Measuring an IP Network in situ

Hal Burch

DRAFT: February 7, 2005

CMU-CS-??-???

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Abstract

The Internet, and IP networking in general, have become vital to the scientific community and the global economy. This growth has increased the importance of measuring and monitoring the Internet to ensure that it runs smoothly and to aid the design of future protocols and networks. To simplify network growth, IP networking is designed to be decentralized. This means that each router and each network needs and has only limited information about the Internet. One disadvantage of this design is that measurement systems are required in order to determine the behavior of the Internet as a whole.

This thesis explores ways to measure five different aspects of the Internet. The first aspect considered is the Internet's topology, the inter-connectivity of the Internet. This is one of the basic questions about the Internet: what hosts are on the Internet and how are they connected? The second aspect is routing: what are the routing decisions made by routers for a particular destination? The third aspect is locating the source of a denial-of-service (DoS) attack. DoS attacks are problematic to locate because their source is not listed in the packets. Thus, special techniques are required. The fourth aspect is link delays. This includes both a general system to determine link delays from end-to-end measurements and a specific system to perform end-to-end measurements from a single measurement host. The fifth aspect is the behavior of filtering on the network. Starting about fifteen years ago, to increase security, corporations started placing filtering devices, i.e., "firewalls", between their corporate network and the rest of the Internet. For each aspect, a measurement system is described and analyzed, and results from the Internet are presented.

Acknowledgements

My friends, family, and colleagues have assisted in many ways. I am unlikely to be able to properly enumerate the contribution of each. I will thus limit myself to these few:

My advisors, Bruce Maggs and Gary L. Miller, have helpfully guided and directed me through-out my tenure at Carnegie Mellon.

My colloborators, Chris Chase, Dawn Song, Bill Cheswick, Steven Branigan, and Frank Wojcik, contributed greatly to much work here-in.

My wife, Kimberly Jordan Burch, supported me, both emotionally and linguistically, fixing my English errors.

Phil Gibbons, Linda Moya, and Tim and Jen Coles opened their homes to me to allow me to continue to visit Carnegie Mellon on a regular basis while living with my wife in New Jersey. AT&T provided me with a workspace while in New Jersey.

Contents

Chapter 1

Introduction

The Internet has grown from a small research network to a global network used by businesses, individuals, and researchers. In December 1969, the Internet (then known as ARPANET) consisted of only four nodes: UCLA, UCSB, Stanford Research Institute, and the University of Utah. From these humble beginnings, the Internet has grown to hundreds of millions of hosts on tens of thousands of networks. This growth in size has been coupled with a growth in the importance of the Internet. According to Cisco[20], $830 billion of revenue was generated over the Internet in 2000. That exceeds the GDP of all but thirteen countries. With the introduction of broadband to households, services that traditionally ran on other networks, such as voice and video, are increasingly moving to the Internet. For voice, this migration has already begun, with major cable companies offering voice over IP (VoIP) service to consumers. This increasing dependence on the Internet for communication and commerce means that the Internet is becoming vital to national and world economies.

As the Internet grows in importance, the ability to monitor and measure it effectively also grows. The same growth that increases the importance of the Internet however, also makes the Internet more difficult to monitor. When the Internet began as the ARPANET, it had only one service provider. Today, the Internet has thousands to tens of thousands of service providers. To handle this growth, the Internet is built as a network of networks designed to make it easy for a new network to join with minimal configuration changes. To facilitate this, each router operates while maintaining only limited information. This decentralized design means that the Internet has been able to grow without over-burdening the routers with both static and dynamic configuration information. This localization of information makes it more difficult to understand the global behavior of the Internet, as each router has only limited data. Understanding the behavior of the network requires building measurement systems to collect the necessary information.

Measurement systems do much more than merely satisfy intellectual curiosity about the Internet. Measurement systems are necessary for operation, architecture, protocol design, and protocol implementation. Internet service providers and companies with corporate IP networks must measure their network to ensure that it is operating properly, and will continue to operate properly. They must make certain that all nodes in the network can communicate properly, that sufficient bandwidth exists to meet the needs of the network, and that the network is sufficiently secure. Network architects want to know what is happening on the current network in order to design the most efficient network they can and to ensure the network meets the demands placed on it.

Protocol designers often face the daunting task of designing a protocol that will work when run by hundreds of millions of hosts. They must do this while being able to test it on only hundreds to thousands of hosts. For this reason, protocol designers often turn to analysis and simulation in order to stress their protocols with large-deployment scenarios. It is important to understand the Internet, as neither analysis nor simulation is useful if the assumptions on which they are based are flawed. Protocol implementors face similar problems, but must additionally deal with field faults that result from implementation error. Good simulation tools can simplify the problem of determining the cause of a field fault and finding a remedy for it. Moreover, having a simulation that closely reflects reality makes it more likely that implementors can find and repair bugs before they are deployed.

Building measurement systems often involves finding ways to extract information from available behaviors. To date, designers of Internet protocols have paid only limited attention to including ways to measure the systems they are building. Instead, protocol designers generally focus on the problem of building a networking system that scales to hundreds of millions of hosts while maintaining high reliability, low latency, and a high level of robustness. This focus is proper, as a precisely measurable system is of little use if it does not work. Even when a protocol includes monitoring features, it is difficult for protocol designers to predict everything that a user may want to monitor.

Although one might think that monitoring features could simply by added to the protocol, reality is not that simple. Changing the protocol requires each vendor to change their implementation of the protocol and then each system administrator to update their hosts to run the new protocol. A protocol, once defined by standards or practice, can have dozens of implementations and be deployed on thousands or millions of hosts. Convincing all vendors and administrators is difficult at best for widely-deployed protocols. Even if they can be convinced, a change in a protocol can take years or longer to become widely deployed. The most obvious example of this is IPv6[27], which was first proposed in 1995, but continues to have limited deployment. Any realistic method of measuring the Internet cannot require a change to the protocols running on more than a few end systems. This limitation often means that measurement methods often find a new way to extract useful information from an existing feature of a protocol.

An example of using existing features in an unusual way is traceroute[46]. Traceroute is a vital measurement tool that has long been used by network administrators to identify the path taken by outgoing packets towards a given destination. Traceroute works by sending packets with low time-to-live (TTL) values. After the packet has traveled through a number of routers equal to the initial TTL value, the next router will discard the packet. When discarding such packets, most routers will return a ICMP error packet[73], which includes an IP address of the router sending the packet. By varying the TTL value, traceroute determines the routers along the path.

Traceroute is commonly employed to determine the cause of network difficulties. If packets are not delivered, the last responsive hop in the path indicates the source of the problem. By examining the round-trip-times of the traceroute probes, sources of high latency can often be identified. In addition, traceroute readily identifies routing loops, which occur when a set of routers become confused about the path to a particular destination. The resulting path, like the snake Uroboros, swallows its own tail and becomes a loop. Routing loops are one of main reasons the TTL field exists. Without the TTL field, the packet would cycle endlessly in the loop, stopping only if a transmission error occurred or the path corrected itself.

This dissertation presents ways to measure the Internet within the constraints of the current infrastructure. It begins with Chapters 2 and Chapter 3, which address measuring the topology. Chapter 2 addresses the problem of collecting and presenting IP-level topology information on the Internet or any IP-based network. Chapter 3 considers the problem of determining if two IP addresses represent the same host, which can be used to refine topologies by combining multiple IP addresses belonging to the same host.

Chapters 4 and 5 utilize topology information for other measurements. Chapter 4 presents an algorithm to estimate the routing towards a particular destination. The algorithm results in routing estimations for a much larger fraction of the Internet than previously-known techniques. The algorithm is analyzed for its coverage and accuracy and compared against those techniques. Chapter 5 describes a (rather evil) method to determine the source of a stream of spoofed packets, common in denial-of-service attacks. It also includes several other schemes to determine the source of such streams or to disallow the production of such streams in the first place. It includes a brief overview of research subsequent to its publication.

The remaining chapters focus on other measurement problems. Chapter 6 gives a method for determining individual link delays from end-to-end measurements. This is commonly of interest to ISPs and other network operators, as it tells them how well their network is performing. Chapter 7 examines common firewall behavior, as observed from the Internet. This provides insight into the concerns and attitudes of firewall administrators. Such information is also useful to protocol designers, which much be sensitive to firewall behaviors. Chapter 8 ends the dissertation with conclusions.

Chapter 2

Network Mapping[1]

One of the basic measures of a network is its connectivity. Having no knowledge of the hosts and links in the network makes information about the behavior of hosts or links nearly useless. It is unclear what the proper questions are to ask about the hosts or links if their relationships are not known. Any available information is difficult to interpret or use without knowing the structure. This chapter describes the Internet Mapping Project, an attempt to measure and record the topology of the Internet and how it changes over years.

The basic technology used by the Internet Mapping Project (IMP) is traceroute[46]. Traceroute gives the set of hops from one host to another. Modulo a few caveats discussed later, adjacent hops in the resulting path are connected by a link. The topology can, at least in theory, be determined by taking a large set of traceroute paths and taking the union. By taking a list of all announced network on a network, such as the Internet, and discovering the path to each of these networks, a good picture of the "center" of the network can be built. This provides a kind of picture of what the network looks like as a whole. Of course, this is an egocentric view, as it captures the paths taken by packets outgoing from the measurement host. Thus, the picture is a reachability graph, not a complete map.

The Internet Mapping Project measures the Internet using this process. Every morning, the mapping program of the IMP measures the topology of the Internet. The Internet data is archived on CD-ROMs. Scans have been run daily since they began in August, 1998. This scanning detects long-term routing and connectivity changes on the Internet. Daily scans are likely to miss a short outage of a major backbone that lasts for only a few hours, unless it happens during a scan. Events that happen on larger time scales, such as a natural disaster, war, or a major act of terrorism, are likely to show up.

Due to the magnitude of the resulting databases, a method of visualizing it is required, allowing the eye to improve the understanding of the collected data. The visualization makes it possible to pick out interesting features for further investigation and find errors in Internet router configurations, such as routers that return invalid IP addresses. It would be desirable to have a large paper map with the properties of traditional flat maps: helpful when navigating towards destinations, display connectivity, readily reveal major features and interesting relationships, and are hard to fold up.

IMP's layout algorithm uses a spring-force algorithm to position the nodes on the map. A few simple rules govern the adjustment of a point's position based on proximity of graph neighbors, number of incident edges, and the number and position of close nodes that are not neighbors. When IMP began, the Internet graph had 88,107 nodes and 99,664 edges. The points were shuffled for 20 hours on a 400MHz Pentium to obtain the maps shown in this chapter. The Internet graph has grown to 153,133 nodes and 183,144 edges

[1]This chapter represents work with William Cheswick and Steven Branigan, first published as "Mapping and Visualizing the Internet" in 2000 USENIX Annual Technical Conference[17]. A later article was also published in IEEE Computer[7]

today. The nodes are still shuffled for 20 hours to layout the graph, but now those 20 hours take place on a 1 GHz Pentium III box.

In the course of developing and testing the mapping software, it become apparent that mapping is a more generally useful pursuit. Large corporate networks are difficult to manage and offer many security problems. Often, local changes in the network may not be properly communicated to the central administrators. For example, if a division of a company needs to communicate with another company, that division may purchase a connection to the other company, and set-up their local network to allow communication. The network, detecting the change, may reconfigure the entire network to be able to communicate with the newly-connected company. Because networks are designed to be robust to network changes, this can occur without action by the central administrators. The net effect of such a change is that anyone in the newly connected company can communicate with any host within the network. This free access is a security problem. Firstly, any virus, worm, or hacker that infiltrates the new company has access to the network. Secondly, often two cooperating companies compete in many areas, making free access extremely undesirable. A map of a company's network topology can yield substantial information and can help spot likely leaks in a company's perimeter security.

2.1 Motivation

The initial motivation for collecting path data came out of a Highlands Forum, a meeting that discussed possible responses to future infrastructure attacks using a scenario from the Rand Corporation. It was clear that a knowledge of the Internet's topology might be useful to law enforcement if the nation's infrastructure was under attack. It might be useful to know how connectivity changes before and during an attack on the Internet infrastructure. The Internet topology could also be useful for tracking anonymous packets back to their source, as will be discussed in Chapter 5. In addition, an openly-available map could be useful to monitor the connectivity of the Internet, and would be helpful to a variety of investigators.

Good ISPs already watch this kind of information in near real-time to monitor the health of their own networks, but they rarely know anything (or care much) about the status of networks that are not directly connected to theirs. No one is responsible for watching the whole Internet. Given its size, the entire Internet would be difficult to watch. There is a major web of interconnecting ISPs that in some sense defines the "middle" of the net—the most important part. The Internet Mapping Project grew out of attempts to map this middle. It uses *traceroute*-style packets, which means that it only maps outgoing paths, and only from the measurement host—these limitations are discussed below. Even this limited connectivity information can yield insights about who is connected to whom.

The database itself can be useful for routing studies and graph theorists looking for real-world data to work with. Since the data is being collected daily over a long period of time, it may be possible to extract interesting trends. Data is systematically collected daily, building a consistent database that can be used to reconstruct routing on the Internet approximately for any day where mapping was done, at least the paths from the measurement host.

The mapping software has lent itself to another pressing problem: controlling an intranet. Software that can handle 150,000 nodes on the Internet can easily handle intranets of similar size. An intranet map can be colored to show insecure areas, business units, connections to remote offices, etc.

IMP's visualizations of the Internet itself have attracted wide media interest[10, 103]. Media generally visually represents the Internet by showing people staring at a web browser. The Internet Mapping Project's maps give some idea of the size and complexity of the Internet. Figure 2.1 shows how the number of nodes and edges has grown over the project's lifetime, giving a rough notion of the growth of the Internet.

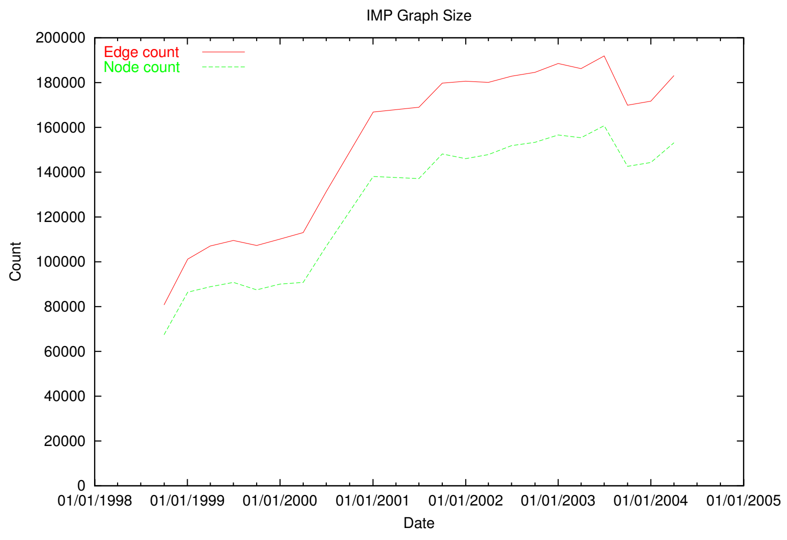


Figure 2.1: Size of the Internet graph, as measured by IMP, in terms of node and edge counts over the last six years.

2.2 Network Mapping

The data from the Internet Mapping Project (IMP) consists of paths from a measurement host towards a single host on a destination network. The list of networks on the Internet is the combination of four lists. The first list comes from the routing arbiter database[60]. This is a central registry of all assigned Internet addresses, including those used only privately. Each provides a target network address as a CIDR block, such as 135.104.0.0/16. Two other sources of networks are BGP route entries and domain name service delegations. BGP route entries are CIDR blocks already. Domain name service delegations come from delegation of reverse lookups (*in-addr.arpa*). A delegation of, say, *2.128.in-addr.arpa* corresponds to the CIDR Block 128.2.0.0/16. Although not originally included, both sources have been added since. The fourth list is every class B (/16) CIDR block. This last list provides insurance against the first three sources missing a large network.

The network scanner must traceroute to a particular host, rather than tracing to a CIDR block. As it is not particularly important that the host actually be present, the network scanner randomly picks an IP number in each CIDR block. This random selection is biased, based on a quick survey of commonly-used IP addresses (e.g., the most common last octet is 1 and lower numbers are more common). If an IP address is found to which the scanner can find a complete traceroute, that IP address is recorded and reused. Essentially, the mapping performs a slow host scan over time until a responsive host is found. Even with this slow host scan, most of the network traces end either with silence (due to an nonexistent address or a firewall) or an ICMP error-reported failure. The large number of incomplete paths is largely a result of aggressively including CIDR blocks. Because testing an nonexistent CIDR block is both quick and non-intrusive, the gathering process discards no potential CIDR blocks listed anywhere.

This technique only records an outgoing packet path. The incoming path is often different: many Internet routes are asymmetric, as ISP interconnect agreements often divert traffic through different connections. Techniques to discover return packet paths are discussed in Chapter 4.

The path may vary between traces, or even individual probes, depending on outages, redundant links, reconfigurations, etc. This means the mapping program may occasionally ‘discover’ paths that don’t exist. Imagine a packet to Germany that is either routed through the United Kingdom or France at random, for example. As alternate packets travel through alternate paths, the mapping program will infer connections between the alternate paths that do not exist. Load-balancing over large stretches of paths is believed to be rare, making the effect of them limited. In terms of outages and routing changes, the number of routes changing during a scan should, in most cases, be relatively small.

The technique employed only discovers the IP path. A single IP-layer link may not represent a physical link. For example, if an ISP is running their backbone over ATM, then each link represents a virtual circuit that may travel through many ATM nodes. Depending on how the ATM network is configured, such an ISP's backbone may appear to be completely connected, even though it isn't physically true. From an IP standpoint, however, detecting the physical connectivity is extremely difficult.

The target, date, path data, and path completion codes are recorded in a simple text format. The database is manipulated with traditional UNIX text tools and some simple additional programs. Each day's database is compressed and stored permanently. Copies are available upon request. The latest Internet database was available online, but was taken down over concerns about the data. The database is routinely supplied to researchers, although they receive a database that is a few months old. The database is about 25 MB.

2.2.1 Mapping, Not Hacking

One goal is that the traceroutes are not confused with hacking probes. This means that the mapping must proceed gingerly. The mapping program probes with ICMP Echo Request packets[73] (ping). Most intrusion detection systems recognize these as ping packets, though the TTL values used differ from traditional pings. At worse, the probes tend to confuse system administrators, as there are few real services that use ping.

Originally, the path was discovered one hop at a time. For each hop, a probe was sent out. If no reply was received in 5 seconds, a second probe was sent. If no reply was received to the second probe in 15 more seconds, a third probe was sent. If no reply was received within 45 seconds after the third probe was sent, the path discovery was halted. Stopping a path trace after failing only one hop stopped the scanning from discovering the second half of many paths[21], but made the scanning less threatening to network citizens.

Today's scanner discovers each path in two passes. The first pass tests each hop sequentially, sending only one packet per hop. The first pass ends either when the target is reached or two hops in a row fail to respond within three seconds. The second pass attempts to fill-in the holes from the first pass left by non-responding hops. Each hole is probed twice more, once with a five second timeout, and once with a 20 second timeout. This scanning methodology is designed to only send one more packet across a blocking firewall than the previous technique, while completing more paths. Allowing one hole completes approximately 4% more of the paths, but will still not scan across an ISP that configures all of their routers to return no responses to traceroute-like probes.

Another aspect of attempting to make it clear that the mapping traffic is not hacking traffic is to enable curious system and network administrators to easily determine what the traffic is. The first clue to them is the name of the measurement host, `netmapper`, and the domain `research.lumeta.com`. This name itself tells most of the story, and it is believed that this makes most administrators who do notice the packets nod and move on to other work. A web page describing the project is maintained[56]. A DNS TXT record has been added to `netmapper`'s entry that points to IMP's web page. In addition, all web queries to `netmapper` are redirected using the world's shortest (and safest) web server (the web server just `cat`'s a file).

A few network administrators have complained. They either did not like the probe, or IMP's packets cluttered their logs (The Australian Parliament was the first on the list!). These networks are recorded in an opt-out list and cease probing them. Certainly others may have simply blocked IMP's packets, or filtered

IMP's probes out of their logs. It might be interesting to compare hosts that were reached early in the scans and later fell out of sight.

A number of emergency response groups made contact asking about the network scanner's activity. The goal was for them to understand the mapping activity, thus satisfying their justifiable curiosity. It would be much harder time justifying the probes if it included overt host or port scans, which often precede a hacking attack. Only a tiny percentage of the Internet system administrators appear to have noticed the mapping efforts.

The measurement host itself is highly resistant to network invasion: some other network scans have promoted powerful hacking responses. Of course, like any other publicly-accessible host, it could fall to denial-of-service attacks.

2.3 Map Layout

A force-directed method similar to previous work[24, 35] is used to lay out the graph. The basic idea is to model the graph as a physical system and then to find the set of node positions that minimizes the total energy. The standard model employed is spring attraction and electrical repulsion. Attraction is done by connecting any two connected nodes by a spring. The repulsive force derives by giving each node a positive electrical charge, causing them to repel each other.

Finding a minimum of such physical models has been well studied. In particular, the most common techniques are gradient descent[84], conjugate gradient[88], and simulated annealing[84]. None of these algorithms are guaranteed to find a global minimum in a reasonable time, but they are able to find approximately minimum configurations. Because it is easier to code, gradient descent was selected.

At the time of the development of IMP's graph layout algorithm, graph drawing work considered graphs with a tenth the size of the Internet dataset as huge[39, 98]. Extending the runtime results of previous work to IMP's graph and adjusting for a faster host yields times on the order of months to millennia. Thus, the standard algorithms at the time were too slow for IMP's graph. Two tricks are used to more quickly compute a layout, at the cost of possibly being less optimal.

The first trick is replace the electrical repulsive field with spring repulsion. Imagine that any two nodes which do not share an edge are connected, via infinite strings, to a spring. Thus, if the nodes are further apart than the rest length of the spring, there is no force applied. If the nodes are are closer than the spring's rest length, the spring is compressed, and the nodes are pushed apart. This bounds the repulsive force, which means that instead of having to calculate a quadratic number of repulsive forces, the number of repulsive forces goes down to approximately linear, since pairs of points that are further apart than the rest length can be ignored. Thus, the exact repulsive force on each node can be calculated in approximately linear time.

The real optimization, however, is laying out the graph one layer at a time. First the links to IMP's three ISPs are laid out and the system is iterated until they stop moving "much." Then, all the routers one hop further away are added, and the system is iterated (which may move the nodes from previous levels as well). Then the next hop, continuing until all the routers are included. This tends to give placement based on information high in the tree. A layout movie of the Internet is available online[16].

IMP's original layouts showed all the paths. This resulted in a picture such as Figure 2.2[2]. While the middle is mostly a muddle, the edges showed intriguing details. Note that a 36x40 inch plot is much more useful—a dense graph is easier to view on a larger printout. Dave Presotto described this smaller version as a smashed peacock on a windshield.

[2]Due to printing limitations, all figures are rendered in black and white. To view color versions, consult the PostScript version of the thesis.

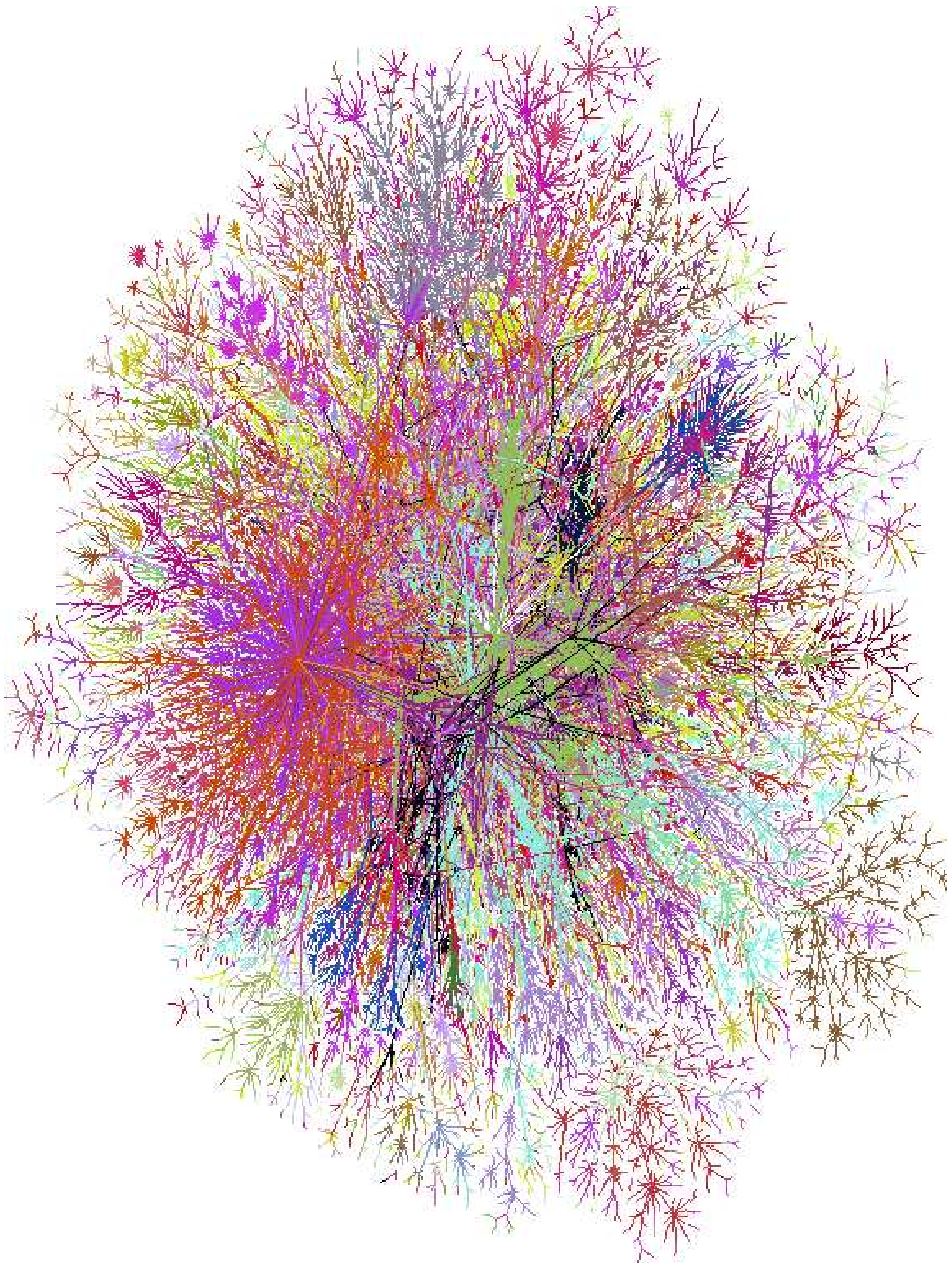


Figure 2.2: “Peacock-on-the-windshield” map from data taken in September 1998. The circled network is `cw.net`.

