a more complex manner. Although the model used in deriving the 2-path equations differs from the model used by Ratnasamy, the trigger probabilities are equivalent[*].

The 2-path equations are solved for the trigger probabilities as follows. The terms $P^{t}_{p\bar{q}}$ and $P^{t}_{\bar{p}q}$ are solved for by dividing equation (4.9) by equations (4.11) and (4.10),

[*] A proof of the equivalence of the trigger probabilities is found in the Appendix.

respectively. The term P^t_{pq} is obtained by substituting the values of $P^t_{p\bar{q}}$ and $P^t_{\bar{p}q}$ into equation (4.9). The solved trigger probabilities are:

$$P^t_{p\bar{q}} = 1 - \frac{P^o_{\bar{p}\bar{q}}}{P^o_{\bar{q}}} \tag{4.15}$$

$$P^t_{\bar{p}q} = 1 - \frac{P^o_{\bar{p}\bar{q}}}{P^o_{\bar{p}}} \tag{4.16}$$

$$P^t_{pq} = 1 - \frac{P^o_{\bar{p}} P^o_{\bar{q}}}{P^o_{\bar{p}\bar{q}}} \tag{4.17}$$

Assume that all of the links in Figure 4-2 congest at a rate greater than zero and the network is observed for an infinite amount of time. If any of the three trigger probabilities defined by equations (4.15), (4.16), and (4.17) are greater than zero, then we have not necessarily discovered an individual true link. Instead, some of these trigger probabilities might refer to a set of consecutive links. For instance, $P^t_{p\bar{q}}$ refers to the probability of congestion on at least one link of links L_1, L_3, L_4, and L_5, while $P^t_{\bar{p}q}$ refers to the probability of congestion occurring on at least one link of links L_6, L_7, and L_8. The term P^t_{pq}, however, corresponds to just one link, L_2. Therefore we could make the following statement:

$$P^t_{pq} = \Pr[x_{\lambda_2} = 1] \tag{4.18}$$

Thus, we have related a trigger probability to an individual link congestion rate.

Since we do not know the network topology *a priori*, we can not necessarily equate a trigger probability that is not fully specified with the congestion rate on an individual link. We must either include or exclude every path in a trigger probability in order to specify an individual link. However, a non-zero trigger probability that is not fully

specified must refer to at least one true link. A trigger probability that has a non-zero value and ignores some paths is a *true lead*, while a trigger probability that has a value of zero and ignores some paths is a *false lead*.

4.2.3.1 3-Path equations

We now extend the 2-path equations to three paths. Writing the equations in 3 sets according to the number of paths excluded by the observation probability on the left-hand side of the equation yields:

$$P^{o}_{\overline{pqr}} = (1-P^{t}_{pqr})(1-P^{t}_{pq\overline{r}})(1-P^{t}_{p\overline{q}r})(1-P^{t}_{\overline{p}qr})(1-P^{t}_{p\overline{qr}})(1-P^{t}_{\overline{p}q\overline{r}})(1-P^{t}_{\overline{pq}r}) \quad (4.19)$$

$$P^{o}_{\overline{pq}} = (1-P^{t}_{pqr})(1-P^{t}_{pq\overline{r}})(1-P^{t}_{p\overline{q}r})(1-P^{t}_{\overline{p}qr})(1-P^{t}_{p\overline{qr}})(1-P^{t}_{\overline{p}q\overline{r}}) \quad (4.20)$$

$$P^{o}_{\overline{pr}} = (1-P^{t}_{pqr})(1-P^{t}_{pq\overline{r}})(1-P^{t}_{p\overline{q}r})(1-P^{t}_{\overline{p}qr})(1-P^{t}_{p\overline{qr}})(1-P^{t}_{\overline{p}q\overline{r}}) \quad (4.21)$$

$$P^{o}_{\overline{qr}} = (1-P^{t}_{pqr})(1-P^{t}_{pq\overline{r}})(1-P^{t}_{p\overline{q}r})(1-P^{t}_{\overline{p}qr})(1-P^{t}_{p\overline{qr}})(1-P^{t}_{\overline{p}q\overline{r}}) \quad (4.22)$$

$$P^{o}_{\overline{p}} = (1-P^{t}_{pqr})(1-P^{t}_{pq\overline{r}})(1-P^{t}_{p\overline{q}r})(1-P^{t}_{p\overline{qr}}) \quad (4.23)$$

$$P^{o}_{\overline{q}} = (1-P^{t}_{pqr})(1-P^{t}_{pq\overline{r}})(1-P^{t}_{\overline{p}qr})(1-P^{t}_{\overline{p}q\overline{r}}) \quad (4.24)$$

$$P^{o}_{\overline{r}} = (1-P^{t}_{pqr})(1-P^{t}_{p\overline{q}r})(1-P^{t}_{\overline{p}qr})(1-P^{t}_{\overline{p}\overline{q}r}) \quad (4.25)$$

We have written seven independent equations in the seven unknown trigger probabilities of interest. The left-hand side of each equation contains an observation probability that only excludes paths. The right-hand side of each equation contains only fully specified inverse trigger probabilities that *contradict* the observation probability that is on the left-hand side. A trigger probability *contradicts* an observation probability if the trigger probability includes a path that is excluded by an observation probability.

We solve for the 7 trigger probabilities in order from the greatest number of excluded paths to the least number of excluded paths. A possible solution order for the trigger probabilities is thus $P^t_{\bar{p}q\bar{r}}$ followed by $P^t_{p\bar{q}\bar{r}}$, $P^t_{\bar{p}\bar{q}r}$, $P^t_{\bar{p}qr}$, $P^t_{p\bar{q}r}$, $P^t_{pq\bar{r}}$, and finally P^t_{pqr}. We refer to these trigger probabilities as trigger probabilities (1) through (7), respectively. Solving for trigger probabilities (1), (2) and (3) involves dividing equation (4.19) by equation (4.22), (4.21), and (4.20), respectively. For example, we solve for $P^t_{\bar{p}\bar{q}r}$ as:

$$P^t_{\bar{p}\bar{q}r} = 1 - \frac{P^o_{\bar{p}\bar{q}r}}{P^o_{\bar{p}\bar{q}}} \tag{4.26}$$

Next, we solve for the three trigger probabilities that exclude one path and include two paths. The procedure is analogous for all three of these trigger probabilities, and therefore we only solve for one of these trigger probabilities, $P^t_{\bar{p}qr}$. Dividing equation (4.19) by equation (4.23):

$$\frac{P^o_{\bar{p}\bar{q}r}}{P^o_{\bar{p}}} = \frac{\cancel{(1-P^t_{pqr})}\cancel{(1-P^t_{pq\bar{r}})}\cancel{(1-P^t_{p\bar{q}r})}(1-P^t_{\bar{p}qr})\cancel{(1-P^t_{p\bar{q}\bar{r}})}(1-P^t_{\bar{p}q\bar{r}})(1-P^t_{\bar{p}\bar{q}r})}{\cancel{(1-P^t_{pqr})}\cancel{(1-P^t_{pq\bar{r}})}\cancel{(1-P^t_{p\bar{q}r})} \qquad \cancel{(1-P^t_{p\bar{q}\bar{r}})}}$$

Thus,

$$\frac{P^o_{\bar{p}\bar{q}r}}{P^o_{\bar{p}}} = (1-P^t_{\bar{p}qr})(1-P^t_{\bar{p}q\bar{r}})(1-P^t_{\bar{p}\bar{q}r})$$

Solving for the trigger probability $P^t_{\bar{p}qr}$:

$$P^t_{\bar{p}qr} = 1 - \left(\frac{P^o_{\bar{p}\bar{q}r}}{P^o_{\bar{p}}}\right)\left(\frac{1}{(1-P^t_{\bar{p}q\bar{r}})(1-P^t_{\bar{p}\bar{q}r})}\right) \tag{4.27}$$

$P^t_{\overline{p}qr}$ is written as a function of observation probabilities and trigger probabilities that have already been computed. Finally, we solve for P^t_{pqr} by substituting all of the previously solved trigger probabilities into equation (4.19). Note that we could have solved for trigger probabilities (1), (2), and (3) in any order, since each of these trigger probabilities only depends upon observation probabilities. We could also solve for trigger probabilities (4), (5), and (6) in any order because these trigger probabilities only depend on observation probabilities and the trigger probabilities (1), (2) and (3).

4.2.3.2 n-Path Equations

We can generalize the path equations to the case of *n* paths. We write $(2^n\text{-}1)$ independent equations for the $(2^n\text{-}1)$ trigger probabilities of interest. The left-hand side of each equation contains one observation probability that only excludes paths. Thus, the $(2^n - 1)$ subsets of *n* excluded paths (we ignore the empty set), constitute the left-hand sides of the equations. The right-hand side of each equation multiplies together fully specified inverse trigger probability terms that contradict the observation probability that is on the left-hand side of the equation.

We generalize the solution technique used to solve the 3-path equations to solve for an arbitrary trigger probability, $P^t_{\overline{E}I}$. In the term $P^t_{\overline{E}I}$, E is the set of excluded paths and I is the set of included paths and $|I| + |E| = n$. We solve for the trigger probabilities in order from those that have the greatest number of excluded paths to those that have the least number of excluded paths. Therefore, when solving for the trigger probability $P^t_{\overline{E}I}$, we assume that we already have the values for all of the trigger probabilities that exclude more than $|E|$ paths. In solving for $P^t_{\overline{E}I}$, we consider two of the $(2^n\text{-}1)$ equations—the *master* equation and the *specific* equation.

Let N be the set of all paths. The master equation has $P^{o}_{\overline{N}}$ on its left-hand side (equation (4.19) is a master equation for *n* = 3). The right-hand side of the master equation multiplies together all of the fully specified inverse trigger probability terms because all of the fully specified trigger probability terms contradict the observation probability, $P^{o}_{\overline{N}}$. The specific equation, used in solving for $P^{t}_{\overline{EI}}$, has the term $P^{o}_{\overline{E}}$ on its left-hand side. The right-hand side of the specific equation only multiplies together fully specified inverse trigger probability terms that contradict the observation probability, $P^{o}_{\overline{E}}$. We classify the fully specified inverse trigger probability terms in both the specific and master equation into three classes. Using $P^{t}_{\overline{ei}}$ as a general trigger probability, the three classes of terms are: $|e| < |E|$, or class 1, $|e| = |E|$, or class 2, and $|e| > |E|$, or class 3.

In these terms, the master and specific equations have the following form:

Master equation:

$$P^{o}_{\overline{N}} = \underbrace{(1-P^{t}_{\overline{ei}})...(1-P^{t}_{\overline{ei}})}_{P^{t}_{\overline{ei}}\ \text{s. t.}\ |e|<|E|} \quad \underbrace{\bullet(1-P^{t}_{\overline{ei}})(1-P^{t}_{\overline{ei}})...(1-P^{t}_{\overline{EI}})}_{P^{t}_{\overline{ei}}\ \text{s. t.}\ |e|=|E|} \quad \underbrace{\bullet(1-P^{t}_{\overline{ei}})...(1-P^{t}_{\overline{ei}})}_{P^{t}_{\overline{ei}}\ \text{s. t.}\ |e|>|E|}$$

Specific Equation:

$$P^{o}_{\overline{E}} = \underbrace{(1-P^{t}_{\overline{ei}})(1-P^{t}_{\overline{ei}})...}_{P^{t}_{\overline{ei}}\ \text{s. t.}\ |e|<|E|} \quad \underbrace{\bullet(1-P^{t}_{\overline{ei}})(1-P^{t}_{\overline{ei}})...}_{P^{t}_{\overline{ei}}\ \text{s. t.}\ |e|=|E|,\ \text{except}\ (1-P^{t}_{\overline{EI}})} \quad \underbrace{\bullet(1-P^{t}_{\overline{ei}})(1-P^{t}_{\overline{ei}})...}_{P^{t}_{\overline{ei}}\ \text{s. t.}\ |e|>|E|\ \text{and}\ i\cap E\neq\varnothing}$$

All trigger probabilities in class 1 are included in the specific equation. For all of the trigger probabilities in class 1, no matter how the $|e|$ excluded paths are chosen there must be at least one path in the set *E* that is not in the set *e*. Therefore, there is at least one path in the set *E* that is also in the set *i*. Thus, all terms in class 1 contradict $P^{t}_{\overline{E}}$ and

must be included in the specific equation. All terms from class 2 are also included, except for the term $P^{t}_{\overline{EI}}$. The term $P^{t}_{\overline{EI}}$ does not contradict $P^{o}_{\overline{E}}$. However, all other terms in class 2 contradict $P^{o}_{\overline{E}}$ because there is no way of picking *i* without including at least one path that is in E. Finally, it is more difficult to generalize about the nature the terms included from class 3. However, we again include only those trigger probabilities from class 3 that contradict the observation probability, $P^{o}_{\overline{E}}$.

Solving for $P^{t}_{\overline{EI}}$ involves dividing the master equation by the specific equation, yielding the *quotient* equation.

Quotient Equation:

$$\frac{P^{o}_{\overline{N}}}{P^{o}_{\overline{E}}} = \underbrace{\qquad}_{\text{all terms cancel}} \underbrace{\bullet (1-P^{t}_{\overline{EI}})}_{\text{only } P^{t}_{\overline{EI}} \text{ remains}} \underbrace{\bullet (1-P^{t}_{\overline{ei}})...(1-P^{t}_{\overline{ei}})}_{P^{t}_{\overline{ei}} \text{ s. t. } |e| > |E| \text{ and } i \cap E = \varnothing}$$

In both the master and specific equation, class 1 terms are identical and therefore there are no terms from class 1 in the quotient equation. The terms from class 2 are identical in both the master and specific equations, except the term $(1-P^{t}_{\overline{EI}})$, which is in the master equation and not in the specific equation. Therefore, the only term from class 2 in the quotient equation is: $(1-P^{t}_{\overline{EI}})$. Finally, the master equation includes all of the trigger probability terms from class 3, while the specific equation includes only those terms from class 3 that contradict the observation probability, $P^{o}_{\overline{E}}$. Therefore, the quotient equation contains only those trigger probabilities from class 3 that do not contradict the observation probability, $P^{o}_{\overline{E}}$. However, we assume that we have already solved for all trigger probabilities where $|e| > |E|$ and thus we can solve for $P^{t}_{\overline{EI}}$.

We write the general form of a trigger probability as:

$$P^t_{\overline{EI}} = 1 - \left(\frac{P^o_{\overline{N}}}{P^o_{\overline{E}}} \right) \left(\frac{1}{\prod_{P^t_{\overline{ei}} \in S} (1 - P^t_{\overline{ei}})} \right), \text{S contains all } P^t_{\overline{ei}} \text{ such that } |e| > |E| \text{ and } i \cap E = \varnothing \quad (4.28)$$

This general trigger probability formula applies to all of the trigger probabilities, except for the trigger probability that does not exclude any paths. We represent a general trigger probability that does not exclude any paths as P^t_I, where $|I| = n$. The value of this trigger probability is obtained directly from the master equation. We substitute the values for all of the trigger probabilities in the master equation and then solve the equation for P^t_I. The general form of the trigger probability that does not exclude any paths is:

$$P^t_I = 1 - \left(\frac{P^o_{\overline{N}}}{\prod_{P^t_{\overline{ei}} \in S} (1 - P^t_{\overline{ei}})} \right), \text{S contains all } P^t_{\overline{ei}} \text{ such that } |e| > 0 \quad (4.29)$$

We now turn to the algorithm that makes use of these trigger probabilities.

4.3 Algorithm

It is not computationally feasible to compute all of the fully specified trigger probabilities. For example, in a network that has 10 monitors, there are 90 paths and thus 2^{90} fully specified trigger probabilities. The total number of trigger probabilities grows exponentially in the number of paths.

We significantly reduce the number of trigger probabilities that need to be calculated by making use of true and false leads. The idea is to start by calculating the two trigger

probabilities that ignore all paths except for one: P^t_a and $P^t_{\bar{a}}$. If either of these trigger probabilities is non-zero, or a true lead, then we add path *b* as both included and excluded to both of these trigger probabilities. Assume that P^t_a and $P^t_{\bar{a}}$ are true leads. Thus, we calculate: P^t_{ab}, $P^t_{a\bar{b}}$, $P^t_{\bar{a}b}$, and $P^t_{\bar{a}\bar{b}}$. Now, assume that $P^t_{a\bar{b}}$, $P^t_{\bar{a}b}$, and $P^t_{\bar{a}\bar{b}}$, are true leads, but P^t_{ab} is a false lead. We calculate $P^t_{a\bar{b}c}$ and $P^t_{a\bar{b}\bar{c}}$ for $P^t_{a\bar{b}}$, $P^t_{\bar{a}bc}$ and $P^t_{\bar{a}b\bar{c}}$ for $P^t_{\bar{a}b}$, and $P^t_{\bar{a}\bar{b}c}$ and $P^t_{\bar{a}\bar{b}\bar{c}}$ for $P^t_{\bar{a}\bar{b}}$. However, we do not add any paths as either included or excluded to P^t_{ab}.

In this manner, the algorithm forms a binary tree (see Figure 4-7), where we calculate trigger probabilities that have excluded and included a total of *d* paths at depth *d*. Each horizontal level in the tree corresponds to solving the d-path equations. If there are *k* true leads at depth *d*, then we calculate (2*k*) trigger probabilities at depth (*d* + 1). Thus, we *prune* the complete binary tree by following true leads and ignoring false leads.

By not calculating certain trigger probabilities, we no longer solve the full set of d-path equations at depth *d*. Trigger probabilities that are not calculated are approximated as zero. In the path equations, this corresponds to setting the inverse trigger probability terms, $(1 - P^t_{\bar{e}i})$, to 1. Thus, we start with the full set of path equations at each level, set some of the $(1 - P^t_{\bar{e}i})$ terms to 1, and then solve what remains.

Since the simulation is run for a finite length of time, the estimates of the observation probabilities have not necessarily converged to their limiting value. Therefore, setting many of the trigger probabilities to zero is an approximation. Thus, the calculated values for the trigger probabilities are approximations as well.

A trigger probability that has a value that is greater than zero is not necessarily a true lead because trigger probabilities are only approximations of their limiting values. Therefore, instead of using zero as a decision threshold, we choose another value, the *pruning_threshold*. We redefine a *true lead* as a trigger probability that ignores some paths and refers to at least one true link and a *false lead* as a trigger probability that ignores some paths and refers to the null set.

The pseudocode of the algorithm is divided into two procedures. The procedure CALCULATE-TRIGGER-PROBABILITY, in Figure 4-3, calculates the value of a trigger probability according to equations (4.28) and (4.29). Figure 4-4 contains the procedure, CORRELATE, which captures the algorithm that was just described.

Input:

$P^{t}_{\bar{E}I}$

SIGINIFICANT_TRIGGER_PROBABILITIES

Output:

value of $P^{t}_{\bar{E}I}$

CALCULATE-TRIGGER-PROBABILITY () {

trigger_value = 1

if the set E is empty

observation_value = P^{o}_{I}

else

observation_value = $\dfrac{P^{o}_{\bar{E}I}}{P^{o}_{\bar{E}}}$

for $P^{t}_{\bar{e}i} \in$ PREVIOUS_TRIGGER_PROBABILITIES

if $i \cap E = \varnothing$

trigger_value = trigger_value $\times (1 - P^{t}_{\bar{e}i})$

$P^{t}_{EI} = 1 - \left(\dfrac{\text{observation\_value}}{\text{trigger\_value}} \right)$

}

Figure 4-3: Pseudocode of the calculate-trigger-probability procedure.

Input:

NUMBER_OF_PATHS

Output:

PATH_LINK_MATRIX

CORRELATE () {

next_triggers = $\left\{ P_{\bar{1}}^{t}, P_{1}^{t} \right\}$

for d = 1 to NUMBER_OF_PATHS

triggers = next_triggers
set next_triggers empty
set significant_triggers empty
$d' = d + 1$

for $P_{\bar{e}i}^{t} \in$ triggers (successive $P_{\bar{e}i}^{t}$'s are selected from the triggers such that $|e|$ monotonically decreases)

CALCULATE-TRIGGER-PROBABILITY ($P_{\bar{e}i}^{t}$, significant_triggers)

if $P_{\bar{e}i}^{t}$ > prunning_threshold

significant_triggers = $P_{\bar{e}i}^{t} \cup$ significant_triggers

if d < number_of_paths

next_triggers = next_triggers $\cup P_{\bar{e}id'}^{t} \cup P_{\bar{e}i\,\bar{d}'}^{t}$

else

add a new column to the PATH_LINK_MATRIX
set rows indexed by the paths in the set e to 0 in the new column
set rows indexed by the paths in the set i to 1 in the new column

}

Figure 4-4: Psuedocode of the correlation algorithm.