The map is colored using IP address; the first three octets of the IP number are used as the red, green, and blue color values respectively. This simplistic coloring actually shows communities and ISPs quite well.

Some features already appear on this map: The fans at the edges show some interesting communities: Finland, AOL, some DISA.MIL, and Telstra (Australia and New Zealand). Looking at the color version of the map reveals a middle that is muddled, showing IMP's ISPs at the time: UUnet (green) and BBN (deep blue). SprintNet (sky blue) peeks through the sides.

The eye is drawn to a rather large star at the top of the picture, which represents the Cable and Wireless (cw.net) backbone, formerly the MCI backbone, formerly NSFnet. It is a major feature (if not the major feature) on the map. There are two reasons for this: (i) they are a huge backbone provider, and (ii) their backbone is an ATM network, connecting well over a hundred nodes around the world. Since IMP's scanning is run at the IP level (level 3), this large network collapses to a single point. The smaller "Koosh" balls may be other ATM networks—this has not been investigated.

This map has changed over time, as changes in routing and ISP configurations are observed. As the Internet has changed, the predominant colors have changed as well.

The Internet Mapping Project started collecting and preserving DNS names for the routers in March 1999. The collection of canonical names provides a rich source of data to color the graph drawing. For example, colors can be selected based on top-level domain, showing the approximate country location of the hosts, or second-level domain, showing ownership of hosts.

2.3.1 Spanning tree plots

Though poster-sized versions of this map were quite beautiful (and quite popular), they did not really meet the original visualization goals. The middle was a mess, and it did not look like it was possible to iterate out of it, making the resulting map not particularly useful.

The picture becomes much clearer if, instead of plotting the entire graph, the shortest path tree is computed and plotted. This is a cheat in one sense, as packets do not always take the shortest path. But the clutter in the middle cleaned up nicely, as shown in Figure 2.3.

If one consider only one shortest path to each destination, the graph turns into a tree, allowing the layout program to produce a much neater drawing. Alternatively, a graph drawing algorithm designed to lay out trees of arbitrary size[94] could have been used, which tends to be faster than the general algorithms. Many of the tree drawing algorithms, however, result in unappealing drawings, due to two features of the resulting tree: the shallow nodes have high degrees (over 100, in some cases) and deep nodes have low degrees. Most of the standard algorithms work well for trees with relatively low degree trees (around two to ten).

Running IMP's layout heuristic on the tree results in a different map. The muddle in the middle is gone. The map looks less like a neuron and more like a coral fan or a space-filling curve. It is now possible to trace individual paths from the measurement host to most destinations. The `cw.net` backbone is still spectacular, and still somewhat muddled.

About 5% of the edges of the graph are lost when this inconvenient data is discarded. The edges still show interesting communities, but much more is visible now. By eliminating a number of inconvenient edges, the map can be made more useful, and traceable by the eye.

Those missing edges can be added back in the background by drawing them in an unobtrusive color, such as light cyan. In some cases, the alternate routes show up nicely. In others, the muddle is back, but out of harm's way. Some nodes attract a number of redundant connections, which the eye can pick out easily.

What works decently for the Internet works wonderfully for Lucent's intranet. That network has "only" 3,000 announced networks (versus some 90,000 registered for the Internet at this writing.) The full map is shown in Figure 2.4.

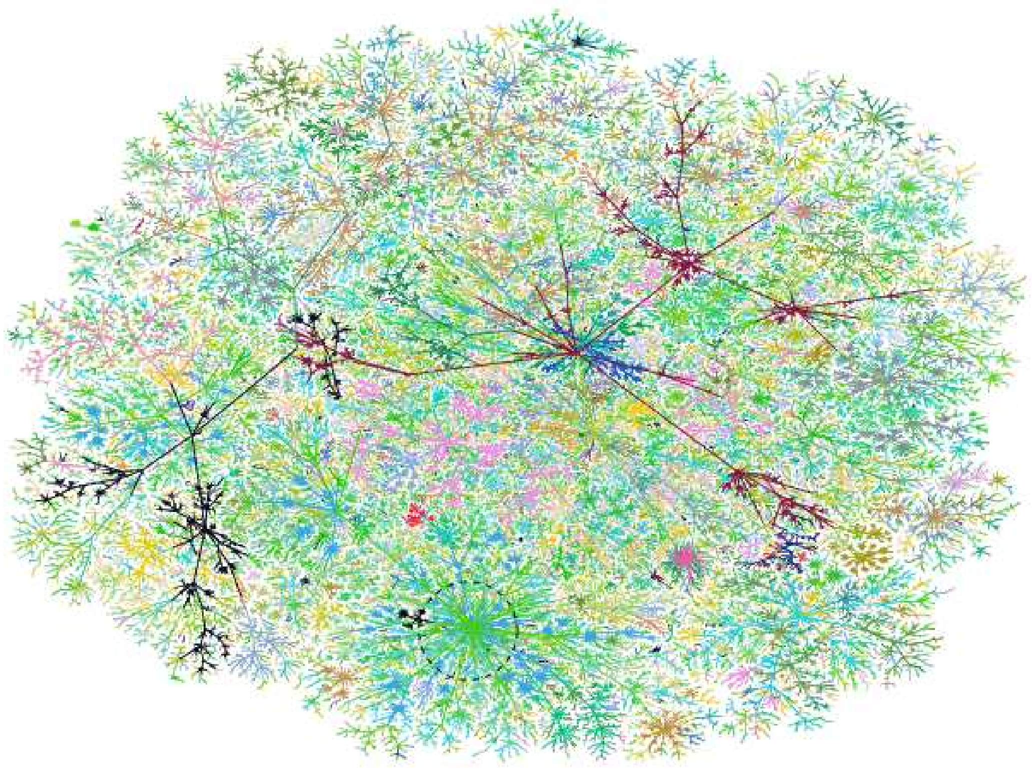


Figure 2.3: Minimum distance spanning tree for data collected on 2 November 1999. The circled star at the bottom is `cw.net`. The black foreground lines are links through net 12/8, Worldnet, one of IMP's ISPs.

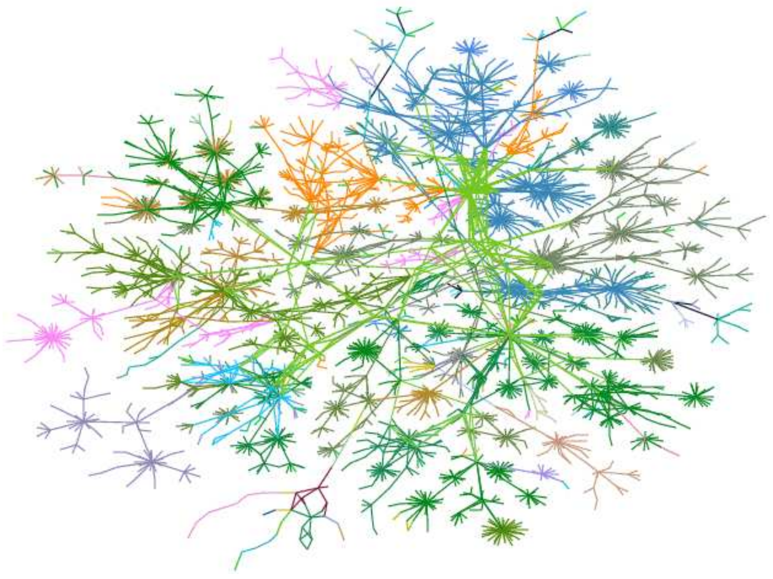


Figure 2.4: Lucent's intranet as of 1 October 1999.

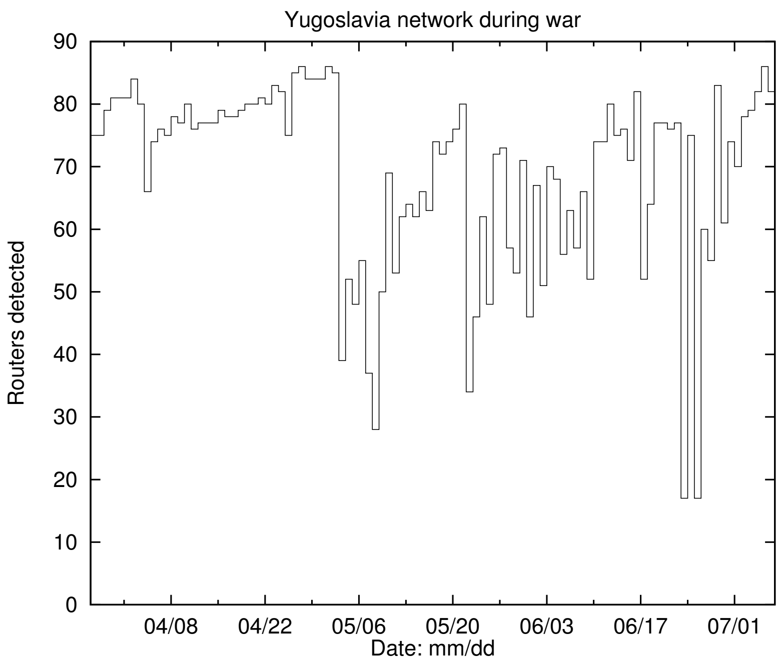


Figure 2.5: The number of reachable routers in the `.yu` domain over the course of the armed conflict.

2.4 Watching Networks Under Duress

Internet monitors have detected major disconnections before: there were stories of ping utilities that incidentally mapped the extense of the Internet outage caused by the Loma Prieta earthquake using pings. IMP's data captured one aspect of the NATO bombing of Yugoslavia in the spring of 1999.

During the first month of the war, few, if any, Internet links were cut. When, in early May, the bombing moved to the power grid, and the resulting disconnection is clearly shown in Figure 2.5. The connectivity returned slowly. Incidentally, the reachable routes in neighboring Bosnia also declined. One might infer (correctly) that Bosnia relies largely on the Yugoslavian power grid.

Figure 2.6 and Figure 2.7 compare May 2nd and May 3rd of the NATO bombing. It is interesting to note two large spiny "Koosh" balls in the upper right of the map have been significantly reduced. This would seem to imply that although the core routers at the center of the "Koosh" balls were not directly damaged, many of the outlying routers were affected, possibly through power loss.

The maps also reveal that there appear to be only a few distinct routes into the Balkans from the measurement host. The power of the mapping technology is quickly apparent when viewing the limited number of gateways that appear, showing the connectivity of Yugoslavian domains with the rest of the Internet.

Measuring the network topology detected the results of distant damage in an semi-automated way. It is unlikely that this is the first time such military uses have been considered. Related techniques will doubtless be useful for monitoring the extent of other natural disasters, particularly in well-connected parts of the world.

One of the largest difficulties with using such monitoring to examine events is that a small fraction of the hosts on the Internet are affected. Thus, it is difficult to observe when considering the entire network. In this case, the set of hosts of interest was all the hosts in a country and nearby countries. This is a relatively simple search to perform. If the set is all the hosts in a state or a city, this becomes more difficult. For

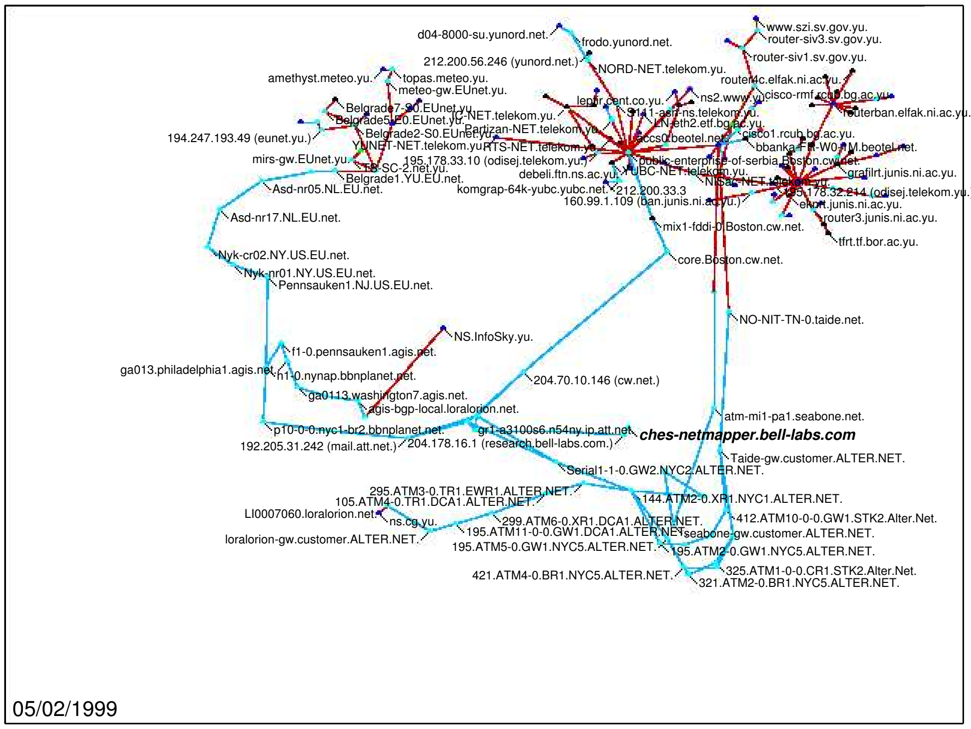


Figure 2.6: Map of paths to the Yugoslavian networks on May 2, colored by network.

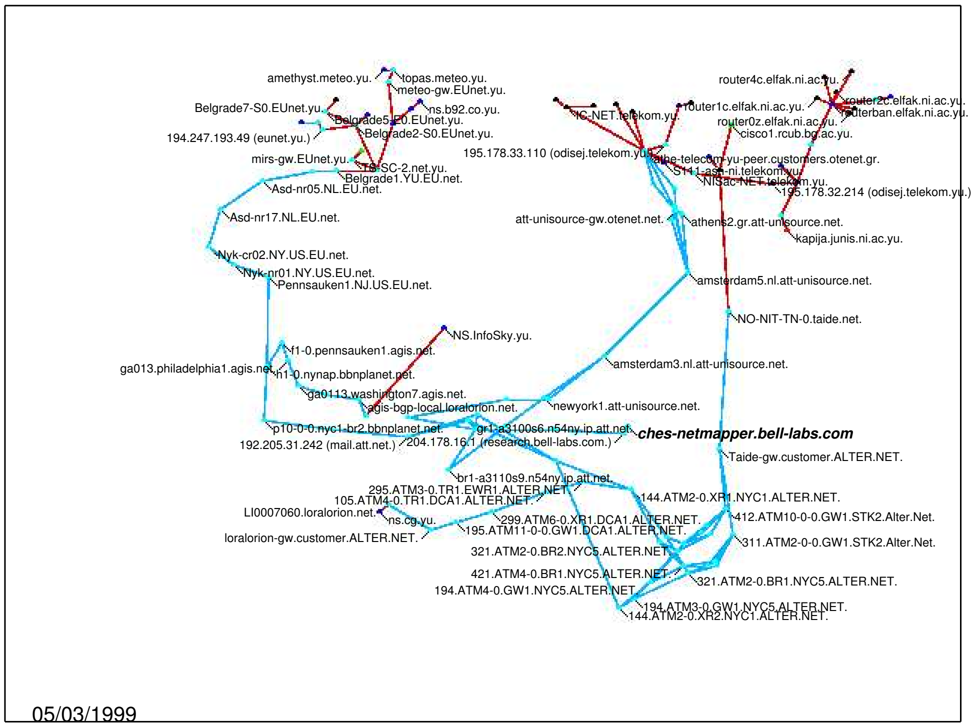


Figure 2.7: Map of the Yugoslavian Internet on May 3, colored by network. The main hubs in the upper right are still reachable, but they have lost a lot of leaves.

example, measuring the effect of the terrorist attack on the World Trade Center would require finding hosts in the New York City and the area. Geolocation techniques should be able to address this problem, although they generally focus on the problem of determining the location of a host, rather than the opposite problem of determining the hosts and networks in an area.

2.5 Related Work

There are a number of Internet data collection and mapping projects underway. Some have been running for a number of years, such as John Quarterman's Matrix Information and Directory Services[61], which includes the "Internet weather report." Martin Dodge has collected many representations of networks at Cybergeography[25]. Pansiot and Grad[68] mapped paths to 5,000 destinations. The Mercator project at USC[41] tried to get a picture of the Internet at a given instance in time.

In terms of long-term mapping, k claffy and CAIDA are collecting a number of metrics from the Internet with *skitter*[59]. They have mapped the MBone, and collected path data to major web sites. IMP is designed to map each known network, which maps to everything that exists, rather than everything that is used (i.e. the web servers). IMP's goal is to discover every possible path, not just those in use.

Spring, Mahajan, and Wetherall's Rocketfuel[92] focused on a single ISP. By focusing on one ISP at a time, they hoped to map it accurately and completely. Rocketfuel mapped from multiple locations, using BGP tables to determine which paths go through the ISP of interest. Rocketfuel was concerned with a snapshot, not daily measurement of the ISP.

Lyon's Opte[58] attempts to measure the Internet in a single day. It is an amalgam of freely-available tools, combined together to produce a program to determine the topology of the Internet. Opte traces to each class C (/24) network, although it uses sampling techniques to discard all the class C in a class B network, either because they are not used on the Internet or because they would all result in the same path. Because it calls traceroute for each scan, Opte is computationally expensive (Opte reports a load of 60 during scanning) and difficult to control when compared to IMP, skitter, and Rocketfuel, which use traceroute tools specially written to run many traceroutes in parallel.

Internet maps are often laid out on the globe or other physical maps. The desire to map the Internet to geography is compelling, but it tends to end up with dense blobs of ink on North America, Europe, and other well-connected regions[3]. However, connections to distant and more sparsely connected regions can be represented nicely, *c.f.* Quarterman's map of connections to South America.

The problem with this method is the well-connected areas remain thoroughly inked, without a prayer of tracing paths through them. One approach is to simplify the map, showing connections by autonomous systems rather than individual routers. This is akin to showing the interstates on one map, and then creating local maps for each state. However, the AS connectivity graph is, proportionally, more connected than the IP graph. Thus, that graph is even more difficult to draw legibly.

Interactive visualization tools can aid in navigating a database like ours. One can zoom, query, and browse at will. It is hard to see the entire net clearly on a screen: there are far too few pixels. However, H3Viewer[64, 65] is one tool that looks like a good start. It displays a spanning tree of the graph and allows the user both to navigate the tree and also view the non-tree edges.

[3]Producing a map distorted based on Internet connectivity in order to alleviate this problem might be an interesting problem for some cartographer.

2.6 Corporate Networks

The usefulness of network mapping is not limited to mapping the Internet. Corporate networks often suffer from loosely-controlled growth, mergers, unauthorized connections, and general chaos. Network administrators are primarily concerned with making the network work properly. When faced with a problem, their fix may result in unexpected changes to the network. As long as the changes are not disruptive to the network, they can go undetected for long periods of time.

When IMP began, it was run from Bell Laboratories in Lucent Technology. Daily scans were run of Lucent's network. The maps helped debug routing tables, which contained route announcements for networks like `lsu.edu` and the U.S. Postal Service. Although it is not clear if either Louisiana State University or the U.S. Postal Service were actually connected to the network, their CIDR blocks should not have appeared in the corporate routing tables.

The somewhat-surprising usefulness of the mapping and related technology resulted in Lucent forming a start-up venture to commercialize the technology. This venture, called Lumeta[57], spun-off of Lucent in October, 2000, taking the Internet Mapping Project with it. The commercialized version of the software, with many features added, is offered as IPsonar by Lumeta.

One of the features added is the ability to display data using the graph layout. By coloring the edges and nodes, the map can show insecure regions, new acquisitions, rare domains (domains with few mapped hosts), unexpected networks, and many other forms of data. The data, and the visualization there of, enables networks administrators to debug, optimize, and secure their corporate networks.

2.7 Conclusion

The Internet maps, while seemingly less useful have certainly excited the media, who lacks good visuals of the Internet[10, 103]. From a less scientific standpoint, the maps are interesting to look at, and one publisher created a poster out of it[71].

A number of researchers have picked up the routing database and run it through their visualization tools or run graph-theoretic analyses of it, and at least one paper has resulted thus far[19]. As the data collection began in August, 1998, it provides a good deal of information about routing for a longer period of time than most routing studies to date have employed.

Chapter 3

IP Resolution[1]

Chapter 2 focused on the methodology used by the Internet Mapping Project[17] to measure the topology of the Internet. There are several other ongoing projects to determine the IP-level topology of the Internet[41, 59, 68, 92]. Although they vary in specifics, all of these projects use traceroute[46] to determine paths and then take the union of those paths to yield a topology. They all suffer from the same problem: IP aliasing.

IP aliasing is a result of how traceroute works. Recall that traceroute uses the source of ICMP error messages to identify the hops along a path. The source of the messages is expressed as an IP address. Hosts, in particular routers, often have multiple IP addresses. If a host responds with different IP addresses in different paths, then that host will appear multiple times in the topology. This behavior, known as IP aliasing, distorts the resulting topology. Routers commonly have one IP address for each physical or virtual interface. Hosts may report any of their IP addresses as the source IP address of traceroute responses[73]. Some hosts choose the responding IP address based on the interface the packet came in, some on the interface the ICMP error message goes out, and some always choose the same IP address. If a host exhibits either the first or second behavior, two traceroutes passing through the host may contain different IP addresses.

IP aliasing distorts the produced topology in several ways. Adjacency is incorrect, as two adjacent hosts with multiple IP addresses are often not adjacent for all pairs of IP addresses belonging to those hosts. This causes hosts to be farther apart in the topology than they are in reality. IP aliasing also distorts distributions in a measured topology. It obviously inflates the size of the topology, increasing the number of hosts and links. Because hosts are farther apart in the topology, distance distributions are also affected. Degree distribution is also distorted, although it is more difficult to determine the net effect of IP aliasing on degree distributions.

In addition to distorting the topology, IP aliasing also makes it difficult to determine whether two paths are the same. Analyzing path symmetry and route stability requires determining if two paths are the same. If two paths differ only in the selection of the IP addresses that represent the hops, the paths are considered different, despite the fact that they describe the same path within the network. Without correcting for IP aliasing, any measurements of path symmetry or route stability understates the true behavior of the network.

The problem of determining the set of IP addresses belonging to a host is known as the IP alias resolution problem. Although IP alias resolution is not a new problem, previous work was focused on producing a topology. The bulk of the focus was on minimizing the number of links and routers not included in the topology. Little effort was made to analyze the IP alias resolution technique used, much less to compare it against other possible techniques.

[1]This chapter represents work originally published as “Eliminating Machine Duplicity in Traceroute-based Internet Topology Measurements” as a technical report at Carnegie Mellon University[6].

This chapter considers several techniques for IP alias resolution. There are three major known techniques for detecting IP aliasing: UDP, IPid and rate limitation. The UDP technique is the oldest approach, first proposed by Pansiot and Grad[68]. This approach looks at the source address of ICMP Port Unreachable responses to UDP packets. It was also used by Govindan and Tangmunarunkit[41], Barford, et al.[3], and Keys[52]. The IPid technique looks for correlated behavior in the IP identification field of the IP header[74]. It was used by Spring, et al.[92]. The rate limitation technique looks for correlation in rate limitation behavior between IP addresses. Although the rate limitation technique was briefly mentioned by Spring, et al.[92], they rarely employed it.

This chapter presents algorithms embodying each technique, analyzing its efficacy, and compares the presented algorithms with each other and with some previous algorithms. Specific algorithms are designed to work on large sets of IP addresses. Two methods are presented to divide the input set into smaller sets: TTL and adjacency. Splitting algorithms employing these splitting methods reduce the number of packets required by IP alias resolution algorithms, but may discard valid pairs of IP addresses. The efficacy of each technique and algorithm is analyzed in four dimensions: cost, coverage, accuracy, and unique coverage. Cost is defined by the number of packets sent and the amount of time taken to test a set of approximately 400,000 IP addresses. Coverage is the completeness of the output. Accuracy is derived from the number of pairs of IP addresses belonging to different hosts that are labeled as belonging to the same host. Unique coverage is how much the technique finds that is not found by any other technique or algorithm.