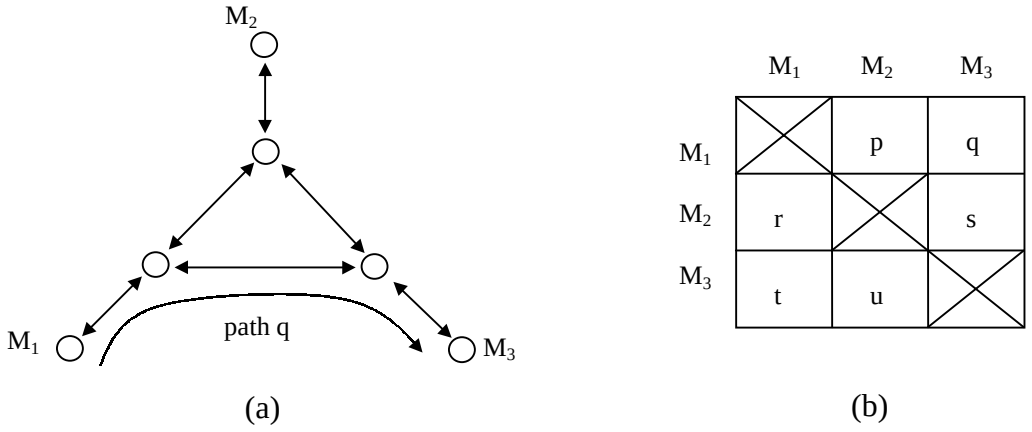


Figure 4-5: Sample network topology for demonstrating the correlation algorithm. The table in Figure 4-5(b) identifies the path names. For example, path M_1-M_3 is path *q*.

We examine the algorithm's behavior on the network shown in Figure 4-5(a). Figure 4-5(b) indicates the names of the paths. For instance, path *q* is the path from monitor M_1 to M_3. The binary tree that might result from running the algorithm is shown in Figure 4-7.

The light-colored lines in Figure 4-7 indicate areas where that algorithm no longer needs to search. At each level, the trigger probabilities are solved in order from most excluded paths to least excluded paths and not in the order that the trigger probabilities appear. The surviving leaves of the tree are the trigger probabilities that refer to true links. Here, we have successfully identified all 9 links and construct the path-link matrix shown in Figure 4-6.

	L_1	L_2	L_3	L_4	L_5	L_6	L_7	L_8	L_9
P	1	1	x	X	x	x	X	X	X
Q	x	X	1	1	x	x	X	X	X
R	x	X	x	X	1	1	x	X	X
S	x	X	1	X	x	x	1	X	X
T	x	X	x	X	1	x	x	1	X
U	1	X	x	X	x	x	x	X	1

Figure 4-6: Path-link matrix for the network shown in Figure 4-5.

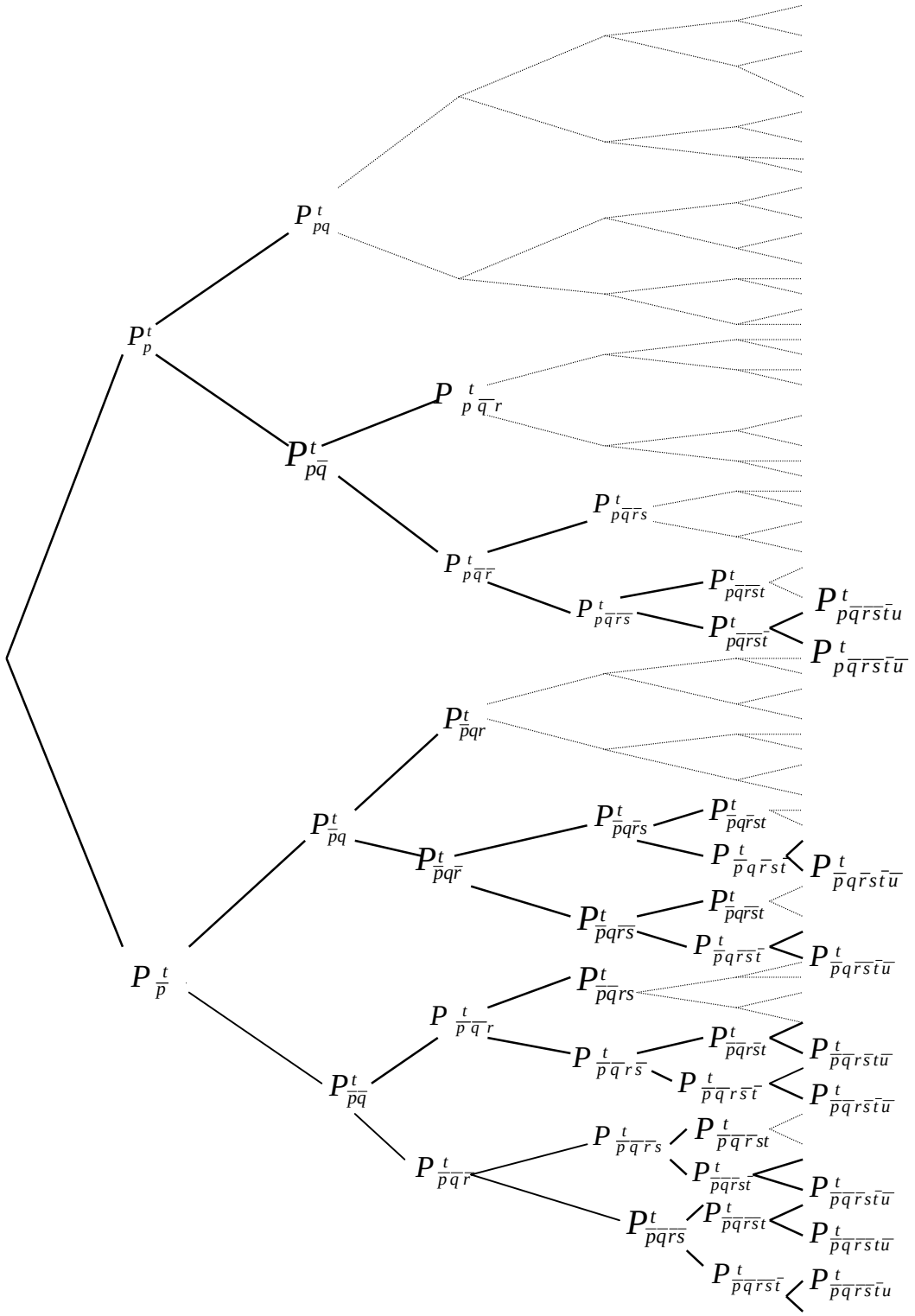


Figure 4-7: Binary tree constructed by the correlation algorithm. Light-colored lines are branches of the binary tree that the algorithm does not search down.

4.3.1 *Pruning_threshold*

We now examine the *pruning_threshold* in more detail. At any depth in the binary tree, there are some true leads and some false leads. We do not know which trigger probabilities are true leads and which trigger probabilities are false leads. The distribution of true and false leads at an arbitrary depth in the tree might look like Figure 4-8(a) or Figure 4-8(b):

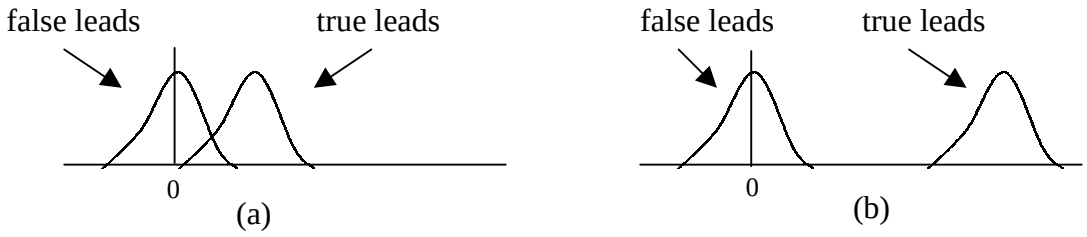


Figure 4-8: Distribution of the true and false leads. In 4-8(a) the true and false leads overlap, while in 4-8(b) the true and false leads could be separated with a threshold.

The distributions in Figure 4-8(a), can not be separated with a threshold value. However, the distributions in Figure 4-8(b) can be separated with a threshold value.

The *pruning_threshold* is based on the assumption that each link in the network congests at a rate that is greater than zero. If a link congested at a rate of zero, then we would have no way of separating true leads from false leads. Therefore, the model assumes a minimum link congestion rate that is greater than zero.

Additionally, we assume that the rate of congestion on the least congested link in the network is known. True leads have a value that is at least the estimated rate of congestion on the least congested network link, since a true lead refers to at least one true link. As previously discussed, false leads are approximately zero. Therefore, a logical value of the *pruning_threshold* is somewhere between zero and the minimum link congestion rate.

A poorly set *pruning_threhold* causes several difficulties. If the *pruning_threshold* is set too low such that there are false leads to the left and right of the *pruning_threshold*,

then false leads are included in the trigger probability calculations. Including false leads in the computation may improve the estimates of the trigger probabilities as we show in chapter 5. However, including false leads increases the running time of the algorithm and we still have the problem of separating the true links from the false links at the last level. Another problem arises if the *pruning_threshold* is set too high such that there are true leads to the left and right of the *pruning_threshold*. If this occurs, then at least one true link will be absent from the path-link matrix. We would like to detect if the *pruning_threshold* is poorly set or if true and false leads overlap as in Figure 4-8(a).

We start by treating values that are greater than the *pruning_threshold* as belonging to one distribution and the values that are less than the *pruning_threshold* as belonging to another distribution. We refer to the values that are greater than the *pruning_threshold* as distribution 1 values and the values that are less than the *pruning_threshold* as distribution 2 values. We define the mean and standard deviation of distribution 1 and distribution 2 as μ_1 and σ_1, and μ_2 and σ_2, respectfully. We define η using the following two equations:

$$\tau = \mu_1 - \eta\sigma_1 \tag{4.30}$$

$$\tau = \mu_2 + \eta\sigma_2 \tag{4.31}$$

A high value of η suggests that the two distributions are well separated, while a low value of η might suggest that the *pruning_threshold* is poorly set or that the true and false leads overlap as in Figure 4-8(a). An assessment of η as an indicator of well separated distributions is found in section 4.4.1.

4.3.2 Time and Space Complexity

Let:

M ≡ number of monitors

P ≡ number of paths = M × (M – 1)

L ≡ number of links

N ≡ simulation length

Assuming that the algorithm always follows true leads and ignores false leads, the analysis of the algorithm's running time is straightforward. The depth of the binary tree that the algorithm explores is P. At depth *d* in the tree, we add path *d* as both excluded and included to all trigger probabilities that are above the *pruning_threshold* at depth (*d* – 1). When the algorithm starts, *d* is set to 1. The algorithm increases *d* by one in each iteration until *d* = P.

At any depth in the search tree, there is at most one true lead for each true link. The number of false leads at level *d* is at most the number of true leads at level (*d* - 1). Therefore, the width of the tree is O(L), and the number of nodes in the tree is O(LP).

At each node in the tree, we calculate a trigger probability as detailed in the CALCULATE-TRIGGER-PROBABILITY procedure. Two observation probabilities are required. The time required to calculate an observation probability is O(N) (using the memory model discussed below). We also need to search through each link as specified in the CALCULATE-TRIGGER-PROBABILITY procedure. This requires O(L) time. Thus, the time required at each node is: O(N+L). Therefore, the time complexity of the algorithm is: $O(LPN+L^2P)$.