Section 3.1 gives the details of the experimental setup and methodology. Section 3.2 explains the splitting methods and algorithms, while section 3.3, 3.4, and 3.5 give the specific IP alias resolution algorithms. Section 3.6 lists the results of the experiments, and section 3.7 presents conclusions.

3.1 Methodology

Before giving algorithms, the requirements of an IP alias resolution algorithm, including inputs and outputs, must be defined. The input to an IP alias resolution algorithm is a set of IP addresses of interest. An algorithm can send packets of any type to any set of IP addresses in any order. The output of an algorithm is a set of pairs of IP addresses, where the IP addresses in a pair are believed to belong to the same host. The output may contain IP addresses not within the original set of IP addresses. An IP address may appear in multiple pairs, allowing the algorithm to express the belief that a host has three or more IP addresses.

It is often useful to have disjoint groups of IP addresses, where each group represents the IP addresses believed to belong to one host. The set of pairs can be mathematically transformed to disjoint groups by taking the equivalence classes of the transitive closure of the equivalence relationship implied by the pairs. Algorithmically, this can be done starting with each IP address in a group by itself. For each pair, combine the groups containing the two IP addresses of the pair, if the two IP addresses are not already in the same group. This process yields the desired disjoint groups. Note that this process creates new pairs of IP addresses believed to belong to the same host that were not in the original set. This will discussed in Section 3.6.

One use of these disjoint groups is refining the network topology resulting from traceroute measurements. This is done by collapsing the IP addresses believed to belong to the same host to a single node. After this combination, the graph is reduced to a simple graph again by removing both self-loops and duplicate instances of edges.

For an algorithm to be accurate, it must have minimal effect on the network. There are two ways an algorithm could affect the network: technical and political. If an algorithm causes the network to rebalance load or deal with host or link failure, the output will be incorrect. If an algorithm causes network operators to become upset and block the measurement host, the output will also be flawed. The effect on a network is difficult to quantify in general. Because all methods, techniques, and algorithms herein employ high-port UDP packets, the effect can be quantified by the packet rate and the length of time the algorithm runs.

An IP alias resolution algorithm should send no more than 2,000 packets per second. A packet rate of 2,000 packets per second corresponds to 60% of an T1. This means that the algorithm is unable to overload any T1 lines in the network. Moreover, the technique is usable from measurement hosts that have only T1-speed connectivity. This bound on connection speed means that almost anyone can use the algorithms.

The time limit for the algorithms to run was selected as one day for an input set of 400,000 IP addresses. The input set limitation is required to analyze runtime. An input set of 400,000 IP addresses represents roughly two and a half times the number of router IP addresses found each day by the Internet Mapping Project[18]. Thus, it should be sufficient for most purposes. Limiting the run time mitigates several issues.

Firstly, the output of the algorithm attempts to characterize the network over the entire measurement time. Long time limits mean that the algorithm is trying to characterize a dynamic network over a long time with a single output. Few configuration changes resulting in backbone router migrating between IP addresses are made on an average day. Thus, this time limit ensures that the network is mostly static over the measured time period.

Secondly, for an algorithm to work on a pair of IP addresses, the corresponding host must be on for the entire run of the algorithm configured with that pair of IP addresses. Requiring an algorithm to run in less than one day limits the amount of network downtime confounding the algorithm.

Thirdly, the longer an algorithm runs, the more packets it may send and the more likely it is to annoy a network operator. The limit of one day and 2,000 packets means that no more than 172.8 million packets (10 gigabytes) are sent, or 432 packets (25 kilobytes) per IP address. In practice, the actual values are much lower, but even 432 packets over a day to an IP addresses should have minimal effect on the host.

3.1.1 Goals

This chapter explores techniques, and algorithms embodying those techniques, for finding sets of IP addresses each belonging to a single host. Given these sets, the topology errors from IP aliasing can be eliminated by combining the nodes representing different IP addresses of a host. Each technique depends on behavior of hosts not specified by standards. Previous algorithms either have poor accuracy or require packet counts that are quadratic in the number of IP addresses to be tested. For a sending rate of 2,000 packets per second (approximately T1 speeds), testing 1,000 IP addresses using an algorithm that sends two packets for each pair would take a little more than eight minutes, which is reasonable. However, it does not scale to testing all IP addresses of routers on the public Internet. This chapter uses a set of 380,000 such IP addresses, which would take over two years to test. Although higher packet rates would decrease the amount of time required, quadratic algorithms obviously scale poorly to large data sets. The goal is to obtain scalability while minimizing the reduction in accuracy. This chapter presents specific ways to test for IP aliasing on Internet scale using a reasonable number of packets in a reasonable time period.

3.1.2 Experimental Setup

All experiments were done over a T1 connection connected via UUNet. As mentioned above, the packet rate limitation mentioned above of 2,000 packets per second represents about 60% of a T1, as replies to the test packets contain 56 bytes. The total utilization of the local T1 was kept under 85%. This means there was not significant local packet loss, which would affect both the experiments herein and other local traffic.

The input set of IP addresses came from hops within 77 days of traceroute data from the Internet Mapping Project[18]. After discarding unroutable and private address space, the original input set consisted of

587,828 IP addresses, although only 386,157 were found to be responsive to UDP packets. The traceroutes were daily runs from May 2002 to July 2002, consisting of about 240,000 individual traces per run. This input set was selected as both representative and rich in IP aliasing. The goal is to improve topology measurements based on traceroute, which is exactly the source of the IP addresses. Moreover, many hosts were expected to appear multiple times under different IP addresses due to route changes between days.

The fact that only 66% of the IP addresses responded to UDP packets was surprising. This is a result of the traceroutes from which the IP addresses came employing ICMP Echo Request packets, not UDP packets. 92% of the IP addresses were responsive to ICMP Echo Requests. Thus, most (76%) of the IP addresses that failed to respond to the UDP packets were either behind a firewall that blocked the UDP packets but not the ICMP Echo Request packets or were configured to respond to ICMP Echo Request packets but not UDP packets. The other 24% of the non-responders (8% of all the IP addresses tested) did not respond for some other reason, such as being turned off, being retired, losing connectivity, or routers that forward packets but cannot, either due to firewalls or residing on unannounced networks, be directly addressed.

3.1.3 Packet Collection

From the responsiveness results, it would be desirable to use ICMP packets to test. However, all presented algorithms use UDP probe packets to attempt to elicit ICMP port unreachable responses. This is a coincidence, not a design constraint. ICMP Echo Request packets are generally treated differently than UDP packets. This is partly because line cards often respond to ICMP packets, rather than the router itself. The specific reason why ICMP Echo Request packets will not work for each method and technique will be given later. Using TCP packets would be more invasive to the measured systems, probably perform worse, and introduce additional complexity by introducing new response types to address.

When collecting responses to packets sent to a large set of IP addresses, there are two complicating features. First, responses must be matched with requests. As UDP packets elicit ICMP port unreachable responses, which have a partial copy of the original packet within them, this does not seem especially difficult. Unfortunately, the destination listed within the packet enclosed in the response is not always the same as the destination of the original probe packet. One cause of this behavior is improper or incomplete network address translation (NAT). Because of this, matching is done using the destination port in the original UDP packet.

The second problem is that not all packets elicit responses. Non-response can be caused by routing problems, packet loss, response rate limits, firewalls, and hosts being disconnected from the network. Thus, the packet collection algorithm must make multiple attempts in order to obtain reliable response rates.

Algorithm 3.1:

```
1. Let $S$ be the set of IP addresses
2. Send a packet to every IP address in $S$
3. Output $R$, the responses collected from step 2
4. If $\|R\| < \frac{1}{3}\|S\|$ then exit
5. Remove all IP addresses of $S$ which yielded a response in $R$
6. Goto step 2
```

As shown as Algorithm 3.1, the packet collection algorithm works in passes. Each pass tests every IP address for which the algorithm does not yet have a response. Passes continue until some pass obtains responses from less than a third of the IP addresses tested. This algorithm gives over a 93% response rate (median and mean of seven runs), with 83% of the IP addresses being reliable responders (responding to seven of seven runs). This algorithm will not obtain responses from IP addresses that have long-term (minutes) reasons to not respond, such as firewalls. It will obtain responses from most IP addresses with packet loss, rate limits, and transient routing problems.

3.2 Splitting Methods

Splitting is the process of dividing the input into smaller sets that can be tested independently. Reducing the set size makes quadratic IP alias resolutions algorithms run more efficiently. The idea is to decrease the size of the sets with a small amount of effort. While some IP addresses may appear in multiple sets, the total number of pairs of IP addresses to test should be less than the number of pairs from the original set. Two methods to split sets of IP addresses were employed: TTL splitting and adjacency splitting.

DNS splitting, used by RocketFuel[92], uses DNS names to divide the set. DNS methods are dependent on knowledge of DNS naming schemes and are sensitive to naming errors[93]. As such, they can be nearly-infinitely improved by adding knowledge of more autonomous systems. This makes analyzing a particular DNS splitting method nearly meaningless.

3.2.1 TTL Method

TTL splitting uses the eight-bit "Time to Live" (TTL) field from the IP header. This field is decremented by one for each hop a packet traverses. If the field reaches zero before the packet reaches its destination, the packet is discarded and an error message may be returned to the source of the packet. This insures that packets on the Internet have limited lifetime.

Hosts initialize the value of the TTL independent of the destination of the packet. Moreover, most hosts decide which interface to send out a response packet based on the destination without regard to which interface received the original packet. It follows from these two behaviors that sending packets to two IP addresses of such a host will result in two response packets with the same TTL. If sending packets to two IP addresses results in response packets with different TTLs, the two IP addresses do not belong to the same host.

TTL values are dependent on two values: (a) initial TTL selection and (b) length, in hops, of the path from the IP address to the measurement host. Because few initial TTL values are used in practice, IP addresses in the same TTL-value bucket are, for the most part, the same distance from the measurement host. Thus, TTL splitting roughly divides the IP addresses into groups based on the length of the path from the IP addresses to the measurement host. This is exploited further in Chapter 4.

Algorithm 3.2:

1. Use Algorithm 3.1 to collect responses from each IP address
2. Discard any IP address that does not respond
3. For each value, output a group containing IP addresses whose response had that TTL value

One problem with this approach is that routes may change in the middle of an experimental run. Thus, TTLs coming from the same host may differ, either due to a true routing change or load balancing. Fortunately, load balancing over paths of different lengths is unusual and most routes are static over reasonable time spans (minutes to hours)[70].

A larger problem with this technique is that the TTL field has only 255 possible values, and the distribution is highly skewed, as show in Figure 3.1. Initial TTL values are not distributed uniformly: There are six common initial TTL values: 30, 32, 64, 128, 150, and 255, with 255 being the most common by a factor of almost four. There is a large spike of packets observed with TTLs of 242 and 243, corresponding to a large

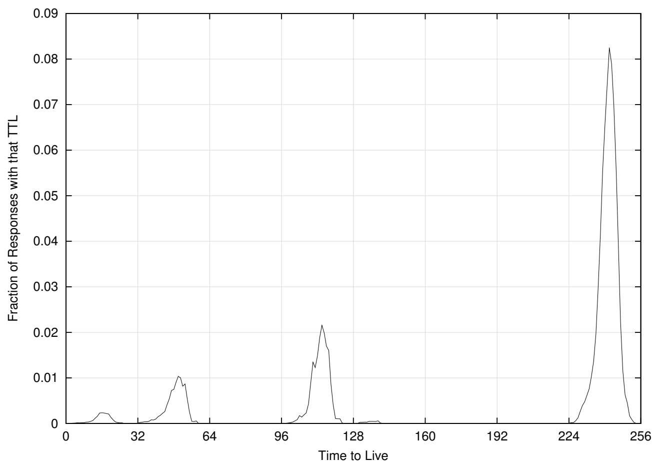


Figure 3.1: Observed distribution of TTL values for packets from 380,000 IP addresses.

number of hosts being 12 and 13 hops away. TTL values of 242 and 243 each represent about 8% of the IP addresses, making this method relatively coarse. Performing multiple tests at different times from the same location or from different locations does not significantly decrease the coarseness of this technique, as the distances from two IP addresses within the same autonomous system to any measurement point at any time are highly correlated.

The coarseness of TTL splitting makes it unsuitable for direct measurement. However, it does provide a method to split the data set into smaller sets. Since the largest output set is less than a twelfth the size of the input set, TTL splitting reduces the total amount of work to be done by a quadratic technique by at least a factor of twelve, a significant improvement.

3.2.2 Adjacency Method

Adjacency splitting uses information within an uncorrected topology to select likely pairs of IP addresses to test. In particular, adjacency splitting produces sets of size two representing all pairs of IP addresses that are two hops apart in the induced topology. This method is based on the premise that two IP addresses of the same host are likely to both be adjacent to some node. Consider adjacent hosts A and B and the scenario of Figure 3.2. On day one, the daily scan sees host A uses A_1 in its time-exceeded responses and host B uses B_1 along a traceroute, as shown in 3.2(a). The next day, host A changes its decision and uses A_2 instead, but host B does not change its decision, as shown in 3.2(b). Host A may change which IP address to use because the traceroute comes in on a different interface or because its routing table has changed and now has a different next hop for the ICMP response. Both of these can occur without a corresponding change in host B. In such cases, A_1 and A_2 will both be adjacent to B_1 within the resulting topology, as show in 3.2(c).

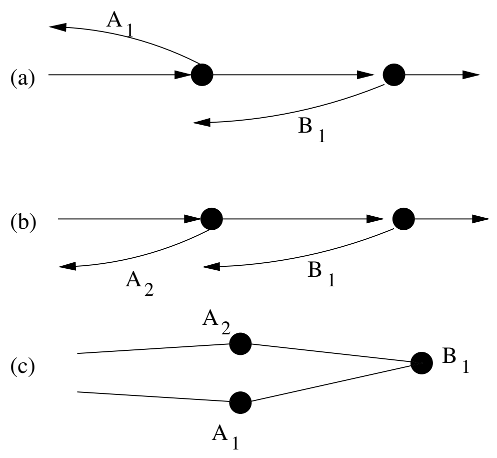


Figure 3.2: Example of how IP addresses of same host become adjacent. Traceroutes (a) and (b) result in topology (c).

Algorithm 3.3:

```
1. Let $S = \emptyset$
2. For each IP address $A$ in the topology
    (a) For each pair of IP addresses $B$ and $C$ adjacent to $A$ in the topology, add $(B, C)$ to $S$.
3. Remove duplicate pairs from $S$
4. Output $S$
```

The obvious algorithm would be to create a set for each IP address consisting of all IP addresses adjacent to that IP address. However, some pairs of IP addresses appear in many such sets, which would result in testing those pairs many times. To avoid this problem, the algorithm, shown as Algorithm 3.3, outputs all pairs of IP addresses that share an adjacent IP address.

The data collection required to obtain the topology is not considered. IP alias resolution is most commonly used to fix topology measurements. Because the data collection required to obtain a topology is done in most cases where this algorithm would be used, that data collection is not counted as part of the cost of this algorithm.

The scenario that makes two IP addresses of a host adjacent to some IP address is less common than the scenario for TTL equality. As such, this method is expected to have worse coverage than TTL splitting. The major advantage of this method is that the number of pairs to test is nearly linear in the number of IP addresses in the input set, making it a useful splitting algorithm for techniques that scale poorly.

3.3 UDP Technique

The problem of IP aliasing was first observed by Pansiot and Grad when mapping multicast trees[68]. Their techniques for IP alias resolution sent UDP packets to non-listening ports to find pairs of IP addresses belonging to the same host. When they sent a UDP packet to a port where they did not expect a service to be listening, hosts receiving these packets responded with ICMP unreachables, specifically port unreachable. Pansiot and Grad noted that the source IP address of the ICMP error packet was not always the same as the destination of the UDP packet. As it is unlikely that a different host would respond with an UDP port unreachable than the destination of the UDP packet, they used such responses as evidence that the source of the ICMP error packet and the destination of the UDP packet were the same host.

This technique is nice in that it requires sending only one packet to each IP address in the input set. Because this technique scales well, it requires no splitting. Many hosts pick the source of the ICMP error packet as the IP address of the outbound interface, making the address constant over all interfaces. This technique will not work on hosts that select the source of the response packet to be the original destination. A priori, it is not clear what the relative prevalence of these two types of hosts is on the Internet.

The specific algorithm used is relatively straight-forward:

Algorithm 3.4:

1. Use Algorithm 3.1 to collect responses from each IP address
2. Discard responses from 0.0.0.0, the measurement host's IP addresses, and private IP addresses.
3. Discard responses which list a different original destination IP address than expected.
4. Group IP addresses by the source IP addresses listed in the responses

This technique is, by far, the most commonly used approach. Govindan and Tangmunarunkit[41] used a more complicated form of this technique that uses multiple packets and source routing[74]. Barford, et al.[3] used the UDP technique, although without the additional features of Govindan and Tangmunarunkit. `iffinder`[52], written by Keys uses several techniques for finding interfaces, but its most successful technique is the UDP technique.

The popularity of the UDP technique is partially due to both its simplicity and speed: only one packet needs to be sent for each IP address, ignoring retries of the packet collection algorithm. This technique cannot benefit from splitting methods, but it does not need them either.

False positives in this technique are caused by hosts responding with IP addresses that are not uniquely their own. This is most obvious when they use invalid IP addresses that are clearly not their own or when they use private IP addresses, which can be used multiple times within the network. The algorithm discards these obvious errors, but some hosts respond with IP address not uniquely their own that are not detected by the algorithm due to operating system peculiarities or system defaults.

3.4 IPid Technique

A different IP alias resolution technique, employed by RocketFuel[92], uses the IP identification field as a bucketing technique to detect multiple IP addresses belonging to the same host. Many hosts set this 16-bit header field using a global counter that is incremented by one for each packet sent. If such a host responds to two packets at approximately the same time, it will send packets with similar IPid values, even if the packets are sent to two different IP addresses of the host. Spring, et al. designed the RocketFuel algorithm to work on small data sets, as they were focused on determining the connectivity of only one ISP at a time. The presented algorithm is designed for larger data sets and includes an improved verification algorithm. The presented algorithm is mathematically compared with RocketFuel.

The IPid field provides a natural way to verify that two IP addresses belong to the same host, as the value of the IPid field within responses from the two IP addresses of the same host will be similar. Discovering pairs of IP addresses belonging to the same host is a bit more difficult. Here, packets must be sent to a large number of IP addresses and the IPid field values in the responses examined to determine which IP addresses may belong to the same host. For large input sets, it is difficult to detect if the first and last IP address belong to the same host.

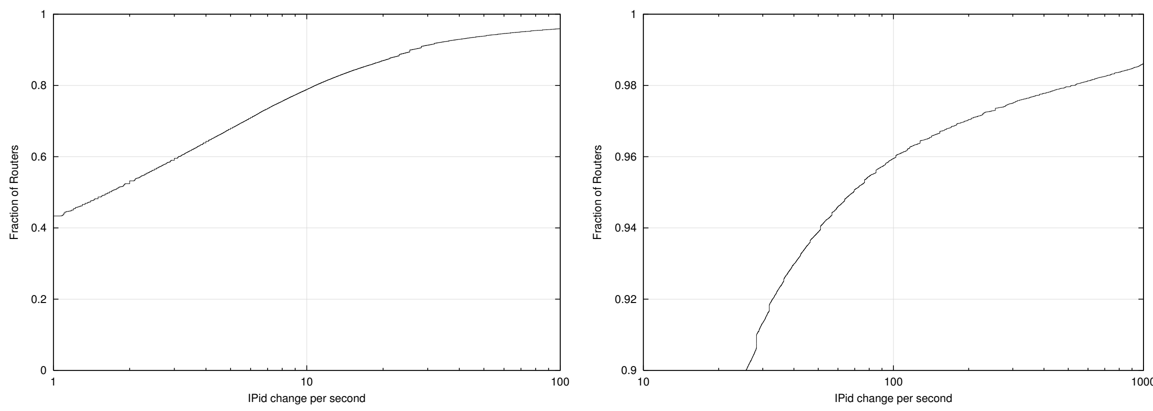


Figure 3.3: Cumulative distribution of IPid change rates for 102,225 IP addresses, mostly routers. The two plots vary only in the selected axis ranges.

Assume both the IPid change rates for two IP addresses probed t seconds apart are both below R, but the change rates are not known. Without loss of generality, presume the IPid of the first response is 0 and the IPid of the second response is d. The two IP addresses may belong to the same host if $d < Rt$. If initial IPid values are random and the two IP addresses are not the same host, this happens with probability $\frac{Rt}{65536}$.

Figure 3.3 shows the distribution of IPid change rates of IP addresses reachable over the Internet. As the rate of change was measured over about 12 seconds, rates above 5,000 were incorrectly measured, but the number of such hosts is extremely small. Responders that send constant IPid values were discarded (total of 1,011 (1.0%), not included in the 102,225). Most of the high rates are hosts that increment in big-endian fashion or pick IPid randomly, representing less than 2% of the whole. This figure shows that the majority of the tested hosts have slow IPid change rates: 79% change at slower than 10 per second, and 96% change at slower than 100 per second. The techniques herein focus on IPid change rates under 100 per second ($R = 100$). The IPid change rate of an IP address can be volatile[4]. While it is assumed the change rate of each IP address is bounded, it is not assumed that it is known.

Let K be the number of IP addresses to test, and P be the packet rate. Assume that the current IPid counters for all hosts are initialized randomly, that there are no pairs of IP addresses belonging to the same host, and that round trip times are zero. The time between the queries of two IP addresses (t) is the distance between them in the query order divided by the packet rate. Thus, the total number of expected false positives is:

$$\begin{aligned}\sum_{i=0}^{K-1}\sum_{j=i+1}^{K}\frac{(j-i)R}{65536P} &= \frac{R}{65536P}\sum_{i=0}^{K-1}\sum_{j=i+1}^{K} j-i \\ &= \frac{R}{65536P}\sum_{i=0}^{K-1}\frac{i(i+1)}{2} \\ &\approx \frac{R}{131072P}\sum_{i=0}^{K-1} i^2 \\ &\approx \frac{R}{131072P}\frac{K^3}{3}\end{aligned}$$

$$\approx \frac{RK^3}{400000P} \approx \frac{RK}{200000P}\binom{K}{2}$$

Note that this formula only works when $\frac{RK}{65536P} < 1$, since otherwise a host's IPid counter may roll-over within the testing window.

In practical terms, if $R = 100$, $K = 400,000$, and $P = 2,000$, the formula states that there will be 8 billion false positives, 10% of all pairs. Thus, algorithms based on this technique need either multiple phases or splitting methods if they are to work on large input sets. The algorithm below does both.

3.4.1 Algorithm with TTL Splitting

The algorithm has four phases: TTL splitting (Algorithm 3.2), IPid splitting, IPid testing, and IPid verification.

IPid splitting

Algorithm 3.5:

```
1. Let $G$ be the set of sets of IP addresses from TTL splitting
2. Let $S \in G$; remove $S$ from $G$
3. Let $R = 100$, the maximum IPid change rate supported
4. Use Algorithm 3.1 to collect a packet from each IP address in $S$
5. Let $t$ be the time taken to collect packets
6. Let $S = \{A_j\}$. Let $d_j$ be the IPid received from IP address $A_j$
7. Let $N = \left\lfloor \frac{65536}{tR} \right\rfloor$, the number of sets to produce
8. For each $0 \le i \le N - 1$,
    (a) Let $S_i = \left\{ A_j | \left\lfloor \frac{Nd_j}{65536} \right\rfloor \equiv \{i, i+1\} (\text{mod } N) \right\}$
    (b) if $\|S_i\| \le 2000$, output $S_i$
    (c) Otherwise, add $S_i$ to $G$
9. If $G \neq \emptyset$, goto Step 2
```

IPid splitting, shown as Algorithm 3.5, divides the TTL-split groups into groups with 2,000 or fewer IP addresses to simplify later testing. It sends packets to each IP address and collects the IPid of the response. The IPid range 0-65,535 is split into subranges based on the maximum IPid change rate supported (R) and the time it took to collect the packets. The size of the range is the product of the maximum rate and the collection time (with round-off distributed roughly evenly across groups). A group consists of all IP addresses with IPid values within a range and the next range (for the highest range group, the next range is the first range). This means that every IP address that responded is in two groups.

Presuming that the host did not change at an average rate exceeding the maximum supported rate and the host responded for all IP addresses, every pair of IP addresses belonging to that host exists in some group. In particular, the first responding IP address of each pair is put into a group containing the last responding IP address. If a resulting group has more than 2,000 IP addresses, the process is repeated on that group. In order to decrease the size of the largest group, at least three groups must be created. Therefore, the number of IP addresses within a single test group cannot exceed $\frac{65536P}{3R} \approx \frac{21845P}{R}$, where P is the average rate of packet transmission and R is the maximum supported IPid change rate. For the experimental results given, $R = 100$ and $P \approx 2000$, which means the groups inputted into this phase cannot exceed 436,900 IP addresses. Although the input set could be run directly (barely), running TTL splitting first greatly decreases the amount of duplication of IP addresses done by IPid splitting.

IPid testing

Algorithm 3.6:

```
1. Let $R = 100$, the maximum IPid change rate supported
2. Let $S$ be an input set
3. Use Algorithm 3.1 to collect a packet from each IP address in $S$
4. Wait at least five minutes
5. Use Algorithm 3.1 to again collect a packet from each IP address in $S$
6. Let $\Delta_1 = Rt_1$, where $t_1$ is the time taken to collect the packets the first time
7. Let $\Delta_2 = Rt_2$, where $t_2$ is the time taken to collect the packets the second time
8. Let $S = \{A_j\}$. Let $d_j^k$ be the IPid received from IP address $A_j$ on the $k^{th}$ run ($k \in \{1,2\}$)
9. Output $\{\{A_i, A_j\} | \ \|d_j^1 - d_i^1\| \leq \Delta_1$ and $\|d_j^2 - d_i^2\| \leq \Delta_2\}$.
```

IPid testing, shown as Algorithm 3.6, takes sets of 2,000 or fewer IP addresses and proposes pairs of IP addresses that may be aliases of each other. Packets are sent to every IP address and the IPid of the responses are collected. Pairs of IPid values are then examined to determine if they are consistent with a host whose IPid counter is changing no faster than the supported rate, with the assumption that the reverse path delays do not exceed 200 milliseconds (this allows for a good deal of jitter). Pairs of IP addresses that meet this criteria are outputted. From the equation in the previous section, the percentage of false positives would be expected to be quite low. However, this ignores the fact that this phase is dependent on the previous splitting work, since 40% of the hosts have IPid values that change at a rate slower than 1 per second (see Figure 3.3). Due to the number of false positives, IPid testing is done twice on each set with at least five minutes between pairs and only pairs that appear in both tests of a set are considered. In practice, the first round of packet collection is done for each set. When all sets have been tested once, they are then tested again. This avoids spending five minutes per set sending no packets.

IPid verification

IPid verification tests pairs of IP addresses to determine if they belong to the same host. IPid verification is done in groups of 2,000 pairs such that no IP address appears in two pairs. To test a pair, packets are sent to the first IP address, then to the second, then to the first, and then to the second, with a one second delay between each packet sent to a single pair. The IPid values of the responses must be consistent with the assumption that the host responding to both addresses does not change its IPid quickly. In particular, the IPid of the responses must be monotonically increasing and separated by no more than 400. This interlacing ensures that regardless of the initial values, the pair will be discarded if the hosts' IPid values are not moving at similar rates. For IP addresses with slowly-changing IPid values, pairs with similar initial IPid values (most of the false pairs tested), the interlacing is likely to reveal two responses with the same IPid value.

IPid verification, shown as Algorithm 3.7, has four possible results for each pair: accept, rejection, failure (did not receive responses to all packets), and skipped (could not find a test set to add the pair to that did not already contain one of the IP addresses of the pair). Verification sweeps are done through the incomplete (not accepted or rejected yet) pairs until at least half of the non-decisions are failures, at which point the remaining unvalidated pairs are considered rejected.

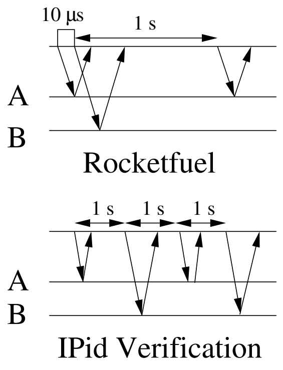


Figure 3.4: Comparison of RocketFuel's algorithm and IPid verification. IPid verification's fourth packet improves the false positive rate from 1 in 4800 to 1 in 37000.

Algorithm 3.7:

```
1. Let $G = \{p_i\}$ be the input set, where $p_i$ are pairs.
2. Let $\Delta = 400$
3. Set $S = \emptyset$
4. If $||S|| = 2000$ or $G = \emptyset$,
    (a) Set the packet rate to $||S||$ packets per second
    (b) Output a packet to the first IP address of each pair in $S$
    (c) Output a packet to the second IP address of each pair in $S$
    (d) Repeat Steps 4b and 4c
    (e) For each pair $p \in S$,
        i. If pair $p$ did not get four responses, mark $p$ as *Failed* and consider the next pair
        ii. If $d_2 - d_1 \bmod 65536 > \Delta$ mark $p$ as *Rejected*
        iii. If $d_3 - d_2 \bmod 65536 > \Delta$ mark $p$ as *Rejected*
        iv. If $d_4 - d_3 \bmod 65536 > \Delta$ mark $p$ as *Rejected*
        v. If $p$ has no mark, mark $p$ as *Accepted*
    (f) Set $S = \emptyset$
5. If $G = \emptyset$, exit
6. Let $p \in G$ be a pair of IP addresses randomly selected from the pairs of $G$.
7. If either IP address of $p$ is in some pair in $S$, mark $p$ as *Skipped* and goto step 4
8. Otherwise, add $p$ to $S$ and goto step 4
```

3.4.2 Algorithm with Adjacency Splitting

The second IPid algorithm employs adjacency splitting instead of TTL splitting. The algorithm performs adjacency splitting and runs IPid verification on the resulting pairs.

3.4.3 Discussion

Figure 3.4 compares the IPid verification algorithm to RocketFuel's algorithm[92]. To compare IP addresses A and B, RocketFuel sends packets to A, then to B, and then to A again. It then checks if the response from

B has an IPid that is between the two responses from A and that the IPid response from B is within 200 of the IPid response of A. The IPid verification algorithm here sends a fourth packet to B, with the additional constraint that the second response from A must have an IPid that is between the two responses from B.

IPid verification's false positive rate is much better than RocketFuel's. Assuming constant IPid rate change, random IPid starting values, and the distribution of IPid rate changes shown in Figure 3.3, the probability of both RocketFuel conditions holding true for a false pair of IP addresses is 1 in 4800. Based on the same assumptions, IPid verification has a theoretical false positive of 1 in 37000[2]. Because this analysis presumes no delay, the improvement of IPid verification over RocketFuel is conservative. The effect of introducing delay and jitter depends on the separation between packets. IPid verification would be affected little, because the separation between packets is one second. The effectiveness of RocketFuel, because of the ten microsecond separation, would be more adversely affected by the introduction of delay and jitter.

The false negative rates of both algorithms are high compared to the density of valid pairs, which is about 1 in two million based on no splitting. RocketFuel improves this false negative rate by running their algorithm twice, with ten seconds between runs. If the two IP addresses are different, ten seconds does not cause the two IPid counters to change much: the false positive rate remains high, at 1 in 26000. IPid verification is only run on pairs of IP addresses that had similar IPid values three times before, once during IPid splitting and twice during IPid testing. The combined false negative rate, while difficult to quantify, is certainly much better than the false negative rate of IPid verification of 1 in 37000.

3.5 Rate Limit Technique

One (unfriendly) idea for determining if two IP addresses belong to the same host is to bring down the host belonging to one of the IP addresses and determine if the other IP addresses still responds. This is obviously extremely antisocial. However, this general technique does not require the entire host be brought down: it needs only that the host stops responding to probe packets. Some hosts on the Internet have limits on the rate that they will send certain replies. UDP packets sent to ports on which no service is listening normally elicit ICMP port unreachable responses. However, many hosts restrict the responses to no more than one ICMP port unreachable response per second, without regard to which interface or IP address those packets are received or sent.

This rate limitation is most commonly implemented as sending no more than one packet every second, as determined by the host's clock. Thus, packets received 100 milliseconds apart will elicit a response only if the clock ticks (increments the second counter) between receiving the packets, which would occur 10% of the time. This implies a technique by which it can be determined if two IP addresses belong to the same host. Because the clock ticks occurs at an unknown time, such a technique technique must do multiple experiments to avoid being fooled by a clock tick and drawing the false conclusion that the two IP addresses do not belong to the same host.

To determine how frequent this rate limitation is in practice, five UDP packets were sent in quick succession to each of the 386,157 IP addresses: five packets to the first IP address, five to the second, etc. Multiple responses to the same packet were ignored. Figure 3.5 summarizes the results. Three-fifths of the IP addresses rate-limit their responses and should be responsive to this technique.

This technique is time-sensitive. Sending two packets to different addresses separated by three seconds yields no information to this technique, as the clock will tick between the times at which the packets are

[2]Note that the selection of 400 as maximum separation does not significantly affect the false positive rate of IPid verification: reducing the separation bound to 100, based on the design goal of supporting rates up to 100 per second, improves the false positive rate to 1 in 43000, only slightly better.

Description	IP Count	Pct.
Tested	386,157	100.0%
Five responses	139,133	36.0%
Two to four responses	10,668	2.8%
One response	227,368	58.9%
No valid response	8,988	2.3%

Figure 3.5: Summary of rate limitation test of sending five packets to each IP address.

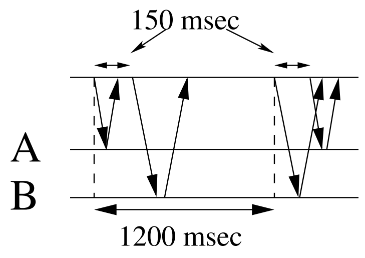


Figure 3.6: Rate limitation algorithm.

received. Thus, splitting is necessary if the number of IP addresses to test exceeds the packet rate. The naïve method would be to create test sets of 1,000 IP addresses each such that every pair of IP addresses share membership to some test set. It would take more than more than 160,000 sets to do this for 400,000 IP addresses. Presuming three seconds a set, it would take more than nine days to run one pass. The rate limitation technique is sensitive to timing errors and packet loss. Its error rate means that multiple passes are necessary to obtain meaningful results. If three passes on the entire input set must be done, the naïve algorithm would take 27 days. Using TTL splitting would reduce the process to about 41 hours, still too long. Thus, adjacency splitting was employed to be able to test the set within a day.

3.5.1 Algorithm

Algorithm 3.8:

```
1. Let $G$ be the set of pairs from adjacency splitting
2. Send five packets to each IP address (five to first IP, then five to second, etc).
3. Discard the pairs of $G$ with either IP address responding to all five packets
4. For each pair $p \in G$ left,
    (a) Let $p = \{a, b\}$ be a pair of IP addresses from adjacency splitting
    (b) Send packet to $a$, wait 150 milliseconds, send packet to $b$
    (c) Wait 1050 milliseconds
    (d) Send packet to $b$, wait 150 milliseconds, send packet to $a$
    (e) If four packets received, mark $p$ as *Rejected*
    (f) Otherwise, mark $p$ as *Accepted*
```

After adjacency splitting, the rate limitation algorithm, shown as Algorithm 3.8, tests each IP address to see if it has rate limitations. Any pair where at least one of the IP addresses does not have rate limitations is discarded as untestable. The set of pairs is divided into subsets of 150 pairs. Each of these subsets is tested separately. Figure 3.6 shows the process for testing a pair. UDP packets are sent to the first IP address in each pair and then to the second. 1,200 milliseconds after the first test started, the test is repeated with the IP addresses swapped. The packets in each test pair are sent out approximately 150 milliseconds apart. The 1,200 millisecond gap ensures that regardless of when the second tick occurs for the hypothetical host, one of the tests will not span that tick, assuming limited jitter. A test is considered successful if the first packet invokes a response and the second does not. The test is negative if four packets are received or responses are garnered both times for the second packet. Otherwise, the test is inconclusive.

Algorithm 3.8 has errors resulting from packet loss, interference between pairs, and jitter. To correct for these errors, the algorithm is repeated eight times. After the third run, any pair that consistently yields three negative results is discarded, and any pair that yields three positive results is accepted without further testing. The remaining pairs are accepted after eight runs only if there were at least two positive results and more than twice as many positive results as negative results. IP addresses that appear in more than 20 accepted pairs and groups with more than 20 IP addresses are eliminated to avoid excessive false positives due to lossy connections.

3.5.2 Discussion

The main source of false positive results in this technique is packet loss. If the second UDP packet is lost or the response is lost, the algorithm would falsely conclude that the two IP addresses belong to the same host. This is why multiple passes are run on the input set. However, multiple passes will not eliminate all false positives. This is covered in more detail in Section 3.6.4.

This general technique was mentioned by Spring, et al.[92]. However, they ran only one pass of this verification, running the IPid test twice, the second time in the reverse order of the first. If only the first packet elicited a response both times, they concluded the IP addresses belonged to the same host. In the experiments, IPid testing was inconclusive 0.3% of the time, making it unclear how often rate limitation was actually employed.

3.6 Experimental Results

The presented splitting and alias resolution algorithms will be evaluated along four axes:

1. Cost: number of packets and time elapsed
2. Coverage: effect on the original graph
3. Accuracy: number of false negatives
4. Unique coverage: number of unique pairs

Cost measures both the time taken and the effect on the network. Cost is important because the IPid approach and the rate-limit approach are quadratic, both in terms of the number of packets sent and the time elapsed. Above, it was shown that a naïve quadratic algorithm could take over a year just to send the packets. Clearly, a faster algorithm is needed that requires sending fewer packets. Minimizing the packets sent also minimizes the average number of packets that each IP address tested sees. This reduces the effect the measurements have on the network itself. Minimizing the time elapsed limits the amount of change in the network that takes place during the measurements.

Algorithm	# Sets	Implied Pairs	IPs Rem.	Coverage	Pkts Sent	Unique Pairs
UDP	45,087	131,518	59,705	38.2%	469,824	21,465
IPid w/ TTL Split	53,736	234,450	97,719	68.2%	12,861,007	7,797
IPid Ver. w/ Adj. Split	50,318	203,541	89,674	59.2%	24,910,020	9,913
Rate Limit	29,176	81,637	44,707	23.7%	9,805,909	13,887
Combined	63,875	343,720	112,851	100.0%	48,046,760	N/A

Figure 3.7: Summary of results for each general technique.

The amount of time that an algorithm takes is not purely a function of the number of packets sent. Two obvious sources of additional time are processing time and algorithmic delay. Although limited effort was put into optimizing the processing time, it represents a small fraction of the total time. Algorithmic delays, from both explicit and implicit delay requirements, comprise a majority of the additional time taken. Explicit delay requirements derive from explicit delay constraints in the algorithm. An example of an explicit delay requirement is Algorithm 3.5, IPid splitting, requires at least five minutes between testing each set. Implicit delays derive from algorithm constraints that imply slower runtimes. The rate-limit algorithm, for example, cannot test two sets at the same time if they share an IP address.

Coverage is measured in several ways: number of pairs outputted by the algorithm, number of IP addresses removed when the pairs are collapsed, and the number of sets of IP addresses that are collapsed to a single IP address. The number of pairs outputted is the most obvious measure of coverage. The number of IP address removed when pairs are collapsed approximates the amount of error in the topology fixed by the algorithm. The number of sets represents, presuming complete accuracy, the number of hosts that were discovered. The coverage percentage is the number of implied pairs discovered divided by the number of implied pairs found by the combination of all the algorithms. An implied pair is a pair of IP addresses that are collapsed to the same node after collapsing each pair of IP addresses found by the algorithm.

Accuracy is measured in terms of false negatives. Presuming perfect accuracy, each implied pair from an algorithm should appear in the output list of the algorithm. Thus, an implied pair that does not appear in the output list is consider a false negative. Such pairs represents a pair of IP addresses the algorithm implied were the same but did not detect directly. Although the technique did identify the pair of IP addresses as the same host, it did not do this explicitly, despite the fact that host is responsive to the technique. As the UDP technique is not expected to output all pairs, this analysis is not performed for that technique.

Under this definition of false negative, a single false positive can result in many false negatives. A false positive incorrectly joins two groups. Presuming random distribution of false positives, false positives are weighted towards joining large groups. By looking at the number of false pairs, false positives are included in this definition of accuracy, weighted according to their effect on the topology.

A unique pair is an implied pair that was not produced by any other technique. There are two reasons a technique has a unique pair: (a) the pair of IP addresses was responsive to this technique but not to any other, or (b) this technique had a false positive which caused the incorrect inclusion of this implied pair. For these techniques, the former is considered more likely, meaning the unique pair count for a technique gives a notion of how many pairs would be missed by not including a particular technique.

Figure 3.7 summarizes the results of the general techniques. IPs removed (IPs Rem.) is the number of IP addresses removed when the IP addresses belonging to each host are collapsed to a single node. In general, Figure 3.7 shows that the IPid algorithms perform best, but at a high cost, the UDP algorithm performs mediocre, but at a low cost, and that the rate limitation algorithm performs poorly at a high cost. More detailed discussion of the performance of each algorithm, including error analysis, follows below.

Description	IP count	% IPs[3]
IPs tested	386,157	100.0%
IPs responded	385,821	99.9%
Responses ignored	2,716	0.7%
Private source IP	2,471	0.6%
Destination IP differed	573	0.1%
0.0.0.0 source IP	5	<0.1%
Source IP of test host	3	<0.1%
Responses w/ different source	67,735	17.5%

Figure 3.8: Summary of UDP algorithm behavior.

Description	Pair Count	Pct.
Implied pairs	234,450	100.0%
Output pairs	226,653	96.7%
Not in output	7,797	3.3%
Not in validation input	2,779	1.2%
Failed validation	5,018	2.2%

Figure 3.9: Summary of IPid with TTL splitting results

3.6.1 UDP Results

The UDP algorithm run took less than 20 minutes total. Figure 3.8 shows the results. A total of 469,824 packets were sent to the 386,157 IP addresses tested (some IP addresses were tested multiple times by the packet collection algorithm). This means 1.22 packets were sent per IP address tested. Testing the 386,157 IP addresses resulted in 67,735 pairs of IP addresses.

The coverage numbers are Figure 3.7. In particular, the 67,735 pairs of IP addresses found by this technique describe 45,087 groups. Collapsing each group to a single IP address removes 59,705 IP addresses. 5,878 of the IP addresses in the output pairs were not in the original set. If those IP addresses are removed, the number of groups containing at least two IP address drops to 39,931. Although this technique found 21,465 unique pairs, it is the only technique that finds new IP addresses. Removing unique pairs that include new IP addresses leaves 14,071 unique pairs found by this technique.

As mentioned above, this algorithm can produce false positives when the IP address listed as the source of the responses does not uniquely belong to the responding host. Although this most commonly happens in the cases excluded, other hosts respond with less-obvious invalid addresses, such as 0.2.0.0, 1.255.1.110, and 2.2.2.5. Network address translation (NAT) also causes problem, as the packet may not be properly translated in both directions. If the outbound packet is NATted, but the response is not properly unNATted, the destination of the enclosed packet may differ from the original destination.

Because this technique is not expected to output all pairs of IP addresses, it is difficult to assess it's false negative rate. Its poor coverage of 38.2% is a result of hosts not being responsive to the technique. It is not clear how the techniques of Govindan and Tangmunarunkit[41] could significantly improve coverage.

Phase	Total IPs	Groups	Largest Group	No. Pairs Left	Time
TTL Splitting	379,843	160	34,954	3,676,963,861	95 min.
IPid Splitting	375,867	16,969	1,989	233,821,741	180 min.
IPid Testing, pass 1	356,064	12,115,122	2	12,115,122	120 min.
IPid Testing, pass 2	280,576	2,965,950	2	2,965,950	125 min.
IPid Validation	203,698	226,653	2	226,653	360 min.

Figure 3.10: Status of IPid with TTL splitting after each phase.

Description	Pair Count	Pct.
Implied pairs	203,541	100.0%
Output pairs	190,783	93.7%
Not in output	12,758	6.3%
Not in adjacency output	7,934	3.9%
Failed validation	4,824	2.4%

Figure 3.11: Summary of IPid with adjacency splitting results

3.6.2 IPid with TTL Splitting Results

Figure 3.10 summarizes the results after each each pass of IPID with TTL splitting on the input set. Note that the drop in the number of IP addresses after IPid testing and validation is primarily due to discarding pairs, not lack of responses. Computing the number of IP addresses lost to packet loss is difficult, as IP addresses are in multiple pairs, but, in general, the observed packet loss rate was similar to the loss rate observed for the UDP algorithm. After IPid splitting, the average duplication was 2.28, although some IP addresses appeared in more than a hundred groups.

Not all pairs were able to be validated by IPid Validation. Of the 2,965,950 pairs to be validated, 108,005 (3.6%) pairs were still inconclusive after six sweeps performed. More sweeps could be done, but later sweeps yielded few conclusive tests, and most of the conclusive tests were failures.

Figure 3.10 shows the final accuracy values. Approximately 3.3% of implied pairs were not in the output. About a third of these pairs were excluded before IPid verification. IPid with TTL splitting had the best accuracy and coverage, but also had the highest cost.

3.6.3 IPid with Adjacency Splitting Results

IPid verification with adjacency splitting performed remarkably well. The 5,620,847 pairs from adjacency splitting took 670 minutes to test. Of these pairs, 190,783 passed validation. Figure 3.11 summarizes the results.

The number of implied pairs not in the output is 6.3%, higher than the 3.3% that IPid with TTL splitting obtained. The percentage of such implied pairs that failed validation is similar to IPid with TTL splitting. Therefore, the additional implied pairs not in the output are a result of errors in the splitting technique used.

Description	Pair Count	Pct.
Implied pairs	81,637	100.0%
Output pairs	55,363	67.8%
Not in output	26,274	32.2%
Not in adjacency output	6,918	8.5%
Failed validation	19,356	23.7%

Figure 3.12: Summary of rate limitation results.

Description	Pair Count
Input set	5,620,847
Pairs tested	2,908,033
After three runs	
Rejected	2,607,481
Accepted	84,190
Inconclusive	216,362
After eight runs	
Accepted	23,886
Rejected by 20 limit	14,272
Total accepted pairs	55,363

Figure 3.13: Summary of rate limitation behavior.

3.6.4 Rate Limit Results

Figure 3.13 shows the behavior of the rate limitation algorithm and Figure 3.12 summarizes the accuracy results. The decision point after three runs allows the elimination of obviously wrong pairs, greatly speeding up the last five runs. The 55,363 accepted pairs represent 29,183 groups. Note that the bulk of the false negatives shown in Figure 3.12 come from validation failure. The high percentage of implied pairs not in the output is likely a result of a large rate of false positives. Not removing IP addresses that appeared in twenty or more pairs would result in even more implied pairs not in the output, as some IP addresses with high packet loss rates were paired with hundreds of IP addresses.

Rate limitation performed the poorest of the four algorithms. It had a high false negative rate, poor coverage, and medium cost. This is partially because less than 60% of hosts had rate limitations, making coverage above 60% unlikely. Despite the fact that rate limitation is the poorest of the algorithms, it still found many pairs not found by any of the other three algorithms.

3.6.5 Splitting Comparison

There are three major goals of the splitting algorithms: reduce the set size, minimize the set overlap, and minimize the induced error. Figure 3.14 summarizes the two methods and the null method. Null splitting, provided for comparison, just outputs the input set. Adjacency splitting optimizes the largest set, but covers less than half of the pairs. TTL splitting falls between the two, reducing the largest set by a factor of ten while still keeping most of the pairs.

Pairs discarded by TTL splitting and adjacency splitting are much less likely to be valid pairs than the

Method	# Sets	Total Size	Pair Count	Largest Set	Coverage	Pkts Sent
Null	1	386,157	74,558,421,246	386,157	100.0%	0
TTL	160	379,833	3,676,963,861	34,954	84.7%	466,756
Adjacency	5,620,847	11,241,694	5,620,847	2	59.7%	0

Figure 3.14: Summary of results for each splitting method. Pair count is the total number of pairs left to test. Coverage is the percentage of pairs found by the combination of the methods that appeared in at least one of the sets.

pairs kept. As a result, the fraction of remaining pairs that are valid is much higher than the input set. The original set had 343,720 valid pairs, as measured by the union of the algorithms. That corresponds to one in 217,000 pairs being valid. For TTL splitting, that fraction went up to one valid pair in 12,600. For adjacency splitting, the fraction was even higher: one valid pair in 27.4.

3.7 Conclusions

The IPid technique is clearly the most successful of the three techniques, finding the largest number of sets and pairs, removing the most IP addresses, and having the smallest number of false positives. However, it required many more packets than any other technique. The UDP technique scales best, growing linearly with the input size. It is also the easiest to implement and has a low false positive rate (after private addresses and common invalid responses are removed). The rate limit technique required the most number of packets and found the least number of sets. Moreover, it had the largest estimated percentage of false positives.

Some combination of all three techniques are necessary for optimal coverage, as no technique had better than 70% coverage. As each technique depends on behavior not specified by any standard, hosts are unlikely to be responsive to all three techniques. The coverage percentages show that there are a noticeable fraction of hosts responsive to only one of the techniques.

IP alias resolution is necessary to obtain a good measure of a topology using traceroutes. Not employing any IP alias resolution algorithm could result in overstating the size of the network by 40% or more. UDP techniques, while simple to use, only reduce the error from IP aliasing by approximately a factor of two. If only one technique is used, IPid technique is clearly the one to select, as it finds 87% of IP aliasing. Using all three techniques would get the best results, but is expensive on large data sets. On medium and large networks, using IPid and UDP techniques will find most IP aliasing at a reasonable cost. For smaller networks, such as a corporate network or the network of just one ISP, combining the results of all three will result in the best topology while not being prohibitively expensive.