We implemented several different memory models for this algorithm that trade off memory and time. A rather straightforward memory implementation produces the O(N) bound for calculating each observation probability. Notice that the algorithm only requires observation probabilities that exclude all paths. Therefore, to calculate an arbitrary observation probability, $P^o_{\bar{e}}$, we start by inverting all of the binary functions of the paths that are in the set *e*. This involves a logical not, since the binary functions are represented as a string of 0's and 1's. Next, we *intersect* the inverses of all of the binary functions using the logical binary operator, *and*. The result of this operation is a single binary function that is $P^o_{\bar{e}}$'s *signal*. Finally, we obtain the value of $P^o_{\bar{e}}$ by adding up the number of 1's in $P^o_{\bar{e}}$'s signal and dividing by the simulation length, N.

We associate a signal with the current *leaves* in the binary tree. Thus, it is easy to compute the signal and the value of the observation probability for the children of these *leaves*. At depth *d*, we create a right-child by adding path *d* as excluded and a left-child by adding path *d* as an included to the observation probability of a *leaf* node. When we add path *d* as excluded to the observation probability of a leaf node, we need only intersect the inverse of path *d*'s binary function with the signal that is associated with the leaf node. This operation results in a *new* signal. The value of the observation probability for the right-child is calculated directly from the new signal. Creating a left-child involves copying the signal and value of the observation probability of the *leaf* node directly to the child node. Therefore, the time required to calculate an observation probability at each node is O(N).

Under this memory model, O(LN) space is required to store the signals and O(PN) space for storing the original observations. Therefore, we conclude that this memory model requires O(N(P+L)) space.

4.4 Results and Discussion

In section 4.4.1, we explain how the *pruning_threshold* was set and propose a method for setting the *pruning_threshold* that does not assume knowledge of the congestion rate on the least congested network link. We also briefly examine the reliability of η. Section 4.4.2 explores the running time of the algorithm. Section 4.4.3 shows how the algorithm performed when the link congestion rates were varied, and section 4.4.4 presents the largest networks that were solved. Finally, section 4.4.5 summarizes our findings and points the way towards the subsequent network model.

4.4.1 *Pruning_threshold* and η

If the observation probabilities converge to their limiting values, then false leads are zero and true leads have a value that is at least the rate of congestion on the least

congested link in the network. We assume that the minimum congestion rate among all of the links in the network is known and set the *pruning_threshold* at 40% of this congestion rate. We choose 40% because it is natural to set the *pruning_threshold* approximately halfway between the minimum link congestion rate and zero, while preferring to include false links rather than excluding true ones. For example, if the rate of congestion on the least congested network link is .1%, then the *pruning_threshold* is set at .04%.

In general, however, one does not know the minimum link congestion rate in a network. We propose the following modification to the algorithm that does not assume knowledge of the minimum link congestion rate. After we have calculated the trigger probabilities at a certain depth, we sort the calculated trigger probability values. Then, we establish the false lead distribution with the first few of the lowest trigger probability values and the true lead distribution with the first few of the highest trigger probability values. We then alternately pick the remaining highest and lowest trigger probability values and assign them to the *closer* of the two distributions. The number of standard deviations away from the mean of each distribution could be used as a measure of distance. The value of η might again be used to determine if we are in troubled case.

We only tested the algorithm using a *pruning_threshold* that was set *a priori*. Given this method of setting the *pruning_threshold*, we briefly investigated η as to its ability to indicate when the *pruning_threshold* was poorly set and when the distributions of true and false leads overlapped. We do not have any conclusive data about the effectiveness of η, but in varying many simulation parameters, we found that a value of η that was greater than 5.5 always yielded the correct path-link matrix. As a sanity check, we also verified that as the simulation was run for longer lengths of time the value of η increased.

4.4.2 Running Time

In assessing the running time of the algorithm, we use η as a proxy for the quality of the result. Since we are interested in the topology of the network, increasing the length of the simulation increases the running time of algorithm while possibly leaving the answer unchanged. Therefore, we use η to normalize the results.

Previously, we formulated the running time of the algorithm as $O(LPN+L^2P)$. Here, we investigate how the simulation length, N, increases as a function of the size of the network for constant η. The most descriptive characteristics of the size of a network are the number of links and the number of paths.

In the log-linear plot shown in Figure 4-9, the simulation length, N, is roughly a straight line with respect to the number of links. Thus, the length of the simulation time required to solve for the path-link matrix is exponential in the size of the network.

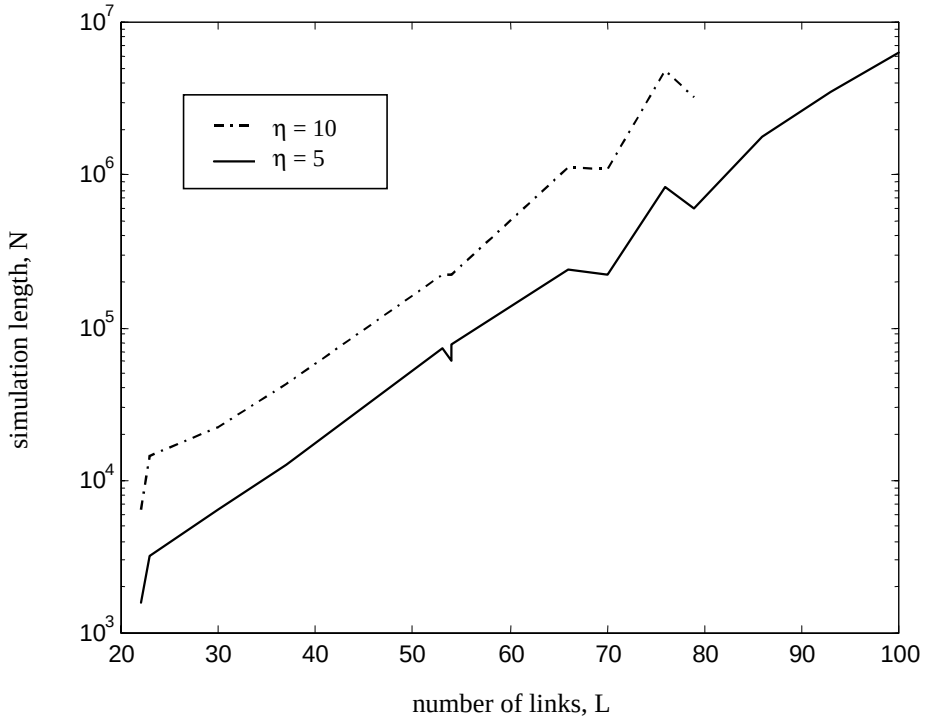


Figure 4-9: Simulation length as a function of the number links. The congestion rate on all links is 10%.

A good fit for the simulation length as a function of the number of links and paths in the network is:

$$N = e^{L+P} \tag{4.32}$$

The R^2 value of the OLS fit of this function was .9937, suggesting that this functional form explains almost all of variation in the simulation length [22].

In Figure 4-10, we plot the running time of the correlation algorithm and the network simulator as a function of the number of links in the network. The time required for both programs grows exponentially as we would expect.

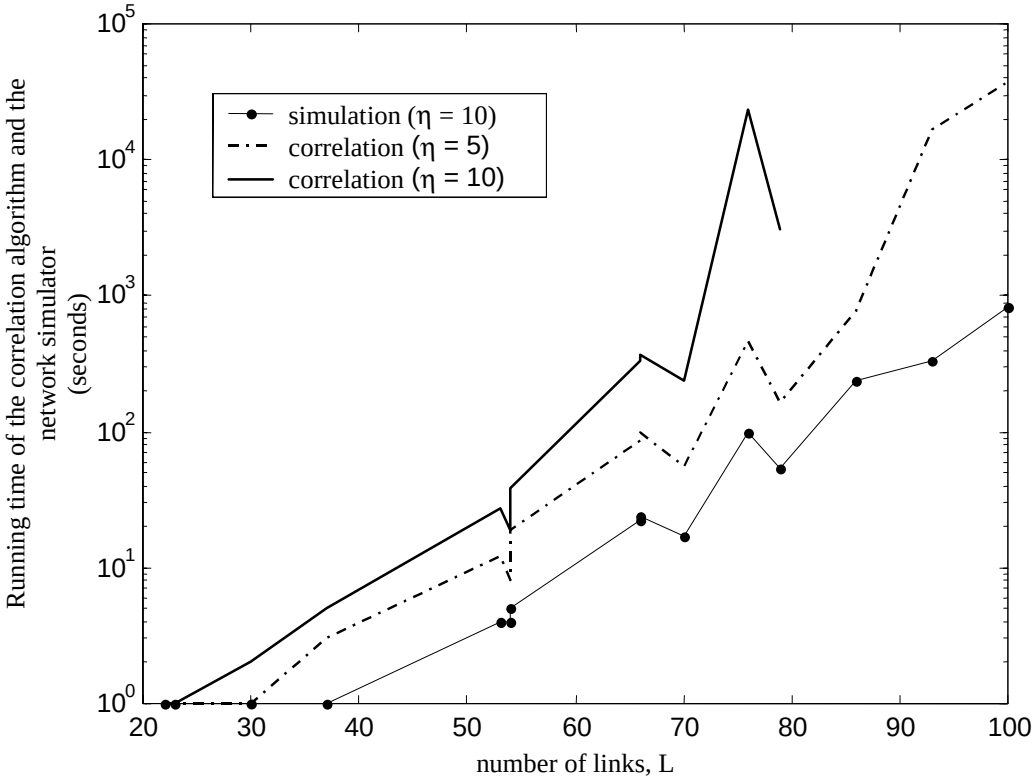


Figure 4-10: Running time of the correlation algorithm and the network simulator as a function of the number of links. Time is measured in seconds. The congestion rate on all links is 10%.

4.4.3 Varying the Link Congestion Rates

In assessing the running time of the algorithm, all links were congested at the same 10% rate. In this section, we explore the convergence time of the correlation algorithm when the link congestion rates varied. For all data points in Figure 4-11, we use the network shown in Figure 4-12.

Congestion rate	Simulation length	η	Simulation time (sec.)	Correlation time (sec.)
.20	15,360,000	3.9	1,096	10,294
.15	960,000	5.5	112	72
.10	64,000	5.1	4	8
.05	9,600	5.3	1	6
.01	3,200	4.7	1	5
.005	6,400	4.7	1	5
.002	19,200	5.4	2	6
.001	25,600	4.7	2	7
.0001	288,000	4.7	17	33
.00002	1,152,000	5.3	68	124
.01-.05	9,600	2.3	1	5
.01-.1	320,000	2.2	21	27
.01-.15	1,920,000	2.3	124	150
.0001-.001	320,000	2.0	20	34
.0001-.005	960,000	2.1	58	106
.0001-.01	3,200,000	1.7	197	1964

Figure 4-11: Varying the link congestion rates. Table showing the effects of varying the link congestion rates on the correlation algorithm's convergence time. A single number for the link congestion rate means that all links in the network congested at that rate. A range for the link congestion rate means that the links were congested a rate that was chosen uniformly from the range.

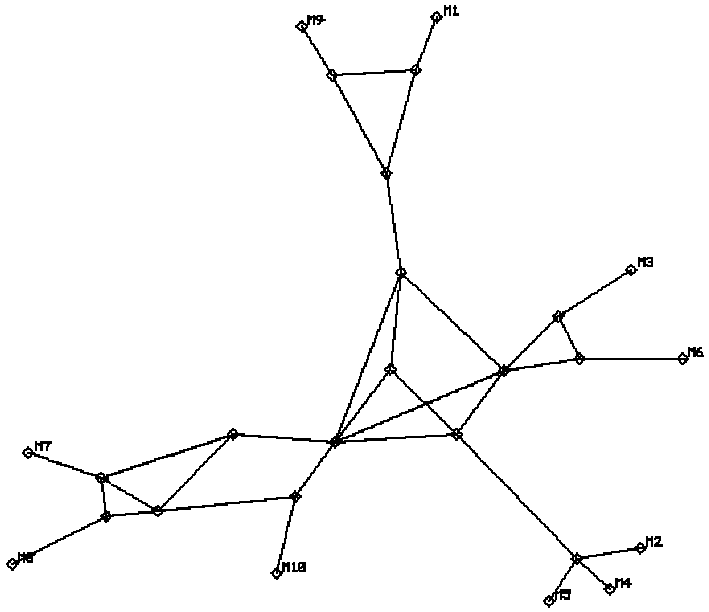


Figure 4-12: Reduced network used for varying the link congestion rates. There are 10 monitors, 16 interior nodes, and 64 links.

There are a few interesting trends to note in Figure 4-11. First, when all links are congested at the same rate, a 1% congestion rate produced the shortest convergence time. The convergence time of the correlation algorithm increases exponentially as link congestion rates are increased above 1% but increases linearly as link congestion rates are decreased below 1%[*]. As the congestion rate increases above 1%, the probability that two or more links congest simultaneously increases relative the probability that only a single link congests. Therefore, a longer simulation length is required to sort out all of the events. Similarly, as the link congestion rate decreases below 1%, it becomes more and more likely that no links in the network congest. Thus, a longer simulation length is required to observe congestion in the network.

A range for the congestion rate means that each link in the network selects its own congestion rate uniformly from this range. Note that the typical value of η decreases. This is due to the fact that the true link congestion rate distribution is spread out and

[*] We demonstrated this relationship with a plot (not shown) similar to the one shown in Figure 4-9.

therefore has a high standard deviation. As the congestion rate range is increased, the simulation length increases exponentially.

4.4.4 Largest Solved Networks

The largest network we solved when all links congested at a 10% rate is shown in Figure 4-13. This reduced network has 20 monitors, 29 interior nodes, and 100 links. The network discovered by the correlation and matroid algorithms is shown in Figure 4-14. The two networks are identical, except for three split nodes (from the matroid algorithm) in the interior of the discovered network.

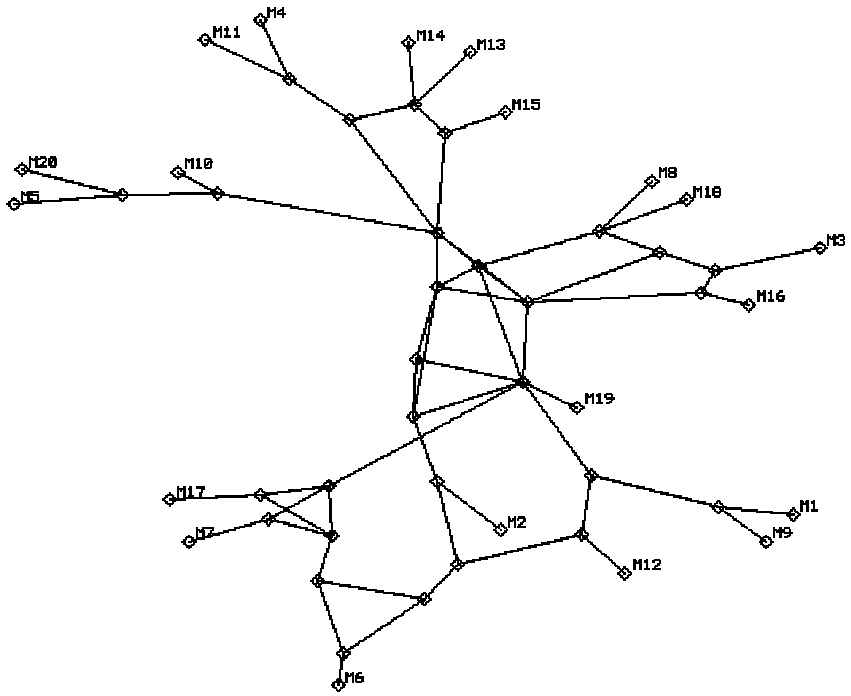


Figure 4-13: Reduced network. There are 20 monitors, 29 interior nodes, and 100 links.

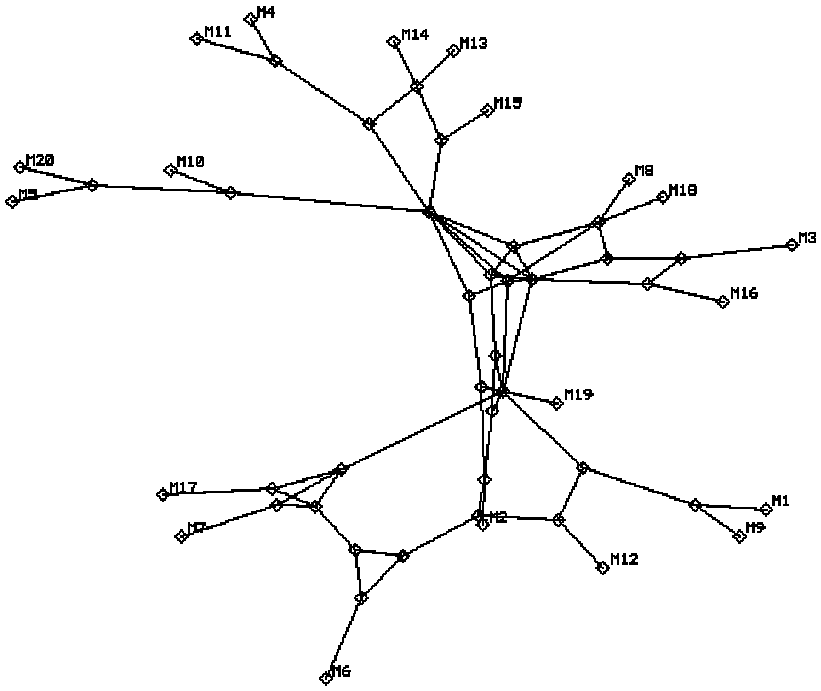


Figure 4-14: Discovered network. Network discovered by the matroid and correlation algorithms. This network is identical to the reduced network except for 3 split nodes. The simulation length, N, was 6.4×10^6 and the correlation algorithm took approximately 10 hours.

We present the largest reduced network we solved for in Figure 4-15, where all links are congested at a 1% rate. The reduced network consists of 50 monitors, 61 interior nodes, and 219 links. The network discovered by the correlation and matroid algorithms is identical to the reduced network, except for five split nodes. The discovered network is shown in Figure 4-16.

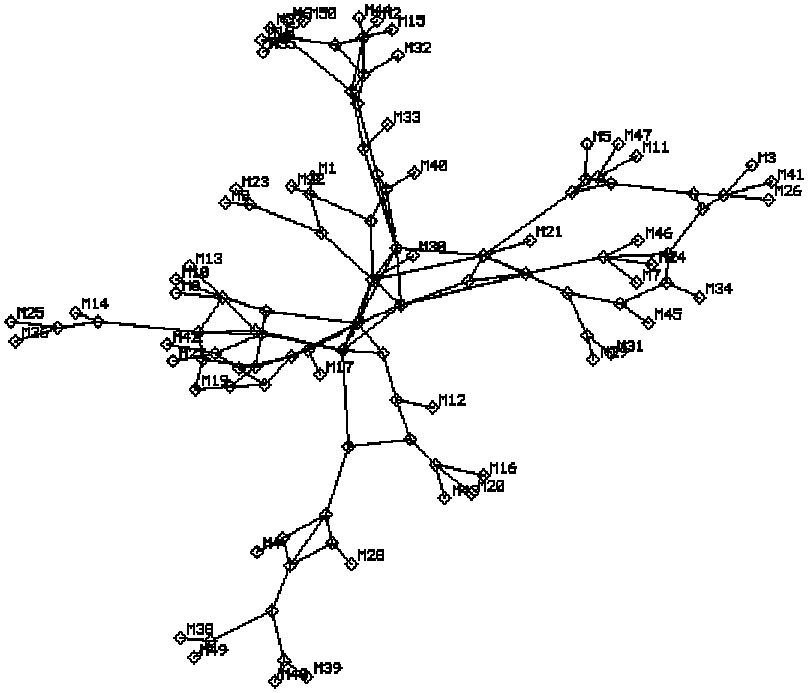


Figure 4-15: Reduced network. There are 50 monitors, 61 interior nodes, and 219 links.

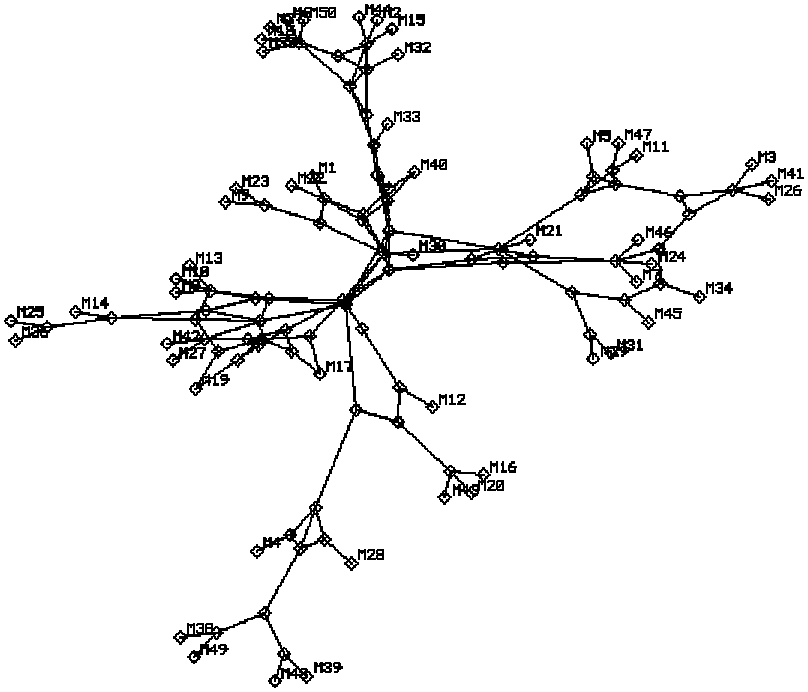


Figure 4-16: Discovered network. Network discovered by the correlation and matroid algorithms. The network is identical to the reduced network except for 5 split nodes. The simulation length, N, was 5×10^5 and the correlation algorithm took approximately 22 hours.