Chapter 4

Estimating the Reverse Routing Map

Good routing is central to the stable and efficient operation of the Internet. Understanding and improving routing requires the ability to measure the current routing behavior. Routing can be viewed as a distributed algorithm in which each router makes local decisions that combine to define the path between any two hosts on the Internet. Generally, routing is difficult to understand because its distributed nature makes it difficult to measure arbitrary routing decisions. Most existing routing-measurement systems are able to measure only one level of routing or examine point issues, such as BGP convergence.

The routing decisions of the Internet can be viewed as a matrix with a row for every router and a column for every CIDR block. Each entry in this routing matrix contains one or more IP addresses to which that router will forward a packet with that destination (the next hop). Routers make decisions with near independence for each CIDR block. Even when the decisions are not completely independent, such as when adaptive routing is employed, the correlation between CIDR blocks is generally low.

One way to measure routing decisions is to use the techniques in Chapter 2 gather outbound paths from a single measurement host to many destinations. Each path provides some of the entries in the column corresponding to that destination. When the routing entries given by multiple paths are combined, the set of routing decisions known is diverse. In terms of the routing matrix, this gives scattered entries with little discernible pattern, making analysis difficult. Using multiple measurement hosts increases the number of entries known, but does not make the set any more coherent. In contrast, each inbound path to a particular measurement host provides a set of routing entries all in one column. Combining many inbound paths gives information only within that column, which yields a simpler set to analyze.

Unfortunately, measuring inbound paths is much more difficult than measuring outbound paths. Several methods do exist, but they do not yield many paths. This chapter describes a method to determine the routing decisions made by a large set of Internet routers for a particular destination. This set of decisions can be considered to be a directed graph with a node for each host. The measurement host is the "root" of the reverse routing map. An arc from node A to node B means that a packet routed through A towards the root will be forwarded to node B by node A. If there is exactly one path from each node to the root, the directed graph is called a "tree". This directed graph is known as the "reverse routing map" for the measurement host. Given a reverse routing map, paths from remote hosts to the root can be determined by finding a path within the map. Given a set of inbound paths, the corresponding reverse routing map is the union of the paths.

This chapter presents a novel method to estimate the reverse routing map for a measurement host using the TTL field. While the new method is not as accurate as previous techniques, it discovers approximately fifty times as many routing decisions as previous techniques. Combining the new method with existing techniques produces a good estimate of the decisions made by the routing algorithms running on the Internet.

The reverse routing map from the new method is compared against previous methods in terms of accuracy, precision, and coverage.

Section 4.1 explains known techniques to estimate the reverse routing map. Section 4.2 gives the new method. Section 4.3 analyzes the output of the new method, both by itself and combined with previous techniques. Section 4.4 presents conclusions.

4.1 Previous Work

There are two previously-known ways to determine a set of paths towards a location: traceroutes from remote locations and loose source routing. Both of these techniques provide limited data, although the data they do provide is highly accurate.

4.1.1 Traceroute Servers

One way to get inbound traceroutes is to do traceroutes from remote locations. This requires some access to the hosts in order to do the traceroutes. Gaining full access to a large number of hosts with diverse connectivity is difficult and expensive. Fortunately, full access in not necessary. Because traceroutes from remote locations are also helpful in diagnosing problems, some people and organizations operate servers to perform such traceroutes as a public service. Such servers are known as traceroute servers[51]. Traceroute servers can provide a variety of functions in addition to basic traceroutes, such as BGP table entries and ping, but only traceroutes are necessary for the purposes of this chapter. Traceroute servers are accessible using HTTP[38], but the query format varies between servers. Thus, collecting a large set of traceroutes requires dealing with these differences. Traceroute servers can be used to gather traceroutes from servers around the world at little or no monetary expense and with minimal time investment.

The set of traceroutes gained from such servers is limited to the available traceroute servers. Unfortunately, many traceroutes are necessary to build a reverse routing map of any reasonable size. Presuming each traceroute is twenty hops long, the first traceroute adds twenty nodes to the reverse routing map. However, later traceroutes add fewer nodes to the reverse routing map, as some hops coincide with previous traceroutes. Presume that each new traceroute provides six additional nodes on average (this value is from results in Section 4.3.2). To build a reverse routing map that covers 50,000 nodes, more than 8,000 traceroutes are necessary. There are approximately 700 servers listed on http://www.traceroute.org[51]. Even without removing stale links, broken configurations, and those servers that do not provide traceroute functionality, this is well short of even the 8,000 under-estimate of the number of traceroutes necessary. The alternative of finding or placing 8,000 hosts around the world is not tractable by any but a handful of entities.

Remote traceroutes produce an accurate, if limited, reverse routing map. This is useful for two reasons. Firstly, the more hosts that use a link to access the root of the tree, the more likely it is that at least one of the remote traceroutes traverses that link. Thus, the parts of the Internet used by many hosts to access the root of the reverse routing map are likely to be contained within the reverse routing map. Secondly, remote traceroutes provide a baseline against which less-accurate techniques can be compared.

4.1.2 Loose Source Routing

Loose source routing (LSRR)[74] is an IP option that allows a host to specify way-points through which a packet should travel before reaching its final destination. LSRR provides an alternative way to collect remote traceroutes. Consider an IP packet sent by a measurement host to itself, but loose source routed through a remote host. The path of such a packet will first travel towards the remote host. After reaching the remote host, the rest of the path will be the inbound path from the remote host. By sending a series of such packets with low TTLs in a similar manner to traceroute[46], the combination of the outbound and inbound path can be measured. Removing the outbound path yields the inbound path from the remote host specified as the way-point.

The problem with using LSRR is that it is not widely available. LSRR can be used for nefarious purposes. Specifically, LSRR can be used to circumvent improperly configured firewalls and other security structures. This means that LSRR is often disabled within routers. As long as this is not universal, it does not, by itself, cause major problems, as LSRR is designed to only require cooperation of the way-points. In this case, only the single remote host though which the packet is routed would have to support LSRR. Each router that supported it would effectively act as a remote traceroute server, and as few as 8,000 such traceroutes may be needed, using the estimate from above. However, some ISPs drop any IP packet that contains an LSRR option. This means that any packet traversing some major ISPs, such as UUNet, cannot include the LSRR option. As a result, LSRR is unable to produce any paths that traverse such ISPs, which greatly limits the coverage. As the trend on the Internet today is to increasingly block LSRR, the effectiveness of this technique is expected to continue to decrease. Because LSRR is of limited use and expected to become even more limited in the future, it is not utilized herein.

4.2 TTL Method

The basic idea of the new method is to apply the investigation methods of Sherlock Holmes: “when you have eliminated the impossible, whatever remains, however improbable, must be the truth”. Given the complete network topology, the options (the possibilities) a particular host has for the next hop towards a destination are the adjacent hosts within that topology. The question is how to eliminate the impossible.

The IP header includes an eight-bit field called the time-to-live (TTL) value[74]. The TTL is decremented by one for each router through which a packet travels. If the initial and final values of the TTL are known, the difference between those two values is the length of the path from the host sending the packet to the host receiving it. For two hosts A and B which are adjacent in the topology, a directed arc from host A to host B can be in the reverse routing map only if the length of the path from host A to the measurement host is one more than the length of the path from host B to the measurement host. Otherwise, it is impossible for the arc to be in the reverse routing map. Eliminating the impossible arcs leaves the true arcs and perhaps a few false ones. However, this elimination is possible only if the initial and final TTL values are known for packets from nodes in the topology to the root of the reverse routing map.

4.2.1 Topology Measurement

This algorithm requires knowing the network topology. There are several projects to measure the topology of the Internet[18, 59, 92]. This chapter uses data from the Internet Mapping Project (IMP)[18], collected using the techniques from Chapter 2. One of IMP’s daily scans was run concurrently with the measurements described herein to collect the topological data. UDP and IPid (with TTL splitting), described in Chapter 3, were used for IP alias resolution.

4.2.2 Reverse Routing Map Construction

Once the topology is known, the initial and final TTL values must be determined for packets from the nodes of the topology to the desired root. Presume that a node emits a packet towards the root. When the packet reaches the root, the final value of the TTL can be simply extracted from the received packet. The initial

Algorithm 4.1: BestTTL array algorithm

```
1. For each final TTL value F from 1 to 255
    (a) For each outbound distance D from 0 to 40
    (b) Best = 900, BestLoc = -1
    (c) For each common initial TTL value I
        i. Delta = Abs(F + D - I)
        ii. If (Delta $<$ Best and F $\geq$ I)
            A. Best = Delta
            B. BestLoc = I
    (d) BestTTL[F][D] = BestLoc
```

Algorithm 4.2: Directed graph creation algorithm

```
1. For each edge (A,B) in the topology
    (a) Let FA and FB be the final TTL values
    (b) Let DA and DB be the outbound distances
    (c) IA = BestTTL[FA][DA]
    (d) IB = BestTTL[FB][DB]
    (e) Delta = (IA - FA) - (IB - FB)
    (f) If Delta = 1, output A $\rightarrow$ B
    (g) If Delta = -1, output B $\rightarrow$ A
```

TTL value, however, cannot be determined by direct measurement except via SNMP, which is intrusive and rarely configured to allow queries from arbitrary hosts, only a small set of different initial TTL values are used in practice. The majority of Internet routers have an initial TTL for ICMP echo replies of 255. Based on SNMP measurements[11] and observed behaviors, other common values are 16, 30, 50, 60, 64, 96, 100, 128, 150, 160, 180, 192, 200, 220, and 224. Given a host some known distance away from the measurement host (determined by the paths from the topology measurement) and a final TTL value, the most likely initial TTL value is the one closest to the sum of the path length and final TTL value. This method yields a likely initial TTL value. In such cases where two TTL values are both closest, select the smaller of the two. This method effectively assumes that the inbound path length is approximately equal to the outbound path length.

The algorithm realizing the TTL method consists of two phases. The first phase creates the BestTTL array, which specifies the likely initial TTL value for a host from the final TTL value for a packet from that host and the length of the path from the measurement host to the host. The second phase considers all edges in the topology to create the reverse routing map. Algorithm 4.1 is the algorithm used to create the BestTTL array. This algorithm implements the method given in the previous paragraph. Algorithm 4.2 is the algorithm used to create the reverse routing map, given a set of edges, host distances, and ending TTL values. The outbound distance of a host is defined as the minimum distance the host is observed along any of the outbound paths. In the rare case that multiple final TTLs are observed from a host, the inner loop of Figure 4.2 is run for each pair of TTLs observed from A and B.

Some hosts did not respond to an ICMP echo request. This could occur for a couple reasons, such as the host's IP address not being globally routed or the host is configured to not respond to ICMP echo requests. In such cases, the TTL observed within the traceroute is used. Resorting to this TTL is less desirable, since some hosts treat ICMP time exceeded packets differently, using either different routing rules or a different initial TTL (for example, 254 instead of 255). Some hosts are non-responsive hops from traceroutes. Edges with at least one non-responsive hop as an endpoint were always dropped.

4.3 Results

Collecting traceroute server output, outbound traceroutes to measure topology, and direct TTL values using ICMP took three hours, limited by the time to do the traceroutes to measure topology. IP alias resolution took an additional three hours. Although route changes during the initial three hours will affect the map, route changes during IP alias resolution do not affect the reverse routing map. The reverse routing map measures, to some extent, a time-average of routing over the first three hours.

The measured topology had, before IP alias resolution, 189,100 edges and 154,900 nodes. IP alias resolution found 17,460 hosts that appeared as multiple IP addresses representing 42,389 IP addresses, although not all of those IP addresses appeared in the topology. After combining IP addresses belonging to the same host, the topology contained 154,947 edges on 136,339 nodes. Of these 136,339 nodes, direct TTL collection using ICMP obtained TTLs for 88,154 of them. For the other nodes, the TTL data contained in the outbound traceroute data was used.

Traceroute servers were queried, both to measure the amount of data from each server and to have a set of accurate data against which the TTL results could be compared. 318 traceroute servers were queried. Of these, paths were obtained from 136 of them. Paths could not be collected from traceroute server outputs for a variety of reasons. 174 of the outputs were rejected because they contained no traceroute. Two outputs were rejected because they contained traceroutes specified in hostnames instead of IP addresses. Two outputs had valid traceroutes, but were in HTML tables that were not understood by the parsing programs. Four of the outputs had incomplete traceroutes that were too short to yield any useful information and were thus discarded. Of the traceroute servers that failed to return a traceroute, some failed because the HTTP request for the traceroute was not in the form the traceroute server wanted. 65 of the traceroute servers from which traceroutes were not obtained requested a form to be submitted using POST. These traceroute servers may have yielded paths if POST was used instead of GET. 23 of the traceroute servers rejected their query because no referrer was given (they rejected it with a message stating that no shortcuts are allowed). With further work, more traceroute server data could have been collected. However, even twice as much additional data would not have changed the qualitative results.

Three different reverse routing maps were created: one derived purely from the traceroute servers, one derived purely from the TTL method, and one derived from both pieces of data. These will be compared against the real tree in three ways:

1. accuracy of the arcs excluded,
2. coverage of the map, and
3. how well the reverse routing map resembles a tree.

The accuracy of the arcs excluded is relevant for reverse routing maps derived from the TTL method. The TTL method discards arcs that it concludes are not in the reverse routing map. How often does it improperly discard an arc that should be in the reverse routing map? This will be evaluated by comparing the arcs from the traceroute servers, which are in the true reverse routing map, against the results from the TTL method.

The coverage of a reverse routing map is the fraction of the Internet for which it provides complete paths to the root. This can be measured both in terms of the nodes of the topology and access routers for edge networks. One of the stated reasons for using the TTL method was the traceroute server data had poor coverage. How much has the coverage increased by augmenting the traceroute server method with the TTL method?

Coverage only measures one part of precision; the other side of precision is how many extra arcs the reverse routing map contains. The reverse routing map should be nearly a tree, but imprecision causes the reverse routing map to contain extra arcs. By comparing some properties of a tree with the properties of the reverse routing map, this resemblance can be evaluated. The first property considered is that each node in a tree that is not the root has only one out-bound arc. The second property considered is that there is only one path from each node to the root. For each node except the root, its out-degree and the number of paths from that node to the root should be one. How much do the true distributions differ from this?

4.3.1 Accuracy

Traceroute is generally considered an accurate method to determine the series of IP addresses along a path. Thus, the traceroute server data should be accurate inbound paths from the server to the root. The accuracy of the TTL method, therefore, can, and will, be measured by comparing its output with the traceroute server data.

The traceroute servers yielded 136 paths representing a total of 833 arcs. The input topology contained only 396 of these arcs. Of these, 19 (4.8%) were dropped by the TTL method. Thus, the TTL method is approximately 95% accurate in maintaining correct arcs. In contrast, only 35.5% of all arcs in the input topology were maintained. The bulk of the inaccuracy results from inaccuracies within the measured topology: 437 of the arcs from the traceroute server data (52.5%) were missing from the measured topology. Because the topology is considered an input to the TTL method, arcs missing in the input that should have been in the output are not counted against the TTL method. Such a large fraction of arcs missing from the input topology is, at first, alarming. However, these numbers overstate the incompleteness of the topology.

One explanation is that the topology used was based on traceroutes from a single source. The observant reader may wonder why only 48% of the arcs were in the topology when Barford, et al.[3] observed that 60% of the arcs are found using traceroutes from a single source. Clearly, there are other effects contributing to missing arcs besides the fact that only one source was used. Since the traceroute server could be any host on the Internet, arcs missing from the topology could be the result of firewalls, the outbound paths not containing the arcs, or simply not deeply exploring edge networks (i.e., not tracerouting to every possible IP address). The second case is what Barford, et al. attempted to measure.

To distinguish between output paths not containing the arcs and insufficient exploration, considered whether the endpoints of the missing arcs are within the topology. If the arc is missing as a result of not using multiple sources, then the far endpoint should be in the topology. Insufficient exploration will result in arcs whose far end-point of the arc is not within the topology. Table 4.1 gives counts of missing arcs with respect to whether or not their endpoints are within or without the topology. Based on Table 4.1, 291 of the missing arcs were beyond the scope of the topology and 146 were the result of missing edges in the topology. Thus, as a result of using a single-source topology, about 17% of the correct arcs were not in the topology. 35% of the correct edges were not in the topology because they were either beyond a firewall or they were in an unexplored edge network.

Far node	Near node	Arc count
Within topology	Within topology	64
Within topology	Without topology	82
Without topology	Within topology	131
Without topology	Without topology	160

Table 4.1: Distribution of arcs in the traceroute reverse routing map but missing in the topology, separated by whether or not the endpoints of the arcs are in the topology.

4.3.2 Coverage

Coverage is the completeness of the information provided. In this case, coverage is the number of IP addresses for which there is a complete path from that IP address to the root in the reverse routing map. This somewhat understates the true amount of information contained in the reverse routing map, as it ignores partial paths that do not connect to the root. This definition of coverage is designed to measure the reverse routing map's ability to provide inbound paths.

Coverage will be measured by analyzing the reverse routing map that results from taking both the traceroute server data and the output of the TTL method. Coverage, unlike accuracy, does not require a source of accurate data to compare against. Thus, there is no need to withhold the traceroute server data.

Coverage of nodes is straight-forward to calculate. Of the 136,625 nodes in the topology or the traceroute server output, 52,024 nodes have paths to the root in the output reverse routing map. Thus, the node coverage is 38.2%. If only the traceroute server data is used, 807 nodes in the reverse routing map have paths back to the root, an average of 5.93 per traceroute server used (hence the figure of six hops per traceroute used in Section 4.1.1). The node coverage of the traceroute server data is 0.6%.

Coverage of nodes is arguably a poor measure of coverage of the Internet, as nodes are generally routers which neither generate nor receive much traffic. An alternative coverage is the fraction of routed IP addresses for which a path exists within the reverse routing map from that network to the root. The difficulty is defining what network an IP address is on. One way to do this is to utilize the traceroutes from which the topology is derived. Each path in the topology measurement is based on targeting a random node in a CIDR block. The last hop on each path represents the likely access point for that CIDR block, at least from the viewpoint of the measurement host. This provides a (mostly accurate) way to assign CIDR blocks to nodes in the topology. A local BGP feed includes announcements representing 1,275 million IP addresses. The nodes that have paths to the root in the combined reverse routing map represent 570 million IP addresses, yielding CIDR coverage of 48%. Using only the traceroute data, 14.5 million IP addresses are covered, a CIDR coverage of 0.8%. This analysis weights CIDR blocks by the number of IP addresses which they contain.

Although neither the node nor CIDR coverage of the combined reverse routing map exceed 50%, the TTL method has greatly expanded on the output from traceroute servers. By augmenting the traceroute server data with the TTL method, the coverage has increased by a factor of 64 in terms of the number of nodes and 39 in terms of the number of IP addresses represented.

4.3.3 Tree Resemblance

As defined, perfect coverage and accuracy could be obtained by discarding no edges. Of course, the resulting reverse routing map would be useless. The missing requirement is that the reverse routing map should be resemble a tree. Although the actual reverse routing map is not a true tree due to vagaries of Internet routing,

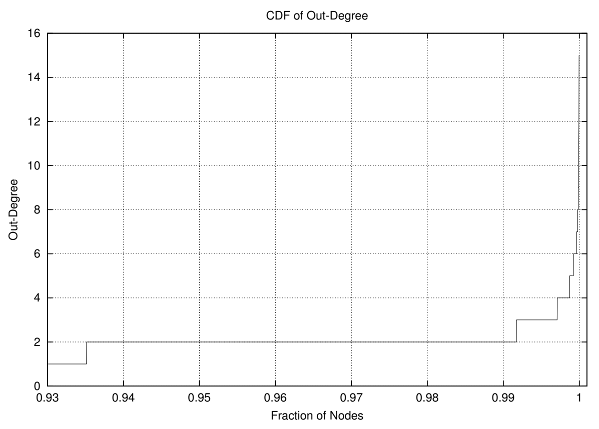


Figure 4.2: Distribution of out-degree of nodes in the output reverse routing map of the full method. Note that the x-axis begins at 0.93

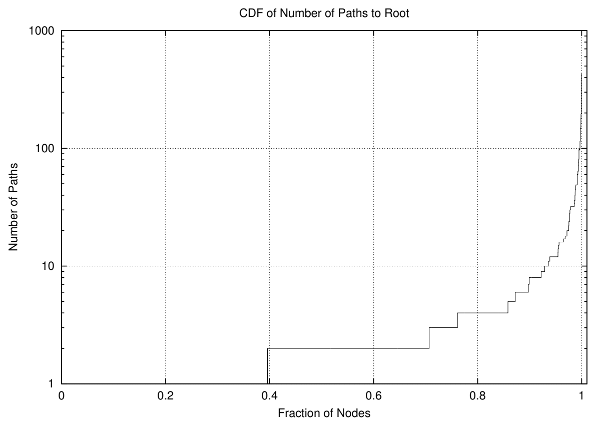


Figure 4.3: Semi-log plot of distribution of number of paths from nodes in the output reverse routing map to its root.

it is close to one. Thus, the measured reverse routing map should be nearly a tree as well. As it is difficult to quantify how closely a graph resembles a tree, three different methods will be employed: degree distribution, path count, and path spread, the last to be defined shortly.

Perhaps the most obvious method to compare the reverse routing map to a tree is to examine the distribution of the out-degrees of the nodes in the output reverse routing map. If the reverse routing map were a tree, the out-degree of all nodes but the root would be one. Figure 4.2 gives the distribution of out-degrees and, indeed, shows that 93.5% of the nodes (48,581) have an out-degree of one. A few of the nodes do have large out-degrees: the top 0.1% of the nodes have out-degrees five or greater and three nodes have out-degrees above ten. However, for most nodes, the output reverse routing map gives a single next hop.

An alternative measure of tree resemblance is the number of different paths in the output reverse routing map from each node to the root. In a tree, there is one path from each node to the root. Figure 4.3 shows that, for most nodes, the output reverse routing map has no more than two paths to the root. 39.6% of the nodes have a single path to the root and 31.1% have only two. Less than 1% of the nodes have more than 50 paths, although a few nodes have more than 400 paths.

Purely counting paths does not consider how different the paths truly are. To correct this, look at the average number of different IP addresses each distance from a node, which will be referred to as the "path

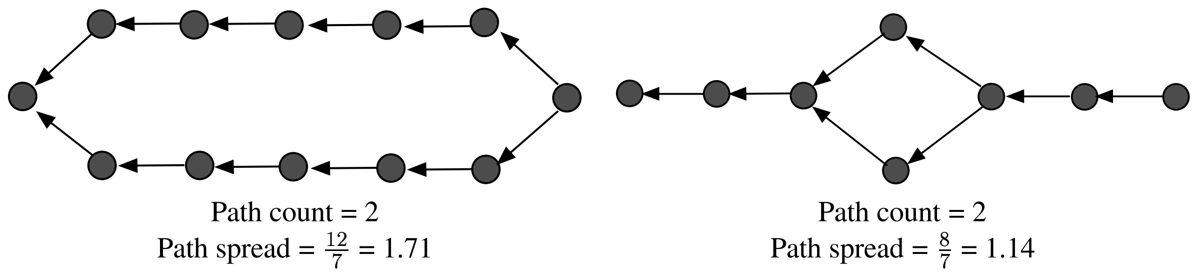

Path count = 2
Path spread = $\frac{12}{7}$ = 1.71
Path count = 2
Path spread = $\frac{8}{7}$ = 1.14

Figure 4.4: Difference between path spread and path count. Path counts and path spreads are for the right-most node of the graph.

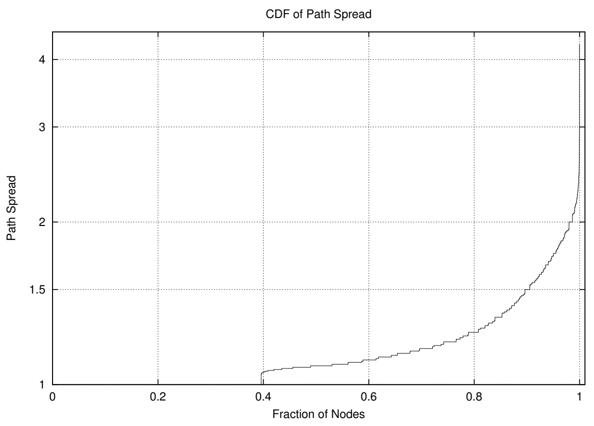


Figure 4.5: Distribution of path spread in output reverse routing map plotted on a semi-log plot.

spread" of a node. The path spread measures how many different nodes the output reverse routing map gives as possible hops along the path from a node to the root. In particular, the path spread of a particular node A is the number of nodes on paths from A to the root of the output reverse routing map divided by the minimum distance indicated by the output reverse routing map. For a pure tree, the path spread should be one for each node, indicating that there is only one path from each node to the root.

Path spread gives a better notion of precision than simply counting paths. To see this, consider the cases shown in Figure 4.4. In both the left and right graph, the number of paths from the right-most node to the root (on the left) is two. However, the two paths in the left graph differ in all hops except the first and last. In such a case, the path spread is nearly two. In the right case, the two paths are similar, differing in only one hop. In such a case, the path spread is nearly one. Path spread correctly identifies the right case as better, while path count does not distinguish between the two.

Figure 4.5 shows the distribution of path spread. 39.6% of the nodes have a path spread of exactly one, corresponding to the 39.6% of the nodes that have exactly one path to the root. Beyond that, path spread makes it clear that although many nodes have multiple paths to the root, most of those paths are similar. A majority of the nodes (89.6%) have path spreads below 1.5, meaning that at least half of the hops along the reverse path are known (only one possible host for that hop).