4.4.5 Summary

The overlapping congestion in discrete time model isolates the issue of simultaneously congesting queues. We developed a probability framework to reason about the model and developed techniques to solve for certain quantities in an efficient manner. We found that the algorithm always found the correct path-link matrix as long as the simulation length was increased sufficiently. In the next chapter, we return to a network model that is similar to that of chapter 3. Measurement uncertainty is added back to the model, and we allow queues to congest simultaneously.

Chapter 5

Overlapping Congestion

In this chapter, we consider the third and final network model, overlapping congestion. This network model combines aspects of the first two network models and is the most complex. In section 5.1, we present the details of this network model. Next, we present the correlation algorithm that we use to solve the model in section 5.2. We provide results and discussion in section 5.3.

5.1 Network Model

The network model is based upon the non-overlapping model. However, in chapter 3, only one queue receives packets at any time. Here, we allow any number of queues to be simultaneously congested with packets as in chapter 4. Specifically, each queue alternates periods of *activity*, when it receives packets, with periods of *idleness*, when it is not receiving packets. The duration of an active period is uniformly distributed between 0 and the *congestion_length*, while the duration of an idle period is uniformly distributed between 0 and the *inter_congestion_length*. The duration of both the idle and active periods are chosen independently. As in the non-overlapping network model, the congestion packets are removed from the network after they have been served. The rate that queues are injected with packets during an active period is again chosen from a heavy-tailed distribution as in chapter 3. All of the data collection and network aspects are the same here as they were in the non-overlapping network model.

As in chapter 3, a threshold is applied to the end-to-end delay time series to create a continuous-time binary function for each path. A sample of some binary functions is shown in Figure 5-1.

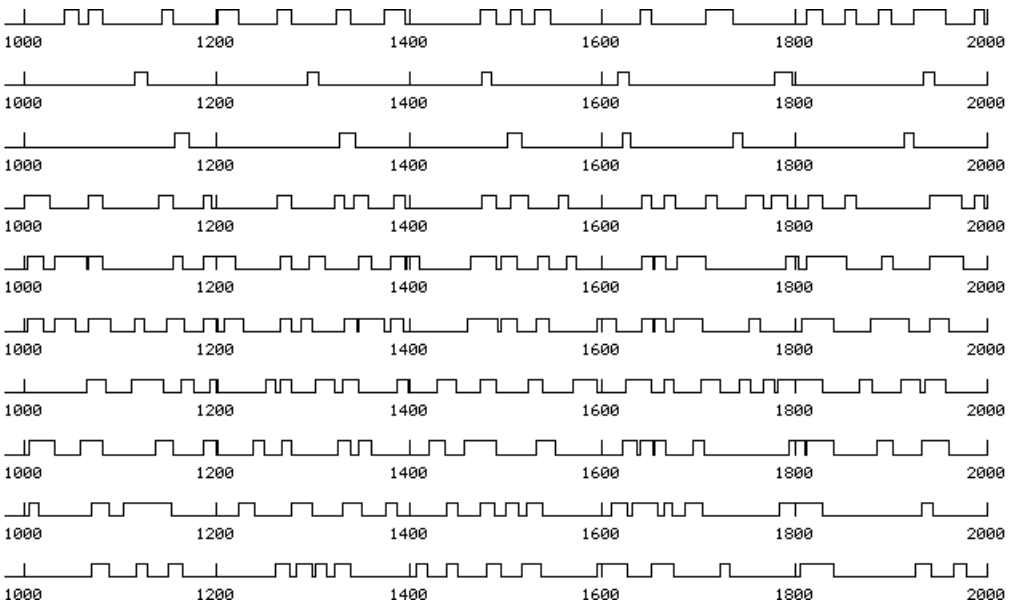


Figure 5-1: Sample of several paths' binary functions. In this simulation the *congestion_length* was 10 seconds and the *inter_congestion_length* was 100 seconds.

5.2 Algorithm

We use the algorithm from the previous chapter with minor modifications to adapt it to continuous time. We also now set the *pruning_threshold* to a low value, instead of between zero and the smallest link congestion rate as was done in chapter 4. Setting the *pruning_threshold* in this way leads to the inclusion of some false leads into the path equations, which we find improves the estimates of the trigger probabilities. Including false leads in this manner means that we carry around *all* of the true leads and *some* of the false leads. Thus, we may be able to separate the true and false leads at some later point. However, if we set the *pruning_threshold* too high, we might exclude some true leads and there would be no obvious way of recovering.

We are not completely sure why setting the *pruning_threshold* to a low value allows the algorithm to succeed. One reason is that if the *pruning_threshold* is too high, then we might prune away true leads. We also reason that in a finite run length, the observation probabilities and the trigger probabilities are not equal to their limiting value. Therefore, although false leads are zero in the limit, in practice they may be better estimated as

being non-zero. Thus, setting false leads to zero may add random noise into the system. By including some false leads in the path equations, the number of nodes in the binary tree formed by the algorithm is no longer easily bounded.

We include false leads in the calculations by setting the *pruning_threshold* just above zero. True links are separated from false links at the bottom level of the binary tree by using a second threshold, the *final_pruning_threshold*. In Figure 5-2(a), we show how the *pruning_threshold* is set at an arbitrary level in the binary tree, and in Figure 5-2(b) we show how the *final_pruning_threshold* is set on the bottom level of the binary tree.

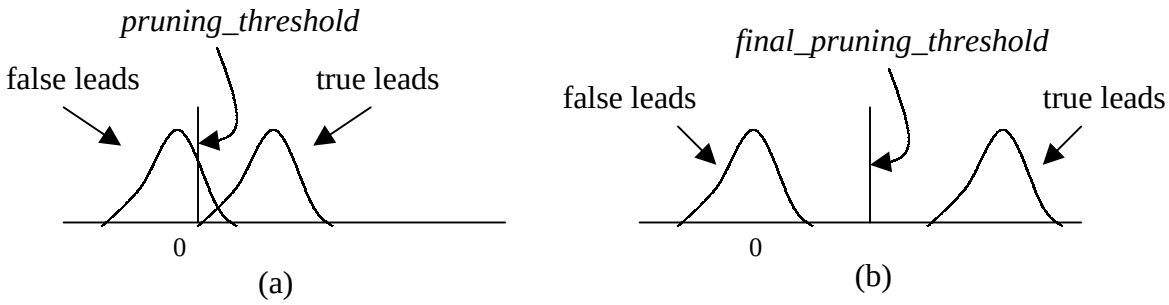


Figure 5-2. Setting of the *pruning_threshold* and the *final_pruning_threshold*. In Figure 5-2(a) the *pruning_threshold* is set just above zero. In Figure5-2(b), the *final_pruning_threshold* is set between zero and the minimum link congestion rate.

In practice, we set the *pruning_threshold* arbitrarily to 0.001. In these experiments, the minimum link congestion rate was never set below 1%. The *final_pruning_threshold* is set in the same way that the *pruning_threshold* was set in chapter 4. That is, we assume that the minimum link congestion rate is known and set the *pruning_threshold* at 40% of this rate.

5.3 Results and Discussion

Tight asymptotic bounds of the algorithm's time complexity are no longer apparent due to the inclusion of false leads. We plot the running time of the algorithm as a function of the number of links in Figure 5-4. We again use η as a proxy for the quality of a result. Figure 5-4 shows the running time of the algorithm for $\eta = 4$.

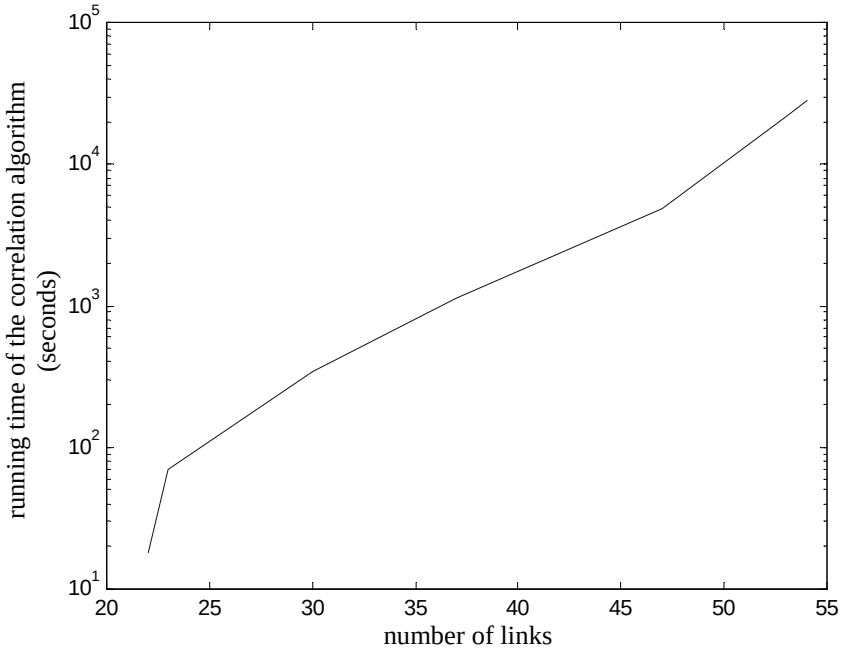


Figure 5-3: Running time of the correlation algorithm, in seconds, as function of the number of links. $\eta = 4$. For all simulations, the *congestion_length* was 10 seconds and the *inter_congestion_length* was 100 seconds.

In this log-linear plot, the running time is roughly a straight line with respect to the number of links. Thus, the length of the simulation time required to solve for the path-link matrix appears to be exponential in the size of the network. Factors that caused the algorithm to take longer to converge included increasing the *congestion_length* relative to the *inter_congestion_length* and increasing the individual link congestion rates.

In Figures 5-4 and 5-5, we plot the histogram of the values of fully specified trigger probabilities for two different simulation lengths. The simulation length used to generate

Figure 5-4 was twice as long as the simulation length used in Figure 5-5. The network used for these plots contained 10 monitors and 54 links. The *congestion_length* was set at 10 seconds and the *inter_congestion_length* was 100 seconds.

In Figure 5-4, the estimated congestion rates on the false links are tightly centered about 0, and a *final_pruning_threshold* set at 0.04 would successfully separate the true links from the false links. However, in Figure 5-5, the congestion rates of the false links and the true links overlap, and therefore the *final_pruning_threshold* can not be set to separate the two distributions.

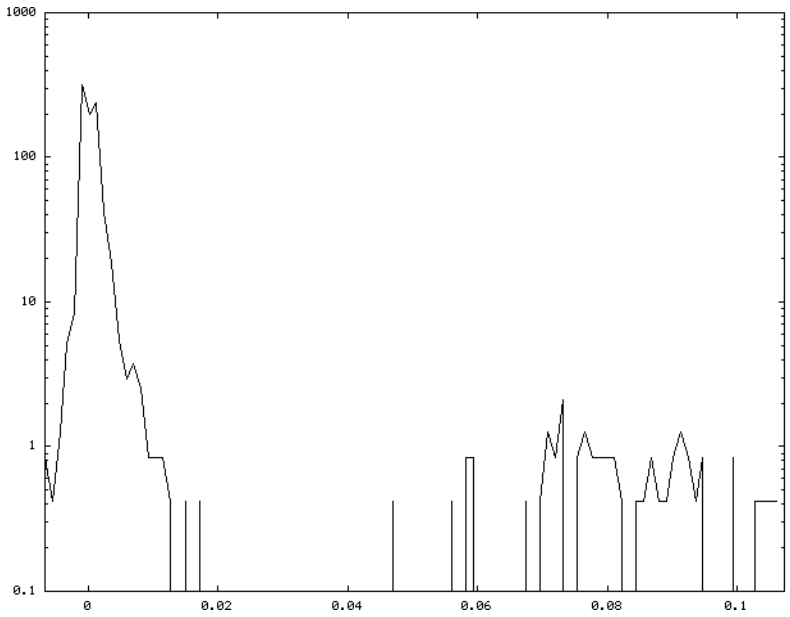


Figure 5-4: Histogram of separated fully specified trigger probabilities. The highest false trigger probability is less than 0.02, while the lowest true trigger probability is greater than 0.04.

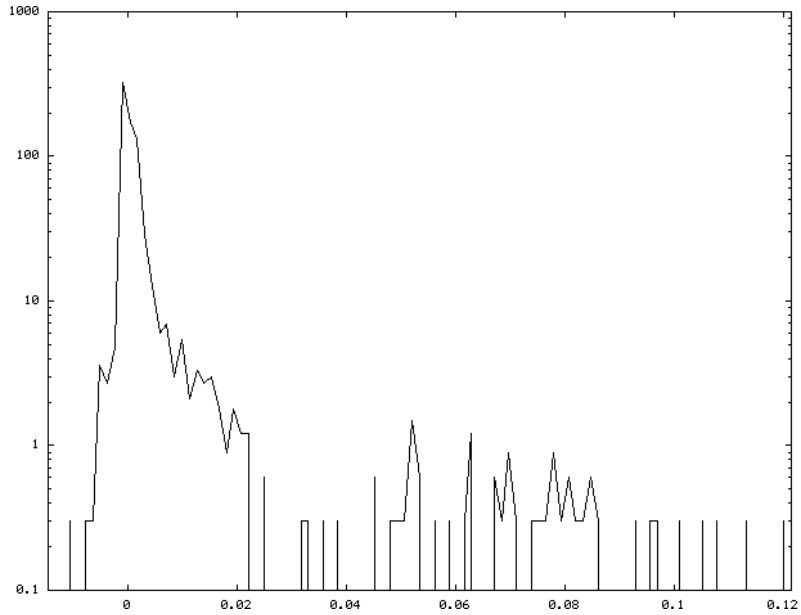


Figure 5-5: Histogram of overlapping fully specified trigger probabilities. Here, the true and false trigger probabilities overlap and therefore they can not be separated with a threshold.

The algorithm successfully solves for the path-link matrix, despite the two aspects that make this network model difficult to solve. The first aspect, simultaneously congesting queues, is addressed using the probability model from chapter 4. The second aspect is that paths that share a congesting queue may not see the effect of this congestion simultaneously. Although the algorithm is not specifically designed for this aspect, we conjecture that this aspect only causes a small amount of noise. Thus, we find that the probability model from chapter 4 is successful in solving for the path-link matrix, if we include a number of false leads.

However, we do not know if the algorithm is successful over the full range of possible parameter settings. For example, it is not clear that the algorithm will always be successful as the *congestion_length* is increased relative to the *inter_congestion_length*. We also do not fully understand how the inclusion of false leads allows the algorithm to solve for the correct path-link matrix.

Chapter 6

Conclusions

Based upon the idea of correlating paths' end-to-end delay times, we have developed topology discovery algorithms for three increasingly complex network models. Path end-to-end delay measurements on a *reduced network* serve as input to the correlation algorithm. The correlation algorithm solves for the path-link matrix, which is used by the matroid algorithm to reconstruct the topology of the network.

In the first network model, *non-overlapping congestion*, only one queue is congested at a time by having packets injected into it for a fixed length of time. These packets are removed from the network after they have been serviced. The correlation algorithm first applies a threshold to the paths' end-to-end delay time series, creating binary functions. A pair-wise comparison test that indicates whether or not two paths share a common congestion cause is used to construct groups of paths with a common congestion cause. Using a systematic construction, these groups correspond to links upon termination. The algorithm performed well. However, as the time between successive queue congestion approached zero, the correct path-link matrix was not always obtained. (Subsequent correlation algorithms were able to solve for the correct path-link matrix in this model even when the time between successive queue congestion was zero. Thus, we view the first correlation algorithm as a stepping stone for the subsequent correlation algorithms.)

We identified two causes of incorrect answers for the first correlation algorithm that motivate the subsequent algorithms. First, congestion in two paths' delay time series that is caused by the same congesting queue is not necessarily temporally identical. Second, congestion in a path's time series occurs when a queue on that path is heavily occupied. Thus, even though only one queue is injected with packets at any given time, two queues

might simultaneously be the source of path congestion. The next network model that we develop isolates this second cause.

Overlapping congestion in discrete time is the second network model. Queues congest independently and thus multiple queues can congest simultaneously. If a queue is congested at time *n*, then all paths that contain that queue are also congested at time *n*. Thus, a path is congested at time *n* if and only if at least one link on that path is congested at time *n.*

We developed a probability model to solve the second network model. We introduced a *trigger* probability that indicates whether a certain set of paths share a common cause of congestion. We solve for the trigger probabilities in an efficient manner and construct the path-link matrix based upon the significant *trigger* probabilities.

We developed η, which measures the statistical distance between the true and false fully specified trigger probabilities, to indicate the quality of a particular result. We found that our ability to find larger and larger networks was limited only by time. There was no numerical instability. However, over many randomly generated network topologies, the time complexity of the algorithm was exponential in the size of the network.

Finally, the third model, *overlapping congestion,* combines aspects of the first two network models. Similar to the first network model, this model is set in continuous time. However, queues congest independently and for random periods of time. Thus, queues can be simultaneously injected with packets as in the second network model, and congestion measurements are not necessarily temporally identical as in the first network model.

Adapting the second algorithm to continuous time and adjusting the setting of some parameters solved the third network model. As in the first algorithm, we initially threshold the paths' end-to-end delay time series to create binary functions. If queues were congested for a short amount of time relative to not being congested, then the correct path-link matrix was obtained. However, a full investigation of the conditions necessary to obtain the correct path-link matrix must be left for future study.

Making some aspects of the network models more realistic motivates much of the future work that we now discuss. An unrealistic assumption that the correlation algorithms use is that the minimum link congestion rate is known. We propose a possible fix for this problem based upon clustering the trigger probabilities into two groups such that the relative distance between the two groups is maximized in terms of the means and the standard deviations of each group.

Another assumption that all of the network models are based upon is that queue congestion occurs independently. In reality, this model is not necessarily realistic since congestion is caused by traffic flows that traverse multiple links. Thus, we would be interested if these (or other) correlation algorithms can cope with this issue.

Modifying the way in which network topologies are generated would increase the soundness of our testing methods. In our study, we have removed *series* and *equivalent* links before using the correlation algorithm. However, a more robust testing method would leave the series and equivalent links in the network. The correlation and matroid algorithms should then output the corresponding reduced network.

An important area of future work is moving from real-world data to the binary functions that are used by the correlation algorithms. We threshold the end-to-end delay time series to create a binary function. However, we do not have a systematic way of setting this threshold. Furthermore, it may be necessary to look at loss rates in order to produce these binary functions. For example, in RED (or another non-FIFO) queuing scheme, loss may be a better indication of congestion than delay. Although we have focused on end-to-end delay measurements, the probability analysis from chapter 4 can be formulated in terms of a loss model, where a trigger probability refers to the packet loss rate on a link rather than its congestion rate. Thus, some hybrid method that combines delay and loss data might prove necessary.

Appendix

Here, we show that the expressions for the trigger probabilities derived by Ratnasamy and McCanne [17] are equivalent to the trigger probabilities derived from the 2-path equations. Here are Ratansamy and McCanne's original equations and solutions for the trigger probabilities:

$$P^{o}_{pq} = P^{t}_{pq} + (1 - P^{t}_{pq})P^{t}_{p\bar{q}}P^{t}_{\bar{p}q} \tag{A.1a}$$

$$P^{o}_{p\bar{q}} = (1 - P^{t}_{pq})P^{t}_{p\bar{q}}(1 - P^{t}_{\bar{p}q}) \tag{A.2a}$$

$$P^{o}_{\bar{p}q} = (1 - P^{t}_{pq})(1 - P_{p\bar{q}})P^{t}_{\bar{p}q} \tag{A.3a}$$

$$P^{t}_{p\bar{q}} = \frac{P^{o}_{p\bar{q}}}{1 - (P^{o}_{\bar{p}q} + P^{o}_{pq})} \tag{A.4a}$$

$$P^{t}_{\bar{p}q} = \frac{P^{o}_{\bar{p}q}}{1 - (P^{o}_{p\bar{q}} + P^{o}_{pq})} \tag{A.5a}$$

$$P^{t}_{pq} = \frac{P^{o}_{pq}P^{o}_{\bar{p}q} + P^{o}_{\bar{p}q}P^{o}_{p\bar{q}} + P^{o}_{p\bar{q}}P^{o}_{pq} + (P^{o}_{pq})^2 - P^{o}_{pq}}{P^{o}_{pq} + P^{o}_{\bar{p}q} + P^{o}_{p\bar{q}} - 1} \tag{A.6a}$$