4.4 Conclusions

Previous techniques to determine the reverse routing map have poor coverage. Both LSRR and traceroute servers depend on gathering remote traceroutes from many different locations. To cover 50,000 nodes would require at least 8,000 remote traceroutes and still not be full coverage of the entire Internet. LSRR is blocked by many ISPs and blocking LSRR is increasing in popularity. There are around 700 traceroute servers, a factor of eleven below the minimum required amount. Thus, obtaining a reverse routing map with good coverage requires new techniques.

The TTL method detailed herein provides much better coverage than previous techniques. When combined with traceroute server data, the resulting reverse routing map covers 38.2% of the nodes, compared to 0.6% coverage using only the traceroute server data. This technique is not as accurate as previous techniques, but maintains 95% of the correct arcs while dropping 64.5% of all arcs.

Besides removing some arcs that it should not, the TTL method includes some arcs in the reverse routing map that should not be included. This manifests itself as multiple paths from nodes to the root. Although there are often multiple paths from nodes back to the root, they do not differ greatly. For 89.6% of the nodes, path spread shows that the various reverse paths offer only a single possibility for at least half of the hops.

Improving the input topology would improve the results. 28% of the arcs included in the traceroute server were not in the input topology. Using multiple sources for the topology measurement or scanning over multiple days would improve the topology, although that would sightly complicate TTL collection, as the majority of TTLs collected by topology measurements would either be for a different measurement host or old data.

One drawback of the TTL method is that it requires access to the root of the reverse routing map. Loose source routing has no such restriction. As some traceroute servers will only traceroute to the host requesting the web page, fewer traceroute servers would be useful if no access to the root was available. This would cause decreased coverage, but the traceroute server technique could still be used.

This chapter presented the first technique to estimate the reverse routing map for a measurement point that covers a significant portion of the Internet. The limited output from previous techniques can be used to augment the estimation provided by this new technique, producing a combined reverse routing map. The combined reverse routing map is a good estimate of the routing decisions towards a particular destination for a considerable fraction of the Internet.

Chapter 5

Tracing Anonymous Packets[1]

One of the major problems on the Internet is denial of service (DoS) attacks against hosts and networks. As compared to other attacks, these are attempts to limit access to a host or service instead of subverting the service itself. DoS attacks are simple to design and implement, and there is a plethora of readily available source code which will perform the task. The idea behind most DoS attack is to send a stream of packets towards a victim such that the victim is unable to serve its clients because it is too busy serving the 'clients' that the attacker is emulating.

There are two basic targets of DoS attacks: hosts and the network. SYN attacks[12] are an example of an attack against a host. In these attacks, a series of TCP SYN packets are sent to a host, which causes the host to create a lot of half-open TCP connections. When a real client wishes to connect, the host cannot find an open slot in its table for the client, resulting in the connection being denied. The basic problem with this attack is that clients and attackers are almost indistinguishable, except that clients respond to the TCP SYN/ACK packets the host returns, while attackers do not. This particular kind of attack has a known defense[12], although other host attacks may be more difficult to defend against.

The second target, the network, is a much more difficult problem. Here, the target is to overload a company's connection to its ISP with traffic. In particular, the attacker causes a large stream of data to flow towards the company. This results in overloading in the pipeline from a company's ISP to the company, causing packet loss. Since routers cannot distinguish between attacking packets and valid clients' packets, they drop them with equal probability. If the attacker can send packets quickly enough, the drop rate can rise enough that an insufficient number of clients' packets get through. This results in clients not being able to get reasonable service from any host through the loaded link. A common form of this type of attack is the smurf attack[13]. Distributed denial of service (DDoS)[14] attacks are generally also of this flavor.

The major advantage of DoS attacks, from the attacker's perspective, is that it is extremely difficult to determine the actual source of the attack. Since the attacker can put almost any packet on the local wire, the attacker creates packets whose source IP address is invalid and random. When the victim receives these packets, they are unable to determine the source. The current technique for tracing a packet stream back to the source requires cooperation of all the intervening ISPs. This is something that is difficult to obtain, since the victim is rarely a customer of all of the ISPs between it and the attacker. This technique will be discussed in more detail later.

This chapter presents a method to trace a steady stream of anonymous Internet packets back towards their source. The method does not rely on knowledge or cooperation from intervening ISPs along the path. In addition, tracing an attacking stream requires only a few minutes once the system is set up. The

[1]This chapter represents work with William Cheswick, originally published as "Tracing Anonymous Packets to Their Approximate Source" at 2000 USENIX Lisa[8].

results presented herein represent work started eight years ago and published five years ago. Although much later work has been done based on this work, the technique itself may no longer work. Its effectiveness is somewhat moot anyway, as most attackers now use distributed denial of service attacks, where the main technique described is not useful. The alternative techniques discussed, however, remain useful.

The basic technique is given in Section 5.1. Section 5.2 lists the assumptions made. Section 5.3 discusses various network loading procedures. The results are detailed in section 5.4. Alternative solutions are listed in section 5.5. The antisocial aspects of this approach are discussed in section 5.6. Section 5.7 covers some of the work since the original publication of this work. Section 5.8 presents the conclusions.

5.1 Basic Technique

The technique begins by creating a map of the routes from the victim to every network, using any known mapping technology[17, 41, 59]. Then, starting with the closest router, the technique applies a brief burst of load to each link attached to it using the UDP chargen service[75]. If the loaded link is a component of the path of the attacking stream, the induced load will perturb the attacking stream. Thus, if the stream is altered when a link is loaded, this link is probably along the path from the source host of the attack to the victim host. If the strength of the stream is unperturbed by the load, it is unlikely that the stream of attacking packets is utilizing that link, making it unnecessary to examine the networks "behind" that link.

The technique continues working back through the network router by router, pruning branches that do not perturb the attack, as the technique tries to narrow the attack source to one network, at which point more standard trace-back techniques can be employed by contacting the entity which controls that network.

Executing a trace effectively does require significant preparation in the way of data collection. The technique needs data collected from the network, as well as traceroutes from the victim to all possible networks. Due to asymmetric routes, näively, directional data must be collected and maintained by reverse traceroute servers or the techniques of Chapter 4 in order to have better data. One way to avoid this is to collect outbound paths and assume that the incoming paths are approximately the reverse of those paths. While this is not completely accurate, collecting the paths to all networks determines what links could be used on a path from a given network to the victim's network. Thus, this assumption does not cause as much inaccuracy as might otherwise occur.

Because the technique must be able to induce isolated load on specific network segments that are not in its purview, it is necessary to identify sources "willing to" (read: will) perform that task. It is known that ISPs quite regularly turn off the services that this technique exploits to induce these loads. Thus, the technique must identify cooperative hosts at the right places in the network map in order to produce the required load.

This element of the technique is worrisome, since it constitutes a brief denial-of-service attack on that network link. Hackers already employ bulk versions of this approach for denial-of-service attacks. This technique, on the other hand, carefully limits load to segments only long enough to rule them out as a possible component of the suspected path. The difference is analogous to that between a sword and a scalpel. In any case, the antisocial aspect of this technique is recognized, and the tool should be used rarely and only in appropriate situations. Possible customers include law enforcement, the military, ISPs, and perhaps companies policing their own private intranetworks.

Before attacks or victims are even known, a trusted host must develop and maintain a current database of networks and load generators. The current version of the tool executes the trace from the victim (targeted) network, but a sufficient complete map of the Internet might allow a neutral third party to run the detecting utility, which would allow flexibility in where to spread some of the bandwidth costs of the tool.

In either case, the tracing host emits packets that stimulate traffic flow through a desired router or link. A visual display of various statistics of the incoming packets on the victim's network helps determine if that link is used by the packets.

The trace-back is done manually. Though there are algorithms that might automate this process, requiring that a human manually apply the load to the link reduces the cost of programming errors. The programs are designed to supply the operator with information about the amount of load she is inflicting on networks, and she can chose to stop using networks that have already be heavily used.

If the induced load is sufficient to induce drops of incoming packets, it quickly and dramatically affects the attacking flow. The discomfort to ISPs and end users is brief enough that it is likely to escape notice. If the load does not induce loss, it may be necessary to run the load generators longer and seek more subtle effects on the workload. This technique appears to work better when the network is already heavily loaded, though one can imagine more subtle statistical effects that may be detectable when the Internet is relatively quiet. Attempts, however, to discover such effects have met with little success.

5.2 Assumptions

The presented traceback technique relies on several assumptions, but experience indicated they were often valid and that the technique would work. These assumptions, particularly the assumptions about hacking behavior, are probably no longer true.

5.2.1 Assumptions About the Internet

Because of the way the search is done, the technique requires that most routes over the Internet are symmetric. Asymmetric routes confuse mapping, trace-back, and loading. However, the proliferation of reverse traceroute servers, which has proven quite useful for network diagnosis and debugging, might also facilitate construction of at least a partial directional map of routes. Techniques from Chapter 4 to build an approximate reverse map would also make this assumption unnecessary.

It is also assumed that enough load can be applied to a particular Internet link to affect performance statistics, in particular loss, of the stream of attacking packets. Enough packet generators beyond the tested link must be available to load it, which can be challenging across infrastructure with fast links and slower downstream networks. The techniques for doing this will be discussed below, as well as experiments on how true this is.

5.2.2 Hacking Behavior

One of the largest assumptios is that the attack is from a single host. In addition, the attack must be at a consistent rate and continue for a reasonably long time. Denial-of-service attacks are more vexing if they are ongoing, and some attacks have lasted for weeks. The technique requires time to move equipment and programs into place, map routes, and perform the actual trace-back. This assumption was often true at the time of the original work. Today, this assumption is no longer valid, as most DoS attacks are distributed.

Because bizarre behavior can occur during the trace-back, the technique must make as effective use as possible of the clues gathered. For example, the operator might notice an attacking stream's statistics are perturbed by 1/3 rather than dropping off entirely. Such behavior would be consistent with two or three concurrent attacks from separate hosts (it also possible that the attacking stream is being load-balanced across three different links). Unfortunately, only one packet stream can be traced at a time, if that. Thus, being able to distinguish among the streams would be essential to be able to perform the trace. This may be possible using other clues in the packets. For example, the operator might be able to distinguish between a few attacking streams using the arriving TTL value, assuming packets within each stream are launched with the same TTL value, and with each stream from different hop distances away. In practice, DDoS attacks are believed to come from many hosts. It is likely to be difficult, if not impossible, to distinguish many attacking streams.

It is assumed that the attacker does not know that their packets may be traced. An effective hacker attacks from co-opted hosts and never returns to the attacking host. She hides her trail through a thread of login sessions across many hosts and networks before attacking the target. Of course, TCP-based attacks require bidirectional packet flow between the attacker and victim, making this technique unnecessary. The denial-of-service attack targetted with this tool is a one-way packet flow, which does not rely on interactive login sessions.

In addition, it is assumed that there is something forensically interesting at the source of the attack. The effort of running the traceback tool may not be justified if the result is just disabling one attacking host or convincing one community of computers to enforce valid source addresses. The technique may be able to catch someone who was not cautious because he did not expect his packets to be traced (the difficulty of the tracing task renders this a common assumption of hackers).

Lastly, the attacker must not be familiar with the techniques listed here. These techniques can be easily thwarted in several ways, including modifying the attacking program to vary the source of the attack, altering the frequency of the packets randomly, and attacking from many different sources.

5.3 Network Load: No Gain, No Pain

Once the path from each network on the Internet to the victim has been determined, either by measuring forward paths using the techniques of Chapter 2 and assuming symmetry or using reverse techniques described in Chapter 4, the trace-back is done by walking backwards through the resulting directed graph. The technique loads a link, hopefully causing enough packet loss to noticeably decrease the rate of attacking packets. If a significant drop occurs, there is good reason to believe that the tested link is on the path from the attacker to the victim. Otherwise, either the link is not on the path or the load, or 'pain', applied to that link was insufficient to incur packet loss. Note that since most links are full duplex, the link must be loaded in the direction towards the victim.

This traceback requires making a high capacity link busy for a short period of time, usually a second or two. It is difficult to generate a flow of packets from a single host or network that will do this. It would have to come from a fast host on a fast, unloaded link. Ideally, the technique would utilize some leverage, some "gain," on packets it emits. That is, if the technique sends out a flow of size x, the resulting load across link of interest should be of size kx, were k, the gain, is large. Since the rate that packets are sent is limited by several factors, such as the initiating network and the initiating host, low gains mean that the resulting load will not be sufficient to alter the packet rate.

In the early days of ARPANET, mailing lists were full of discussions of situations of the desired type, talking about routing loops and exponential problems such as broadcast storms. These discussions resulting in an Internet that has been carefully engineered to avoid allowing a single emitted packet to cause a lot of resulting packets. This is certainly a good thing for the Internet, because if it was possible to get a huge gain, then hackers would be exploiting the design flaw. In fact, the smurf attack[13] exploits one flaw that was not fixed.

To produce the load, the technique sends a series of messages, such as ICMP echo request (ping) packets[73], from the victim's network out to some distant networks whose return paths are expected to include the link to be loaded. However, an ICMP echo request packet only gets one byte in return for every byte sent, which is a gain of only 1. In addition, the return packets traverse the entire path back to the victim, which loads the entire set of links from the assistant network to the victim, which obscures the data when trying to determine, say, the third link out. Sending ICMP echo requests from a separate network dedicated to this service is also problematic, since the nature of Internet routing means that it is hard to assure that their return path traverses the link being tested.

The technique solves the obfuscation problem by sending spoofed packets. When testing a particular link, probe packets are sent to the router on the far end of the link, using as a return address the router on the near end of the link. The near router indignantly discards the unsolicited replies (if using TCP, it actually may reset; for UDP, it may reply with a ICMP Port Unreachable). Thus, the return packets traverse only the link of interest.

5.3.1 More Gain

There are additional problems with using ICMP echo requests. Many routers make special efforts to put rate limits on handling of ICMP echo requests, since they are sometimes used in attacks. More importantly, a gain of 1 makes loading a backbone link difficult, if not impossible. Thus, a different service to supply the load must be found.

The most obvious choice of service to employ is the often-overlooked tiny service TCP character generator (chargen)[75]. This service generates continuous data to anyone who connects to it, exactly what is desired. The rate of data flow is limited in general by the rate that the data is acknowledged by the client host. At the cost of a few TCP ACKs from the victim, the technique can coax a steady stream of data out of a site supporting this service. Several of these streams routed over the target link will generate substantial load. In theory, TCP ACKs could be used to pulse all the transmitters, providing a fine burst of load by ACK-ing several open chargen sockets simultaneously.

As TCP chargen was implemented as part of this technique, two major problems arose: the TCP processing on the local host slowed this chargen stream down more than desired, and, more importantly, the chargen stream must traverse the path all the way back to the sender, unless much more complex techniques, such as TCP sequence guessing, are used. This second problem can be more easily circumvented by using UDP chargen instead of TCP, and spoofing the packets, but this method provides little gain, usually around 102 bytes back for each sent packet of 40 bytes[2], a gain of only 2.55. The chargen RFC specifies that the return packet should have between 0 and 512 bytes of data[75] (not counting the 28 bytes for the IP and UDP headers[74, 72]). Some Windows NT 4.0 hosts, however, violate this standard and return up to 6,000 bytes in response to a single packet, a gain of 150.

A spoofed ICMP echo request to a broadcast address can yield gain as well. By locating networks ‘beyond’ the link to send broadcast ICMP echo requests to, the technique gets a gain of one for each host on that network which responds. Unfortunately, many routers process broadcast ICMP echo requests in such a way that only the router on the network returns a packet. This is, of course, fortunate for potential victims of broadcast ICMP echo request attacks, and is, in fact, recommended for that reason[13]. However, for the purposes of this technique, it limits the usefulness of broadcast ICMP echo requests.

Such routers do let other broadcast traffic through, however, and experiments showed that gains in excess of 200 can be obtained quite often using broadcast UDP chargen packets. Surprisingly, many networks within Lucent still respond to broadcast address 0 instead of 255. Therefore, both had to be checked to determine the correct one for each network. Figure 5.1 shows a distribution of networks and their gain for

[2]Sent packets consist of a 20 byte IP header, a 8 byte UDP header, and 12 bytes of data that gives information about the source of the packets, since they are spoofed.

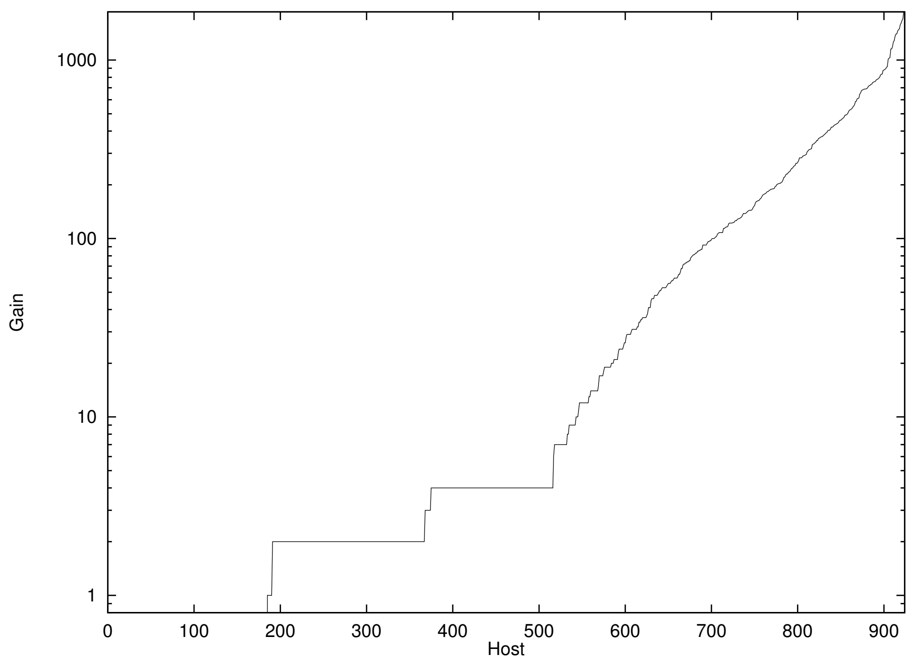


Figure 5.1: Distribution of gains seen using the broadcast address for Lucent's intranet. The network that generated a gain 43,509 and is excluded from this graph.

Lucent's intranet. Note that the networks with a gain of less than 1 have a gain of 0, which means that they did not respond to broadcast UDP chargen at all.

The goal of the load is to load one line or, maybe, one router. Loading the entire path back is certainly undesirable. This is prevented in two different ways: First, as mentioned above, the return address of the UDP chargen packets is spoofed to be the address of the router on the victim's side of the link. Second, the technique utilizes multiple UDP chargen hosts. To test a link, the technique selects networks that reside behind the link, as seen from the victim. In particular, the technique selects networks that have hosts that respond to UDP chargen broadcast packets and whose reverse paths are believed to traverse the link to be tested. This strategy focuses the load on the link under examination; the load packets travel to the far router over mostly disjoint paths, affecting each other little. The returning packet paths then join at the far end of the link, traverse that link, and end. The load is limited by the lines the load must traverse, the speed of the networks where the load is being generated, and the ability of the measurement host to generate packets to emit the UDP chargen packets.

Experimentally, the average gain observed in Lucent is around 133.8, ignoring one network that has a gain of several tens of thousands (a factor a 25 above any other), due to oddities in its configuration (discussed below). A host can generate 2,500 40-byte packets per second without a problem, a total of 800,000 bits per second. To flood a 10Mbps Ethernet only requires a gain of 12.5. Table 5.3 shows the necessary gains to load a variety of line types. In order to flood a backbone link, such as an OC-48 or OC-192, the gain must exceed 3,000, which is larger than all but one of the observed gains. However, when loading links, the link is already partially loaded by the rest of the traffic that is traversing those links. Thus, the actual amount of traffic required to start packet loss is less than the number in Table 5.3. Also, because this technique is no longer sensitive to the location from which the load-inducing packets are sent, the rate of outbound packets could be increased greatly by using multiple computers that connect to the Internet over different links.

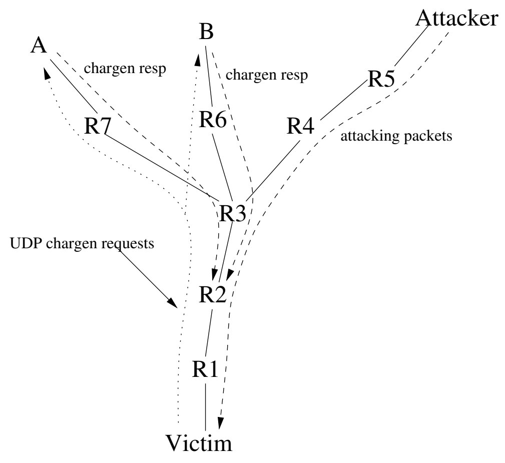


Figure 5.2: Example of trace-back step. Packets are sent to A and B, spoofed from R2, in order to initiate packet flows towards the victim. This causes increases congestion along the R3-R2 link, which, if sufficient, will induce packet loss.

Line Type	Gain Required
10Mbps Ethernet	12.5
100Mbps Ethernet	125
1Gbps Ethernet	1250
T1	1.9
T3	56
OC-12	777
OC-48	3,110
OC-192	12,441

Table 5.3: Required gains to load a variety of line types, assuming 800 kbps of emitted packets and no packet loss.

From the numbers of Table 5.3, it is clear that the hardest problem is usually tracing through the wide-area network. Besides requiring a large gain, testing links on the wide-area networks is difficult because, by the time the chargen packets get to the link under examination, congestion-induced loss through previous routers has diluted the stream. The technique overcomes this difficulty by using several chargen generators whose paths do not coincide until the target link. This will be referred to as 'focusing' the load. Focusing can also alleviate the problem of testing lines that are nearer and faster than lines behind them. However, backbone links are already loaded, especially during peak times. Thus, it may not require adding as much traffic to the link to noticeably increase packet loss.

Thus far, only techniques for loading a link have been considered; another possibility is to load the router. Diverting packet flow by sending a message directly to a router is quite difficult, as Internet backbone routers ignore various ICMP messages to redirect or stifle packet flow. Most methods to load a router have to tackle its system configuration to limit return data flow. Router designs also typically have almost all forwarding handled by a simple host that delegates difficult tasks to a higher layer. Less legitimate options, such as hijacking BGP sessions or breaking into the router itself, are too malicious to be seriously considered.

There are other possibilities for ways to slow routers, however. One option is to ping-flood the router, i.e., send it ICMP echo requests as fast as possible. A similar alternative is to send the router a flood of packets whose Time to Live (TTL) value expires at the desired hop along the path, or to transmit a stream of UDP packets to high ports to stimulate responding UDP port unreachables. Since most routers appear to rate-limit UDP port unreachable messages, this idea was abandoned before testing it extensively. Other methods do not seem to have a major effect.

Another idea is to attempt to upset the router's routing table by spewing packets at the router. That is, find some sort of packet it responds regularly to (TTL exceeded, echo request) and send it a bunch of packets with random return addresses. Coping with the packets may require enough attention to unsettle the route table cache. In order to combat the incoming stream, it may be useful to pick a handful of sources and cycle through them. This approach has not shown much promise when used within Lucent, perhaps because most Lucent routers use only a single default route to avoid making forwarding cache state a resource issue.

5.4 Results

To test the technique, logins were obtained on various hosts throughout Lucent's intranet. A non-privileged program named *sendudp* was written and run to generate a stream of packets back to a nonexistent host on the same network as the tracing host. In most cases, the technique was able to trace the packets back to the "attacking" building. In many cases, the technique could trace back to the individual Ethernet segment.

Some links did not respond to an applied load. In some cases, it was necessary to go a hop beyond the non-responding links (all the links that are connected to a host that are one hop away from where the technique had traced back to) in order to find a link which, when loaded, affected the packet flow. Sometimes, it was possible to pick up enough of a signal from one of these next layer links that the trace-back could continue. It was a quite manual process, however, which could become difficult when an untestable link had a large number of incident links.

With two exceptions, corporate users appeared to be ignorant of the tests. If the mapping is subtle, and the load applied for short periods, users are unlikely to notice the performance hit, or dismiss it as normal network variability. Early on, testing was confined to a few networks, and the network administrator received enough complaints to notice the activities several times. Subsequent tests appeared to go unnoticed, though in both cases there no attempt to hide the experimental activities. On one network used to generate load, the broadcast UDP chargen packet initiated a broadcast storm on their network. This network had a gain in the tens of thousands. Local users definitely noticed every time it was used, since it brought the network to a halt. Of the 2,000 networks in Lucent, only this one appeared to be unstable in this matter. The Internet likely has an even lower rate of misbehaving networks.

5.5 Alternative Strategies

The proposed technique is hardly ideal. This section will discuss multiple alternative techniques that may be able to reduce the problems with denial of service attacks. Like the proposed technique, some techniques provide mechanisms in the network to trace anonymous packet streams. Other techniques address the problem by reducing or eliminating the ability of hosts to produce anonymous packet streams. This list is by no means complete, but it gives some of the more promising methods.

5.5.1 Filter Return Addresses at the Source

The most desirable way to solve the problem of anonymous packets is to enforce correct source addresses at or near their source via a method called ingress filtering[37]. A company or university should block outgoing packets that do not have appropriate return addresses. ISPs should have similar filters for each of their customers.