The 2-path equations:

$$P^{o}_{\bar{p}\bar{q}} = (1 - P^{t}_{pq})(1 - P^{t}_{p\bar{q}})(1 - P^{t}_{\bar{p}q}) \tag{A.1b}$$

$$P^{o}_{\bar{p}} = (1 - P^{t}_{pq})(1 - P^{t}_{p\bar{q}}) \tag{A.2b}$$

$$P^{o}_{\bar{q}} = (1 - P^{t}_{pq})(1 - P^{t}_{\bar{p}q}) \tag{A.3b}$$

Solving the 2-path equations using the technique from chapter 4:

$$P^{t}_{p\bar{q}} = 1 - \frac{P^{o}_{\bar{p}q}}{P^{o}_{\bar{q}}} \qquad \text{(A.4b)}$$

$$P^{t}_{\bar{p}q} = 1 - \frac{P^{o}_{\bar{p}q}}{P^{o}_{\bar{p}}} \qquad \text{(A.5b)}$$

$$P^{t}_{pq} = 1 - \frac{P^{o}_{\bar{p}} P^{o}_{\bar{q}}}{P^{o}_{\bar{p}q}} \qquad \text{(A.6b)}$$

Now, we show that A.4b, A.5b, and A.6b are equal to A.4a, A.5a, and A.6a respectively:

$$P^{t}_{p\bar{q}} = 1 - \frac{P^{o}_{\bar{p}q}}{P^{o}_{\bar{q}}} \qquad \text{(A.4b)}$$

$$= 1 - \frac{P^{o}_{\bar{p}q}}{P^{o}_{p\bar{q}} + P^{o}_{\bar{p}q}}$$

$$= \frac{P^{o}_{p\bar{q}} + P^{o}_{\bar{p}q} - P^{o}_{\bar{p}q}}{P^{o}_{p\bar{q}} + P^{o}_{\bar{p}q}}$$

$$= \frac{P^{o}_{p\bar{q}}}{P^{o}_{p\bar{q}} + P^{o}_{\bar{p}q}}$$

Since $P^{o}_{pq} + P^{o}_{\bar{p}q} + P^{o}_{p\bar{q}} + P^{o}_{\bar{p}q} = 1$, then:

$$= \frac{P^{o}_{p\bar{q}}}{1 - (P^{o}_{\bar{p}q} + P^{o}_{pq})} \qquad \text{(A.4a)}$$

Similarly,

$$P^{t}_{\bar{p}q} = 1 - \frac{P^{o}_{\bar{p}q}}{P^{o}_{\bar{p}}} \qquad \text{(A.5b)}$$

$$= 1 - \frac{P^{o}_{\bar{p}q}}{P^{o}_{\bar{p}q} + P^{o}_{\bar{p}\bar{q}}}$$

$$= \frac{P^{o}_{\bar{p}q} + P^{o}_{\bar{p}\bar{q}} - P^{o}_{\bar{p}\bar{q}}}{P^{o}_{\bar{p}q} + P^{o}_{\bar{p}\bar{q}}}$$

$$= \frac{P^{o}_{\bar{p}q}}{P^{o}_{\bar{p}q} + P^{o}_{\bar{p}\bar{q}}}$$

$$= \frac{P^{o}_{\bar{p}q}}{1 - (P^{o}_{p\bar{q}} + P^{o}_{pq})} \qquad \text{(A.5a)}$$

Finally,

$$P^{t}_{pq} = 1 - \frac{P^{o}_{\bar{p}} P^{o}_{\bar{q}}}{P^{o}_{\bar{p}\bar{q}}} \qquad \text{(A.6b)}$$

$$= 1 - \frac{(P^{o}_{\bar{p}q} + P^{o}_{\bar{p}\bar{q}})(P^{o}_{p\bar{q}} + P^{o}_{\bar{p}\bar{q}})}{P^{o}_{\bar{p}\bar{q}}}$$

Since $P^{o}_{\bar{p}\bar{q}} = 1 - P^{o}_{pq} - P^{o}_{p\bar{q}} - P^{o}_{\bar{p}q}$, then:

$$= 1 - \frac{(P^{o}_{\bar{p}q} + (1 - P^{o}_{pq} - P^{o}_{p\bar{q}} - P^{o}_{\bar{p}q}))(P^{o}_{p\bar{q}} + (1 - P^{o}_{pq} - P^{o}_{p\bar{q}} - P^{o}_{\bar{p}q}))}{P^{o}_{\bar{p}\bar{q}}}$$

$$= 1 - \frac{(1 - P^{o}_{pq} - P^{o}_{p\bar{q}})(1 - P^{o}_{pq} - P^{o}_{\bar{p}q})}{P^{o}_{\bar{p}\bar{q}}}$$

$$= 1 - \frac{1 - P^{o}_{pq} - P^{o}_{p\bar{q}} - P^{o}_{pq} + (P^{o}_{pq})^2 + P^{o}_{pq}P^{o}_{\bar{p}q} - P^{o}_{p\bar{q}} + P^{o}_{pq}P^{o}_{p\bar{q}} + P^{o}_{\bar{p}q}P^{o}_{p\bar{q}}}{P^{o}_{\bar{p}\bar{q}}}$$

$$= \frac{P^{o}_{\bar{p}\bar{q}} - 1 + P^{o}_{pq} + P^{o}_{p\bar{q}} + P^{o}_{pq} - (P^{o}_{pq})^2 - P^{o}_{pq}P^{o}_{\bar{p}q} + P^{o}_{p\bar{q}} - P^{o}_{pq}P^{o}_{p\bar{q}} - P^{o}_{\bar{p}q}P^{o}_{p\bar{q}}}{P^{o}_{\bar{p}\bar{q}}}$$

$$= \frac{(1 - P^{o}_{pq} - P^{o}_{p\bar{q}} - P^{o}_{\bar{p}q}) - 1 + P^{o}_{pq} + P^{o}_{\bar{p}q} + P^{o}_{pq} - (P^{o}_{pq})^2 - P^{o}_{pq}P^{o}_{\bar{p}q} + P^{o}_{p\bar{q}} - P^{o}_{pq}P^{o}_{p\bar{q}} - P^{o}_{\bar{p}q}P^{o}_{p\bar{q}}}{P^{o}_{\bar{p}q}}$$

$$= \frac{P^{o}_{pq} - (P^{o}_{pq})^2 - P^{o}_{pq}P^{o}_{\bar{p}q} - P^{o}_{pq}P^{o}_{p\bar{q}} - P^{o}_{\bar{p}q}P^{o}_{p\bar{q}}}{P^{o}_{\bar{p}q}}$$

$$= \frac{P^{o}_{pq} - (P^{o}_{pq})^2 - P^{o}_{pq}P^{o}_{\bar{p}q} - P^{o}_{pq}P^{o}_{p\bar{q}} - P^{o}_{\bar{p}q}P^{o}_{p\bar{q}}}{1 - P^{o}_{pq} - P^{o}_{p\bar{q}} - P^{o}_{\bar{p}q}}$$

$$= \frac{P^{o}_{pq}P^{o}_{\bar{p}q} + P^{o}_{\bar{p}q}P^{o}_{p\bar{q}} + P^{o}_{p\bar{q}}P^{o}_{pq} + (P^{o}_{pq})^2 - P^{o}_{pq}}{P^{o}_{pq} + P^{o}_{\bar{p}q} + P^{o}_{p\bar{q}} - 1} \qquad \text{(A.6a)}$$

References

[1] J-C. Bolot. “End-to-end packet delay and loss behavior in the internet,” *ACM SIGCOMM ’93*, pp. 289-298, Sept. 1993.

[2] R. Caceres, N. G. Duffield, J. Horowitz, F. Lo Presti, and D. Towsley. “Loss-based inference of multicast network topology,” *Proc. of 1999 IEEE Conference on Decision and Control*, Phoenix, AZ, Dec. 1999.

[3] R. Caceres, N. G. Duffield, J. Horowitz, D. Towsley, and T. Bu. “Multicast-based inference of network-internal characteristics: accuracy of packet-loss estimation,” *IEEE INFOCOM ’99*, New York, Mar. 1999.

[4] K. Calvert, M. Doar, and E. Zegura. “Modeling internet topology,” *IEEE Communications Magazine*, June 1997.

[5] M. Coates, and R. Nowak. “Unicast network tomography using EM algorithms,” Technical Report TR-0004, Rice University, ECE Dept., September 2000.

[6] Felix: Independent Monitoring for Network Survivability. http://govt.argreenhouse.com/felix

[7] M. W. Garrett, *et al.* “Felix project technical documentation: final report,” Darpa contract F30602-97-C-0188, Telcordia Technologies, October 2000.

[8] M. W. Garrett, W. Willinger. “Analysis, modeling and generation of self-similar VBR video traffic,” *ACM SIGCOMM ’94*, Sept. 1994.

[9] K. Harfoush, A. Bestavros, and J. Byers. “Unicast-based characterization of network loss topologies,” Technical Report TR-2000-013, Boston University, CS Dept., May, 2000.

[10] Internet Monitoring & PingER at Stanford Linear Accelerator Center. http://ww.slac.stanford.edu/comp/net/wan-mon/tutorial.html#metrics

[11] IPMA: Internet Performance Measurement and Analysis. http://www.merit.edu/ipma

[12] S. Moon, J. Kurose, P. Skelly, and D. Towsley. “Correlation of packet delay and loss in the internet,” Technical Report TR-98-11, University of Massachusetts at Amherst, 1999.

[13] NIMI: National Internet Measurement Infrastructure. http://www.psc.edu/networking/nimi

[14] V. Paxson. “End-to-end internet packet dynamics,” *Proc. ACM SIGCOMM ‘97*, Cannes, France, pp. 139-152, Sept. 1997.

[15] V. Paxson. “End-to-end routing behavior in the internet,” *Proc. ACM SIGCOMM ’96*, Stanford, Aug. 1996.

[16] F. Lo Presti, N. G. Duffield, J. Horowitz, and D. Towsley. “Multicast-based inference of network-internal delay distributions,” Preprint AT&T Laboratories and University of Massachusetts.

[17] S. Ratnasamy and S. McCanne. “Inference of multicast routing tress and bottleneck bandwidths using end-to-end measurements,” *Proc. IEEE INFOCOM ’99*, pp. 353-60, New York, March 1999.

[18] D. Rubenstein, J. Kurose, and D. Towsley. “Detecting shared congestion of flows via end-to-end measurement,” *ACM SIGMETRICS ’00*, Santa Clara, CA, June 2000.

[19] Skitter. http://www.caida.org/tools/measurement/skitter

[20] Surveryor. http://www.advanced.org/surveyor

[21] A. Tanenbaum. *Computer Networks*. Prentice Hall, Upper Saddle River, NJ, pp. 347-348, 1996.

[22] J. M. Wooldridge. *Introductory Econometrics*. South-Western College Publishing, pp. 78-81, 2000.

[23] M. Yajnik, J. Kurose, and D. Towsley. “Packet loss correlation in the mbone multicast network,” *Proc. IEEE Global Internet*, Nov. 1996.

[24] M. Yajnik, S. Moon, J. Kurose, and D. Towsley. “Measurement and modeling of the temporal dependence in packet loss,” *Proc. IEEE INFOCOM ’*99, pp. 345-52, March 1999.