This solution is undoubtedly correct. Anonymous packets have no place on the Internet. However, these filters have the unfortunate side effect of making life more complicated, and for large users behind slow routers, they can even degrade performance. For all users, these filters are an additional administrative problem: one more thing to install, maintain, and get wrong. Many Internet entities already have firewalls that enforce this policy, and several RFC's recommend it as essential for any responsible participant in the global routing system. Most firewalls have the ability and capacity to perform these checks. Source-based filtering may also upset mobile networking methodologies.

This solution also requires that most, if not all, Internet sites adopt this approach. ISPs would have to agree to refuse to connect to other ISPs that do not enforce this strategy, a constraint that most ISPs do not impose at this time.

5.5.2 Filtering in Backbone Routers

Routers at the core of the Internet, those running BGP4 and exchanging full Internet routing tables, inherently enforce proper destination addresses on packets, since the routing system is built around forwarding the packet toward the value of this field. Theoretically, routers could perform a similar check on the source address, i.e., drop those with source addresses that are inconsistent with their incoming interface.

Unfortunately, the verification is not nearly this simple, since a packet may come from more than one possible incoming interface, requiring routers to maintain a huge amount of state. Not only do routers not have spare memory resources to maintain this state, they do not have spare CPU resources to perform the verification. In the midst of sustained forwarding rates of millions of packets per second, often operating quite near, if not at, their maximum capacity, router designers must optimize for speed. An additional lookup of the source information would require similar optimization and subsequent re-engineering of many routers, an expensive and unlikely scenario unless ISPs are willing to pay for it.

One could imagine that legal fallout from a particularly damaging attack might force this scenario, and some routers may emerge that support such functionality service without re-engineering. In general, however, the industry has long resisted source-based policy routing, and there is little reason to expect a fundamental change in this mind-set in the near future.

5.5.3 Tracing by Hand

The obvious ad hoc solution to finding a spoofing host is to trace packets back to their physical source manually. This method requires significant cooperation and attention from intervening ISPs, which has proven a problem in past incidents. They may not have the policy, inclination, time, expertise, or instrumentation to help out. Test equipment may not be available for some locations or links within their network. For example, some Cisco routers have been known to crash if `IP DEBUG` is used under sufficiently heavy load.

Further, the traces may be needed off-hours: the Panix attack in 1996[106] started a little after five one Friday afternoon. It may be hard to find someone at any hour at the ISP who can handle the technical details. Sometimes attacks are only solved because a victim happens to be well-connected to admin-able friends at ISPs that are willing to help them out.

This cooperation is helpful, but selective, and slows the process down immensely. A quicker method would be extremely useful.

5.5.4 Shutting Down a Router

One could imagine sending a message to a router requesting that it drop all packets for a particular destination for a about a second. This interruption would be long enough to detect a break in incoming packets, without noticeably affecting service. Implementing this feature would provide an obvious denial-of-service attack of its own. The router could require that requests be strongly authenticated, but there is no infrastructure present for such validation in the current Internet. In self-defense, a router would have to do similar rate-limiting as it does with UDP responses, rendering the feature useless for a significant attack.

Given the ease with which an attacker can hide her attack, it probably is not worth deploying such a service. Without global support, an attacker will simply launch the attack from a place without this feature. This feature would be difficult to deploy and provide only limited usefulness. It is unlikely a large fraction of ISPs would be interested in deploying such a service.

5.5.5 Marking Packets with IP Addresses

Another alternative that has been suggested is to place the IP address of all the routers that a packet goes through during its flight across the Internet. This has two obvious disadvantages: it requires CPU time of the routers and it increases the size of packets, especially in the case of routing loops. Both of these could be reduced by having it mark only every 1 in N packets through a given interface. If N is small enough, a long enough attack would give a complete list of routers along the path, if not their actual order. If N is chosen large enough, the additional router time and packet size increase would be negligible. In practice, randomly varying N may avoid possible problems with routers synchronizing. If N is too small, than the attacker can insert packets into the network that can 'fool' the system into misdiagnosing the path. In practice, keeping only one address in a packet at a time may be desirable to simplify the header.

5.6 Ethics

The proposed methods to trace-back anonymous packets resemble techniques used by hackers. This resemblance raises several ethical questions which must be addressed.

1. *Does the tracing attempt cause more damage than the actual packets?* If the answer is yes, then, obviously, this technique should not be pursued. It is not possible to provide a universal answer to this question; it really depends on the situation. If the anonymous packet stream has shut down the daytime service on a web server, it is perhaps not harming the operator enough to take serious action. If the attackers are crashing the victim's network and denying customers access to the primary service of the victim, then perhaps stopping the attack is worth the cost of congesting a few network links for a few seconds (it almost certainly seems worth it to the victim). This is a judgment call that must be made by any potential user of this method.

2. *Does leaving a service (such as UDP chargen) enabled on a host implicitly mean the owner has given permission to use it?* The technique does not attempt to gain access to private information or crash individual hosts, but it does leverage accessible services from private hosts. However, these hosts have left the UDP chargen enabled (or whatever service is employed).

> The easy answer is yes, but it harbors dangerously close to the common hacker defense that leaving a service with possible security holes running indemnifies those who intentionally exploit it. On the other hand, the technique does not really exploit a security flaw in implementations. Indeed, the technique follows the intended protocol specification exactly. Nonetheless, the essence of the technique is the imposition of a denial-of-service of the attacker's own denial-of-service attack against the victim.

After much consideration, the appropriate answer to this question appears to come down to motivation. While a hacker is generally trying to harm the host, gain access to private information, or journey on an ego trip, the technique leverages the host for a secondary purpose that is helpful to the Internet community.

This argument is almost assuredly insufficient to convince some organizations that reside on the Internet that it is reasonable to use the presented trace-back technique.

5.7 Later Work

Since the original publication of this work, there have been many papers published in this area. Although there is little hope of including all the research, this section outlines the general lines of research.

Much of the later work focused on probabilistic marking schemes as described in Section 5.5.5. Savage, Wetherall, Karlin, and Anderson[85] were the first to propose a specific method to mark the packets and reconstruct the reverse path. Using 16 bits of IP header space, they designed a scheme to reconstruct a 30-hop path in 4,500 packets with 95% probability. Their scheme placed a partial mark into the packet with probability 0.04. These marks were then collected and correlated by hop and then combined to produce a coherent path. Their approach performed poorly for distributed attacks. Doeppner, Klein, and Koyfman[33] proposed a similar scheme based on adding IP options to the packet. Song and Perrig[91] refined the technique to reduce the number of false positives that occur in distributed attacks. Song and Perrig exploited knowledge of the network topology. They developed an approach designed to scale to a few thousand attackers. Dean, Franklin, and Stubblefield[26] used algebraic techniques to achieve similar results. Their algorithm uses 25 bits of the header, more than Sing and Perrig, but did not require knowledge of the topology.

Some theoretic results have followed. Lee and Park[69] analyzed how may alternative paths an attacker can spoof under probabilistic packet marking. Adler[1] analyzed the general problem of a network trying to communicate information (in this case, the attacker's location(s)) statelessly.

Alternative approaches have also been proposed. In 2000, Bellovin proposed in an expired IETF draft sending an "ICMP traceback" message instead of marking the packet. Ioannidis and Bellovin[43] proposed distributed filtering. The idea behind their approach is that attack traffic is concentrated through a few routers while legitimate traffic is more evenly spread. By limiting the amount of all traffic flowing through upstream routers to the victim, routers that forward only legitimate traffic see little traffic and will therefore forward all of it. Routers that see attack traffic see a lot of traffic and will therefore filter both legitimate and attack traffic. The net result is that much of the attack traffic will be filtered before it gets to the victim. The further upstream this filtering can be pushed, the less attack traffic the victim will see.

These two approaches have been combined in more recent work. Yaar, Song, and Perrig[104] proposed marking packets traveling the same path with a unique identifier. The victim can then filter packets based on the path they take. Attacking paths, when detected, can be filtered, removing the attacking traffic. Jin, Wang, and Shin[47] proposed a filtering technique based on TTL values, using the behavior mentioned in a different context in Section 5.2.2. The advantage of their scheme is that it does not require changes in the core Internet and is therefore usable today.

5.8 Conclusions

This technique is not ideal, either in efficiency, speed, or impact on other Internet users. It has been shown that it works on an Intranet, which tends to be a more controlled environment than the Internet itself.

It would be preferable to find a better solution involving ISP coordination and cooperation. Unfortunately, it must be admitted that sometimes the perpetrators are *at* ISPs. As such, an official mechanism that tips them off might be completely impotent.

It is expected that ISPs will drift toward better solutions as their own customers demand assistance.

Chapter 6

Monitoring Link Delays with One Measurement Host[1]

Internet service providers (ISPs) routinely monitor both end-to-end and link delays within their network. This monitoring serves several functions, including:

1. Making the ISP aware of changes in end-to-end delays in the network
2. Identifying the cause of end-to-end delays change (increased congestion on a link, increased delay on a link, rerouting), to aide the ISP in remedy a problem when it arises,
3. Supporting customer service-level agreements, and troubleshooting/segmentation of customer delay complaints,
4. Monitoring congestion within the network, and
5. Providing an input into network engineering (e.g., identify daily or weekly fluctuations)

A common way to monitor delays is to deploy measurement hosts at every or almost every router within the network. One way to do this is to use the router as a measurement host. Many routers manufacturers include monitoring or debugging facilities that can send basic ping (ICMP Echo Request)[73] or RMON-like probes. Unfortunately, these facilities run on the router controllers, consuming limited resources. Router controllers are often fully loaded with standard network operations, such as routing computations and other control functions. The little remaining resources are more effectively spent on monitoring such as bandwidth utilization and router table size, which which would be difficult to do on a separate host. End-to-end delay and link delay can be done naturally by an separate host.

Because using a separate host relieves load on the router controller, ISPs would prefer to deploy separate measurement hosts to monitor delays. This has the additional advantage of keeping all packets in the forwarding plane, as the packets require neither control plane support nor special packet handling or non-hardware-assisted forwarding. This ensures that the measurement packets see the same performance as normal traffic. Unfortunately, measurement hosts are difficult and expensive to install and maintain. Adding a measurement host next to a router requires obtaining approval to install a measurement host, procuring the hardware and software for the host, shipping that hardware to the point of presence (POP) where the router is located, and configuring the host for its particular location. After the host is deployed, it must then be

[1]This chapter represents joint work with Chris Chase.

maintained, administered, and upgraded as needed. Thus, ISPs look to minimize the number of measurement hosts that must be deployed. Techniques such as network tomography enable an ISP to monitor all, or most, of the end-to-end and link delays in their network without needing to deploy a measurement host at every router.

Network tomography[99] is the decomposition of observations on a path into the individual components making up the observation. For network delay, this means taking end-to-end delay, which is the sum of the delays across each link in the path and computing the individual delays. Network tomography relieves the ISP from directly measuring the delay of each link. Instead, an ISP can place measurement hosts at only some of the routers within the network, collect end-to-end delay measurements, and deduce link delays from the resulting measurements.

This chapter presents RCM (router connectivity monitor), a system that uses a single measurement host to monitor link and end-to-end delays within a network, which is probably a tight lower bound. RCM is deployed and operational on an AT&T network that provides VPN services using MPLS. RCM uses tunnels to connect to various edge routers around the network to perform measurements. With these measurements, RCM uses a novel network tomography technique to infer link delays. RCM is designed to appear as a customer to the network, reassuring network operations that RCM is unlikely to operationally affect the network. This makes it easier both to deploy initially and upgrade with new software or new tunnels.

Previous network tomography techniques to measure network delay suffer from various problems making them unapplicable to this network. Many previous techniques require sending packets to routers internal to the network and many probes for each path. On RCM's network, the MPLS tunnels appear as a single link to IP traffic, making it impossible to address internal routers. Moreover, sending thousands of packets to each measured path would require an enormous number of probes, as the number of measured paths must be at least as much as the number of links. RCM's network tomography technique does not need measurements to internal routers and requires few probes per path, when compared to previous techniques.

Another problem with some previous network techniques is that they assume that the path matrix (defined below) is full rank. As will be shown, this is true only if there is a measurement host at every router or if delays are assumed to be symmetric. Placing a measurement host at every router is expensive, and makes network tomography techniques moot. Assuming symmetry without regard to the measurements limits the scope of the technique, as it cannot model an asymmetric network regardless of how asymmetric the measured delays are. RCM does not require a full-rank path matrix, avoiding this problem.

RCM splits the delay across each link into two components: the propagation delay (constant portion) and the queue delay (variable portion). This means that it can distinguish between delay from network design and delay from network congestion. Moreover, RCM can incorporate additional information known about the links, such as the geographic separation, to produce better results. Although RCM's network tomography technique is designed with RCM-like measurements, the technique is applicable to general delay-measurement systems.

This chapter is organized as follows: Section 6.1 describes some related network tomography techniques. Section 6.2 describes RCM's measurement system. Section 6.3 details RCM's network tomography system that infers link delays from its measurements. Section 6.4 evaluates the results from running RCM on AT&T's network. Section 6.5 discusses future work. The chapter concludes with section 6.6.

6.1 Related Work

Previous work by Hero and Shih[89] and Coates and Nowak[23] attempted to determine link delay distributions using unicast probes. Coates and Nowak focused on the case of a single sender, which results in a tree topology. It is not clear how such tree topologies might be combined to produce a coherent model of the entire network. Hero and Shih considered more general probes, but required the path matrix A (defined later) to have full rank, which will be shown to mean that a measurement host must be adjacent to each router. Because they are trying to determine the delay distribution, both require each path to be probed a substantial number of times (1,500 for Hero and Shih and 1,000 for Coates and Nowak). The number of paths that must be measured is at least the number of links in the network. For large networks, this means an extremely large number of probes are required to use their techniques.

Breitbart, et al.[5] focused on the problem of trying to minimize the number of paths that must be measured in order to use a particular method of determining link delays. Their method is able to determine the delay along a path p only if there exists some path α such that path α and $\alpha \cdot p$ are probed. This chapter proposes a network tomography technique that allows arbitrary linear combinations of paths, yielding more data from the same input.

Wei, et al[101] and Duffield[34] focused on the more specific problem of detecting a few poorly behaved links. Wei, et al. presented a method to detect whether or not the poor behavior along a path was the result of one bad link. Duffield considered the problem of finding the smallest set of links which, if they all had high delay, would account for all high-delay paths.

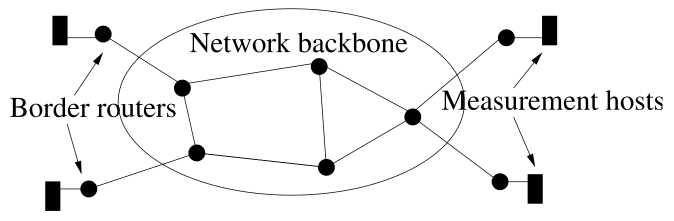
Tsang, Yildiz, Barford, and Nowak[97] considered the problem of determining how much of the delay variability between two routes is the result of the shared path. Their system is designed to look at two routes and only attempts to determine the variable (queue) delay on the shared path.

RCM's network tomography is most closely related to the technique presented by Shavitt, Sun, Wool, and Yener[87], which uses linear algebra to model the system, making it able to model arbitrary networks. However, Shavitt, et al. require a full-rank path matrix, which they obtained be forcing symmetry on the network. Additionally, the technique of Shavitt, et al. is unable to distinguish between propagation delay and queue delay and is not designed to utilize extra information.

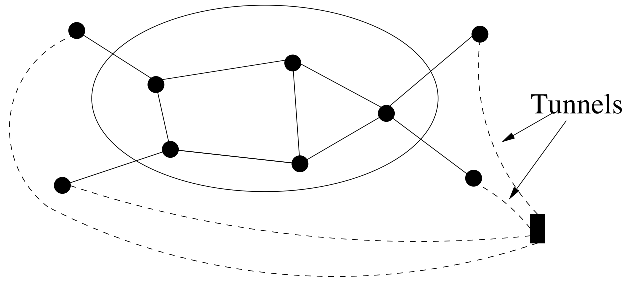
6.2 Measurement System

The goal of network tomography is to determine link delays from end-to-end measurements. The idea is to utilize the diversity of the paths measured to deduce additional information not directly measured. The reason to use network tomography instead of direct measurements is either because direct measurement is not available or because network tomography requires fewer measurement hosts. RCM is designed to collect a diverse set of paths using a single measurement host. Figure 6.1 shows how RCM compares with the standard measurement system. The standard measurement system includes measurement hosts at most or all border routers. It might additionally include measurement hosts at backbone routers. Instead of directly deploying measurement hosts around the network, RCM replaces those measurement hosts with virtual measurement hosts. These virtual measurements hosts are created by attaching a single measurement host to the border routers via tunnels (other connectivity, such as leased lines, would also suffice, but are cost prohibitive for most applications). These tunnels allow the measurement host to send and receive probes from each location in the network to which the measurement host is connected.

The specific network on which the system runs is an AT&T network that provides VPN service using MPLS[82]. The measurement host contains an ATM card configured with a virtual circuit to each border router in the network. These tunnels could be configured to run over the same network using tunneling protocols such as GRE[36] or L2TP[95]. However, because an independent ATM network was available, the system instead uses tunnels (virtual circuits) on that network. In this case, the virtual circuits require no special configuration because they appear normal to the ATM network. The connectivity to the MPLS network is standard as well, as customers are connected via similar ATM connections.



(a)



(b)

Figure 6.1: a) Standard measurement system and b) RCM

The standard measurement system measures end-to-end delays by emitting a packet from a measurement host adjacent to one border router and receiving it at another measurement host. RCM, on the other hand, emits a packet down the source tunnel, destined for the IP address assigned to the destination tunnel. The network then transmits that packet to the border router to which the destination tunnel is attached. That border router the forwards the packet back to the measurement host along the destination tunnel. This measures the sum of the delay out the source tunnel, across the network, and back through the destination tunnel.

One disadvantage of using tunnels is that the raw data is not meaningful to operations. When using a standard measurement system, each measured delay equals the end-to-end delay observed between two border routers. Using tunnels means that each measured delay includes artificial delays from both the outbound and inbound tunnels. Previously, analysis was necessarily only to determine link delays. Now, analysis is required to determine end-to-end delays as well. Even if an ISP only wanted to monitor changes from a baseline, the tunnel delays must still be determined, because a change in a measured delay could be the result of a change in the network or in the delay across one of the tunnels. This is a minor drawback, as analysis is already required to detect which links are responsible for changes in end-to-end delays. Once link delays have been determined, end-to-end delays can be determined by subtracting the tunnel delays from the raw measurements.

Using tunnels somewhat complicates the analysis, as each measurement now includes two more pseudo-links (the tunnels added at the beginning and end of each path). Moreover, all paths become loops instead of paths. In order to determine link delays, the paths must remain sufficiently "diverse". It will be shown in Section 6.3 that the introduction of these pseudo-links does not adversely affect path diversity.

The drawbacks thus far are minor, in that they can be handled by analysis techniques described below. One potential issue that can not be dealt with by analysis techniques is that RCM replaces the need to deploy measurement systems with the need to deploy tunnels. The ISP must still deploy the tunnels to a

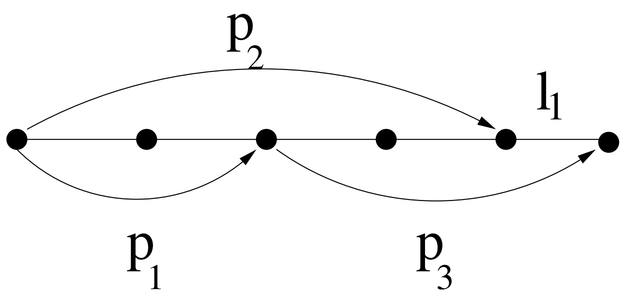


Figure 6.2: Example set of measured paths useful for determining link l_1.

diverse set of routers. Is deploying tunnels better than deploying measurement hosts? Using tunnels instead of measurements hosts has three advantages: deploying tunnels is simpler than deploying measurement hosts, using one measurement host allows more coordination of the measurements, and collecting the raw measurements for analysis is simplified.

Deploying tunnels is much cheaper and faster than deploying measurement hosts. Tunnels can be deployed by changing configurations. As mentioned in the overview, adding a measurement host next to a router requires obtaining approval to install a measurement host, procuring the hardware and software for the host, shipping that hardware to the POP where the router is located, and configuring the host for its particular location. The alternative of using existing hosts requires the availability of compatible hosts with sufficient resources to host the measurement software and gaining permission to use said hosts. For the particular AT&T network on which the system resides, deploying a new ATM tunnel follows nearly the same process as adding a customer, which is done on a daily basis. Tunnels can be deployed in minutes with no effect on the network. Hardware deployment can take weeks or longer. Using existing hosts has the additional danger of affecting other programs running on that host. Using tunnels also allows flexibility in the placement of the measurement host. For the deployed system, the host was placed in the network operations center, where it is easily serviced and managed. Remote measurements hosts must be placed in a POP, which may have limited or no on-site staff, greatly complicating maintenance.

Using tunnels means that the measurement packets are sent and received by the same host. This has additional benefits beyond the obvious advantage of avoiding the problem of clock synchronization. For example, fine-grained scheduling allows the probes to be timed to minimize the load on routers and links. While such fine-grained scheduling is straight-forward within a single host, it would be difficult to do across multiple hosts.

Because data collection is now done by a single host, the data is stored completely on that host. Thus, the raw data does not need to be transferred to a single host to do analysis. This means the network load induced by raw data transfers or distributed algorithms can be avoided or flexibly scheduled to minimize the effect on the network. Analysis of the data can take place instantaneously, as complete data is available to the measurement host at all times. This means the results (generally much smaller than the raw data) can be available quickly and that the data collection can adapt to measurements more rapidly.

6.3 Tomography Technique

Conceptually, RCM's network tomography technique works by taking linear combinations of paths to determine delays of links not directly measured. Consider the set of paths shown in Figure 6.2. The delay across link l_1 should be the sum of the delay across paths p_1 and p_3 minus the delay across path p_2. These linear combinations can be arbitrary complex.

Before defining the technique, some notation is necessary. Denote the (directed) links as $L = \{l_j\}$ and the paths as $P = \{p_i\}$, where each p_i is a subset of L. Define the path matrix, A, as the $\|P\|$ by $\|L\|$ matrix given by $A_{ij} = \|p_i \cap \{l_j\}\|$. If all the paths are simple, then A is a boolean matrix.

RCM's network tomography technique expands on the technique presented by Shavitt, Sun, Wool, and Yener[87], which solved for the delay vector d from the measured delays b using the following equation:

$$\begin{array}{ll} \text{Subject to} & Ad - e = b \\ \text{Minimize} & \|e\|_2 \end{array} \tag{6.1}$$

Equation 6.1 is solvable using least-squares errors techniques[96] if A is full-rank. Unfortunately, A is never full-rank unless each router in the topology is the end-point of a measured path. This would imply a measurement host is next to each router, making network tomography techniques unnecessary.

Lemma 1 *rank*$(A) \leq \|L\| - \|I\|$, *where I is the set of routers within the network which are not the endpoint of any measured path.*

Proof: For all routers along a measured path except the endpoints, the in-degree equals the out-degree. That is, for $x \in I$, the in-degree equals the out-degree for all measured paths. Let y be a router which is the endpoint of at least one measured path. For any $x \in I$, let p_x denote a path from x to y. The path p_x cannot be expressed as a linear combination of measured paths, since the in-degree of x does not equal the out-degree. Each path p_x corresponds to a vector v_x specifying the links in the path. Let $V = \{v_x | x \in I\}$. V is linearly independent for the same reason that v_x is linear independent from the measured paths. Therefore, *rank*$(A) \leq \|L\| - \|V\| = \|L\| - \|I\|$. ■

Shavitt, et al. avoided this problem by assuming that link delays are symmetric. While often true, it limits the ability to model arbitrary networks. Shavitt, et al. additionally allowed negative link delays and did not distinguish between propagation delay and queue delay. RCM addresses each of these issues.

6.3.1 Algorithm Inputs and Outputs

The input to the algorithm is a set of one-way delay measurements for a set of paths over a time period of interest. For the purposes of the algorithm, this set of paths need not be measured from a single measurement host. Thus, the algorithm will work when a standard measurement system is employed. Each path in the input is measured a medium number of times (twenty in the implementation). The output should be the delay across each (directed) link within the network. This chapter will present the algorithm in three steps: conceptual form, basic mathematical form, and relaxed mathematical form. The algorithm does not use computed models from previous time periods, making the algorithm stateless.

Note that the paths over which the delay measurements are taken are assumed to be known. The implementation does this by examining the OSPF configuration. At the time of writing, the system does not have access to live OSPF data. As such, network events such as a link outage and router maintenance cause measurements to take different paths than the one specified. Although such events are rare, the resulting input flaws usually result in RCM being unable to model the system. This problem is solvable by listening to OSPF announcements using technologies such as Route Explorer[44], RouteDynamics[83], or OSPF Monitor[86]. Adding such a monitor is left for future work.

This assumption is also adversely affected by equal-cost multi-path (ECMP). With ECMP, knowing the routing state is insufficient, as the routing state gives a set of possible paths that a packet can take. The decision of which path each packet should take is left to the forwarding engine. Its decisions may be measured by traceroute[46] on non-tunneled networks. On many tunneled networks, including RCM's, the tunnel appears as a single hop, making traceroute unable to identify which particular path is used. Network tomography on a network with large amounts of ECMP that cannot be resolved by traceroute is discussed in Section 6.5. For the network of interest, measurements whose paths are unknown are discarded. Because ECMP is rare, discarding these measurements does not decrease the usefulness of the results.

6.3.2 Algorithm Motivation

The delay across a link has two components: the propagation delay (p) and the queue delay (q). The propagation delay is constant over medium time frames (minutes to days), while the queue delay varies by the millisecond. Given a set of measurements for a particular path, the minimum observed delay corresponds to the smallest total queue delay seen for that path. Presuming congestion is transient and not endemic, something for which all ISPs strive, the smallest observed queue delay should be nearly zero. This means the minimum measured delay should represent only propagation delay. The average measured delay corresponds to the sum of the propagation delay and the average queue delay.

The delays must obey the laws of physics. This means that both the propagation delay and the average queue delay must be nonnegative. In addition, the propagation delay must be no less than the distance between the endpoints divided by the speed of light in the fiber. If the geographic locations of the routers are known, the geographic separation between routers can be used as a lower bound on propagation delays.

There are three goals for the linear program. First, the sum of the propagation and the average queue delay along each path should be close to the average measured delay for that path. Second, the sum of the propagation delay of the links along each path should be near the minimum measured delay for that path. Third, in the absence of data to the contrary, propagation delays and average queue delays should be approximately symmetric.

The first and second goals derive from the definition of propagation and queue delay. However, the second goal is implemented in a slightly unusual way. The obvious method would be to select propagation delays such that the sum of the propagation delays along each path is as close as possible, but not more than, the minimum measured delay. Presuming the goal is to minimize the average absolute difference between the propagation delay along a path and the minimum measured delay, the obvious method has the disadvantage of focusing on links that are used more frequently. Thus, the answer changes if, for example, a path is duplicated. For this reason, RCM minimizes the sum of the average queue delays, which is insensitive to measurement duplication. Trying to minimize the sum of queue delays has the effect of moving as much of the observed delays over to propagation delay as possible without having the propagation delay along any path exceed the smallest observed delay for that path.

The third goal, maximizing symmetry, is perhaps the most controversial of the three. When compared to previous work, the assumption that links must be symmetric is replaced with a goal of maximizing symmetry. If the link delays are, indeed, asymmetric, then the measured delays will be. Previous work would have somehow taken an average of the asymmetric delays to force a symmetric solution. In contrast, RCM is able to model asymmetric networks. However, is maximizing symmetric a reasonable goal? Bidirectional links are almost universally allocated as a single bidirectional circuit. Because the lengths of the light (or electronic) path in each direction along such a link are equal, the propagation delays will be symmetric. It is less clear that maximizing the symmetry of queue delays is proper.

Without this symmetry goal, the problem becomes under-constrained, making the space of valid solutions large. Thus, it is a matter of selecting the most likely solution among the possibilities. In the case of a network with little congestion, the average queue delay is small, making the assumption of symmetric average queue delay unimportant to the overall model. In a network with large amounts of congestion, this could be a poor assumption. However, if the network has high, asymmetric congestion, then the measurements will be asymmetric. RCM would relax this symmetry goal (the weakest of the three) in order to properly model the measurements. Thus, the asymmetry of the input will be reflected in the output. This goal ensures that the output does not have more asymmetry than the input implies. Section 6.4.3 examines symmetry in the particular network on which RCM is deployed.

6.3.3 Mathematical Form

A few more mathematical definitions are necessary before the mathematical form can be given. Let A equal the path matrix, as defined above. Link l_i is the reflection of link l_j if they are between the same pair of routers but in opposite directions. Let R be an $|L|$ x $|L|$ symmetric matrix, given by:

$$R_{ij} = \begin{cases} 1 & \text{if } l_i \text{ is the reflection of } l_j \\ 1 & \text{if } i = j \text{ and there is no reflection of } l_i \\ 0 & \text{otherwise} \end{cases} \tag{6.2}$$

Each row and column in R contains exactly one non-zero entry. Such a matrix is called a permutation matrix, because multiplying R by a vector x permutes the vector. In this case, if p is the propagation delay across each link, then Rp is the propagation delay across the reverse of each link. Thus $\|Rp - p\|_1 = \|(R - I)p\|_1$ is twice the sum of the asymmetry of the propagation delay of each link. For simplicity, define $\overline{R} = R - I$.

Let the vectors b_{min} and $\overline{b}$ respectively correspond to the minimum and average measured delay for each path in the time frame of consideration. Let p_0 be the lower-bound on propagation delays that results from geographic separation. If the geographic locations are not known, set $p_0 = 0$ (the effect of lacking geographic information will be examined in Section 6.4.1). The output of the technique is two vectors, p and q, respectively specifying the propagation delay and average queue delay for each link in the network.

Given these definitions, the basic form of the network tomography technique is:

$$\begin{array}{rc} \text{Subject to} & Ap \leq b_{min} \\ & p \geq p_0, q \geq 0 \\ \text{Goal 1} & \text{Minimize } \|A(p+q) - \overline{b}\|_2 \\ \text{Goal 2} & \text{Minimize } C\|q\|_1 + \|\overline{R}p\|_1 + \|\overline{R}q\|_1 \end{array} \tag{6.3}$$

The first goal is solved as a least square error problem using the Householder QR factorization[96] to project $\overline{b}$ to range(A). This technique ignores the lower bounds on p and q. Goal 2 is solved assuming the result from goal 1. The constant C specifies the relative weight of queue delay and antisymmetry. As C increases, the system becomes more willing to increase antisymmetry to reduce queue delay. $C = \infty$ is equivalent to minimizing $\|q\|_1$ and, subject to that minimum, minimizing $\|\overline{R}p\|_1 + \|\overline{R}q\|_1$. Figure 6.3 shows the antisymmetry and total queue delay for different values of C for the measured network. The behavior is similar for values of C between 1 and 10^6. For $C = 10^7$, the antisymmetry increases from 180 to 70,000 to accommodate a decrease of 0.016 in the queue delay. This focuses undesirably on minimizing queue delay without regard to antisymmetry. Large C values have the additional disadvantage of magnifying numerical errors. After experimentation, $C = 100$ was selected for the network in consideration. This still strongly favors a decrease in queue delay, as it allows the linear program solver to accept an increase of 100 milliseconds in asymmetry in order to reduce the total queue delay by 1 millisecond. As implied by Figure 6.3, other values around $C = 100$ would yield similar results.

The problem with the basic form of the system given in equation 6.3 is that measurement errors can result in the system having no solution. A small measurement error can result in the basic form missing a reasonable solution because the delays are slightly below the lower bounds. Equation 6.3 cannot distinguish

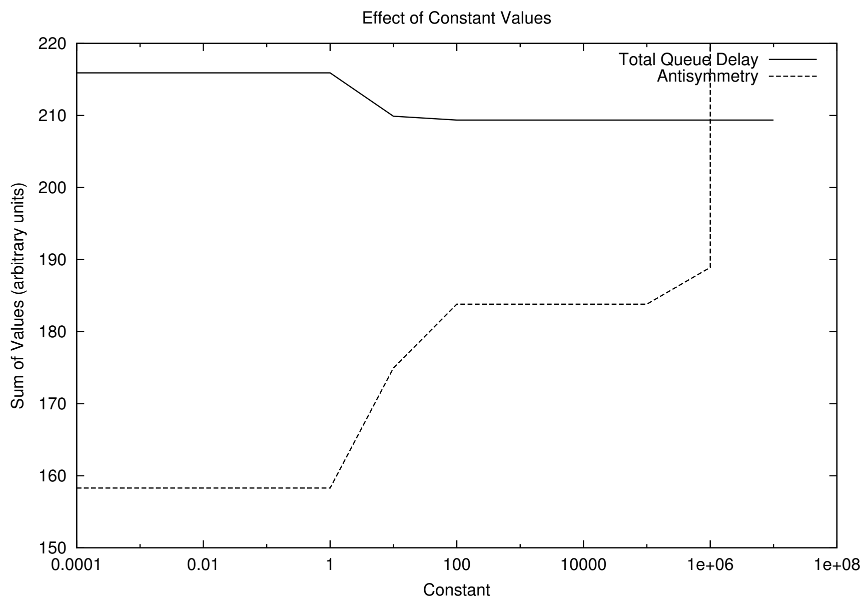


Figure 6.3: Effect of constant C on total queue delay and antisymmetry.

between that and no remotely reasonable solution existing. For this reason, the basic form must be relaxed by introducing slack variables p_e and q_e. These slack variables allow the propagation and queue delays to be less than the specified lower bounds. The relaxed formulation is:

$$\begin{array}{ll}\text{Subject to} & Ap \leq b_{min} \\ & p + p_e \geq p_0, q + q_e \geq 0 \\ & p_e \geq 0, q_e \geq 0 \\ \text{Goal 1} & \text{Minimize } \|A(p+q) - \overline{b}\|_2 \\ \text{Goal 2} & \text{Minimize } \|p_e\|_1 + \|q_e\|_1 \\ \text{Goal 3} & \text{Minimize } C\|q\|_1 + \|\overline{R}p\|_1 + \|\overline{R}q\|_1 \end{array} \tag{6.4}$$

The relaxation goal (goal 2) is solvable using linear programming. This form has the advantage of quantifying the violation: $\|p_e\|_1 + \|q_e\|_1$. Large violations (more than 0.1 milliseconds total), based on experience, reflect errors in the inputs, either the measurements, the geographic separation bound, or the path matrix. This will be discussed in more detail below.

6.3.4 Requirements

For RCM's network tomography technique to function properly, a few requirements on the input are necessary. The first requirement is obvious: RCM cannot determine the delay across any link that is not in at least one measured path. This requirement is unavoidable, as the measurements contain no information about links not along any measurement path. Despite being obvious and unavoidable, it is an important consideration when selecting the set of paths to measure.

The less obvious requirement is that the measured paths must be "diverse enough". Diversity will be defined more precisely in a moment, but the basic form of diversity is that no internal router can have degree two in the undirected form of the network. Consider the two cases shown in Figure 6.4. Any path will either traverse both links or traverse neither link, giving low path diversity. If a path traverses both links, then packets will be take 10 milliseconds to traverse both links in either case. Because the behavior of network

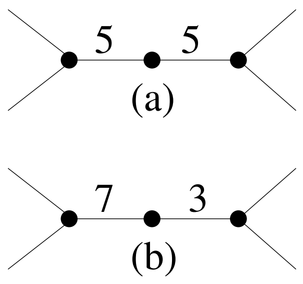


Figure 6.4: Example of two indistinguishable networks with a degree-two router in an undirected network.

(a) and network (b) are the same, it is not possible to determine how the 10 millisecond delay should be divided between the two links.

In the deployed measurement system, the border routers have degree two. However, the measurement system measures the delay along the path path from the measurement host, across the tunnel to the border router, and then back across the tunnel to the measurement host. This path traverses only one of the two (undirected) links, breaking the assumption. This works because one of the end points of the links is the measurement host. Measuring such paths is usually not possible for internal routers of the network.

The true measure of diversity comes from linear algebra: $\mathit{rank}(A)$. Ideally, $\mathit{rank}(A) = \|L\|$, but Lemma 1 states $\mathit{rank}(A) \leq \|L\| - \|I\|$. For RCM's measurements, all the measured paths are cycles. In this case, $Ax = b$ cannot be solved for any x_i. However, if $\mathit{rank}(A) = \|L\| - \|I\|$, then $x_i + x_j$ is solvable when link l_i is the reflection of link l_j (otherwise, adding that path would increase the rank without altering $\|I\|$). That is, all of the rank deficiency is a result of directing the graph. Note that undirecting the graph is equivalent to presuming exact symmetry. That is, $\overline{R}x = 0$. Thus, $\mathit{rank}(A) = \|L\| - \|I|$ is equivalent to $\mathit{rank}(A||\overline{R}) = \|L\|$, where $||$ is the vertical matrix join operation.

This requirement is difficult to evaluate except by building the path matrix and computing its rank. For RCM's network, A is full rank when tunnels are constructed to each edge router.

6.4 Results

RCM is deployed and operating on one of AT&T's networks. This section presents the results of RCM on that network. To protect the proprietary nature of the data while still allowing for discussion, all values are given in arbitrary, but consistent, units.

There are several questions about the techniques and results addressed. The first question is whether or not the output delays are meaningful. If the output delays are wrong, then they are clearly not useful. This question is addressed by comparing the computed delays with the expected delays. One way to do this is to examine the use of the relaxation from equation 6.3 to 6.4. Another technique is compare the output delays to data withheld from consideration, whether that data is withheld artificially or by design. This is done by comparing the computed model with lower bound data not used by the algorithm. The third method is to look at the stability of the results. The variation between time periods should be commensurate with the variation in the measurements. Otherwise, the system is likely to be suffering from numeric instability.

The second question is the efficacy of the method. That is, is it able to assign delay changes to a particular link? If not, the system is able only to determine when there might be a problem in the network and not be

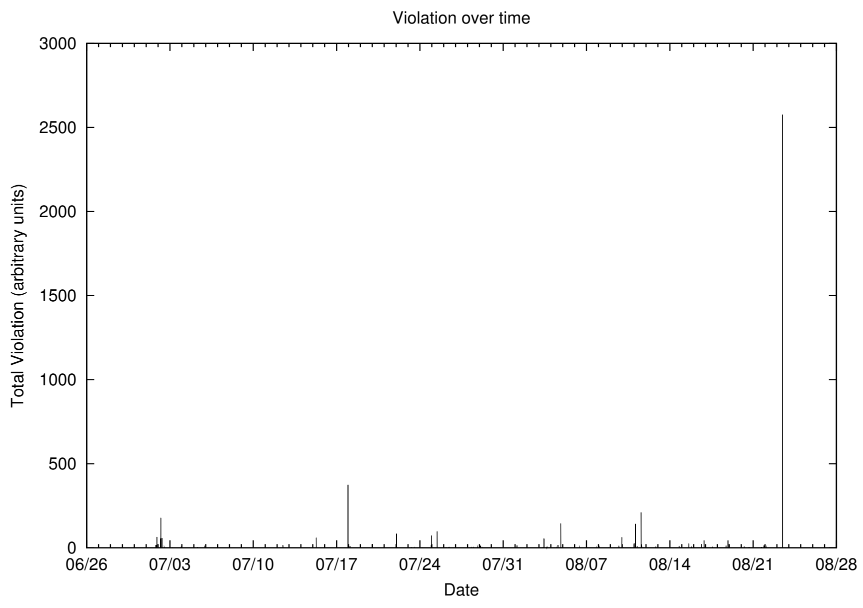


Figure 6.5: Sum of constraints violations for each time period

able to pinpoint the problem. In fact, the system would not even be able to conclusively distinguish between delays changing in the ATM network and delays changing in the MPLS network, leading to false alarms.

The last question is how reasonable the symmetry goal is for this particular network. RCM assumes symmetry in the absence of data to the contrary. If the resulting model routinely contains large amounts of asymmetry, then this may be a poor assumption.

For the network on which the system is deployed, the measured delays vary little over short time frames (milliseconds). The variation is is the order of magnitude of the expected error of the measurement system. As such, the network has little actual queue delay. This small queue delay is reflected in RCM's output. While small queue delay is a good property for a network to have, it makes meaningful analysis of queue delay difficult at best. Because queue delays are all nearly zero, this analysis focuses on propagation delays.

6.4.1 Meaningful Results

As mentioned, the first item to consider when looking if the results are reasonable is the extent to which the slack variables in equation 6.4 are employed. Figure 6.5 shows the total violation ($\|p_e\|_1 + \|q_e\|_1$) for each time period over two months. The violation is nonzero in only 113 of the 7,722 time periods shown (1.5%). There are two known causes for non-zero violation: time division and routing knowledge. Most of the cases of non-zero violation result from the way in which time is divided. Currently, time is modeled into ten minute periods. If the behavior of the network changes in that ten minute time period, then the algorithm is effectively trying to model two different behaviors with a single model. This difficult task can result in a solution with non-zero violation. Properly dividing time into epochs of similar behavior would eliminate such problems. That problem is tangential to the problem determining delays once time has been properly divided and is left to future work.

Persistent non-zero violations can result from changes to the network. As mentioned above, the current system does not include an OSPF monitor. Instead, it uses manual dumps taken from OSPF. If a link becomes disconnected or a router is taken offline for maintenance, the topology information becomes incorrect. This results in a difference between the paths taken by the measurement packets and what the system

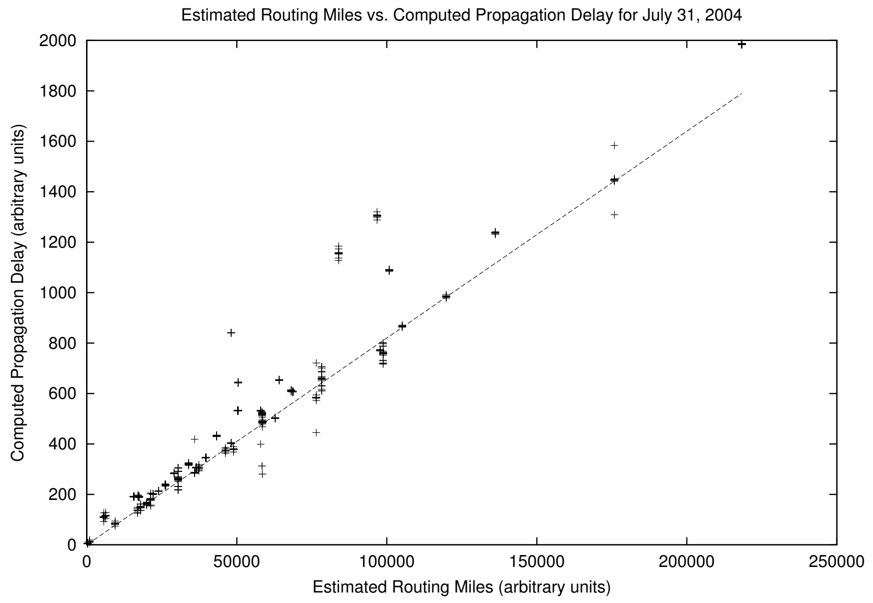


Figure 6.6: Plot of router miles versus computed propagation delay. The dashed line represents speed-of-light propagation.

believes them to be. Several existing systems are able to monitor OSPF[44, 83, 86]. Changes in topology are much more rare than changes in link delays Of the 113 time periods with non-zero violation, 95 are the result of time division problems and 18 result from topology changes.

Note that when the violation is zero, RCM produces the same result as Equation 6.2 would. When the violation is non-zero, Equation 6.2 would yield no solution. Because it uses the relaxed form given in Equation 6.4, RCM is able to produce a potential model of the system. The non-zero violation reports that the produced model is inaccurate and gives a quantitive value of its inaccuracy.

Another question to address is the how well the results correspond to unknown data. For each link in the network, there is an estimate of the number of "routing miles" traversed by that link. Routing miles are computed by looking at the network beneath the MPLS network and approximating the actual number of miles of fiber a packet must travel to traverse a link. Because it is an approximation, routing miles data is not used to raise the lower bound on propagation delays. Despite not being an input, the routing miles should closely correspond to the propagation delay. Figure 6.6 shows the relationship between routing miles and the computed propagation delay for every time period on July 31, 2004. For most links, routing miles correspond to the computed propagation delay: 83% of the computed propagation delays are within 100 units (5% of the maximum propagation delay) of the delay implied by routing miles. Routing miles are computed with knowledge of the network and limited knowledge of routing, while the propagation delay is computed from measurements. Therefore, when routing miles diverge from the computed propagation delay, it is not clear which value is more likely to be in error. In any case, RCM yields propagation delays that closely correspond to routing miles for most links.

Another comparison with withheld data can be performed by artificially withholding the geographic knowledge. This also models the case where a user is trying to use network tomography to determine link delays, as a user is unlikely to have geographic knowledge for all routers in the network. Figure 6.7 compares the computed propagation delays with and without geographic knowledge. Although 40% of the computed

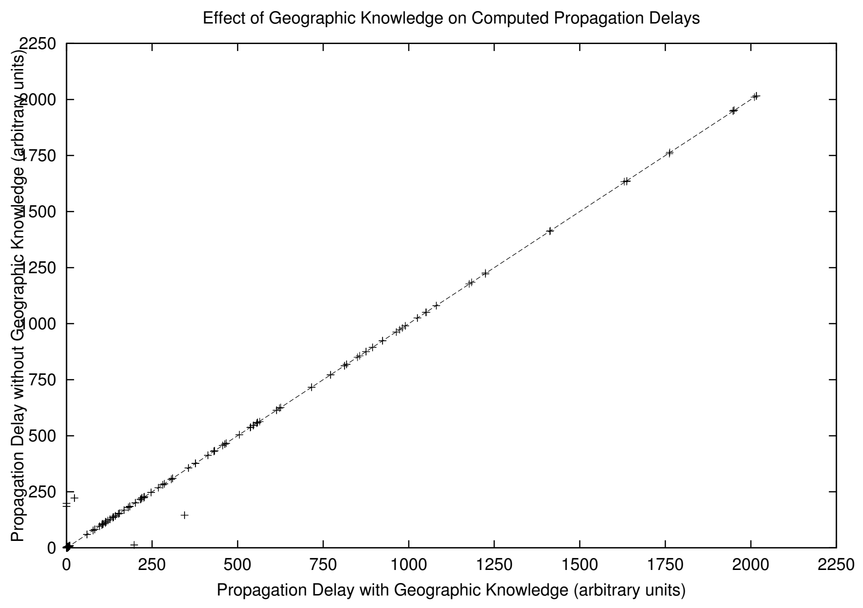


Figure 6.7: Comparison of results with geographic knowledge and without.

delays changed when geographic knowledge was removed, only 4% changed significantly[2]. All significant changes occurred in links whose propagation delay is less than seven units. The computed propagation delay of long links is not affected by the removal of the geographic knowledge. Since the resulting propagation delays violate the geographic knowledge, the geographic knowledge is helpful. However, the system is not heavily dependent on these bounds. Geographic knowledge improves the accuracy of computed propagation delays primarily for links with a low propagation delay.

Besides constraint violation and comparison with withheld data, another expected behavior is that propagation delays should be stable. Thus, the propagation delay across a link might change occasionally when the link is rerouted within the underlying network, but such reroutes should be rare and result in sharp measurement changes. Outside of these abrupt changes, the propagation delay should not change. Some variation in the computed propagation delay is expected due to unavoidable measurement errors, but such variations should be minimal. Because the algorithm is stateless, all stability is a result of the underlying algorithm. Figure 6.8 shows that the computed propagation delay for a particular link over three days. Although this link's computed propagation delay changes little over these three days, and yet observed delays along some paths changed significantly. These changes, discussed in Section 6.4.2, were determined to be the result of changes in the delay over other links. In the face of the network changing, the computed propagation delay across this link is stable, varying by no more than 20 units from the median of 163.2 and rarely more than 3 units. This link's stability is representative of all the links in the network.

6.4.2 Efficacy

Figure 6.9 shows the measured delay between a pair of tunnels (border routers) over three days. At approximately midnight on July 26, the delay changed from 2600 to 2800, dropping back to around 2600 about 21 hours later then dropping to 2400 six hours after that. As the routing of this path did not change, these fluctuations must be the result of a change in the propagation delay across links in the path. Notice,

[2]Significant is defined as a move of at least 2% and at least 0.1 units. This definition falls within an large gap: the smallest significant move was 137% and 4.0 units.

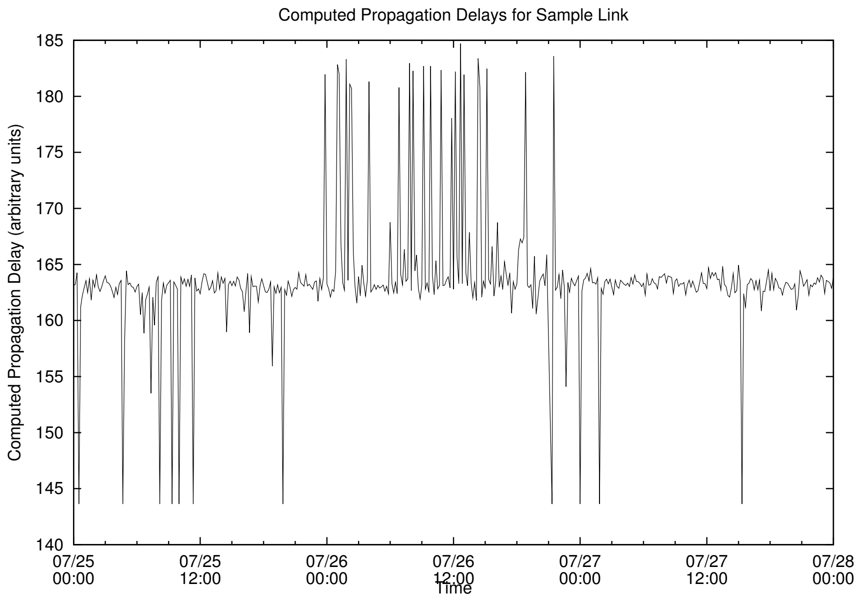


Figure 6.8: Stability of computed propagation delay over three days.

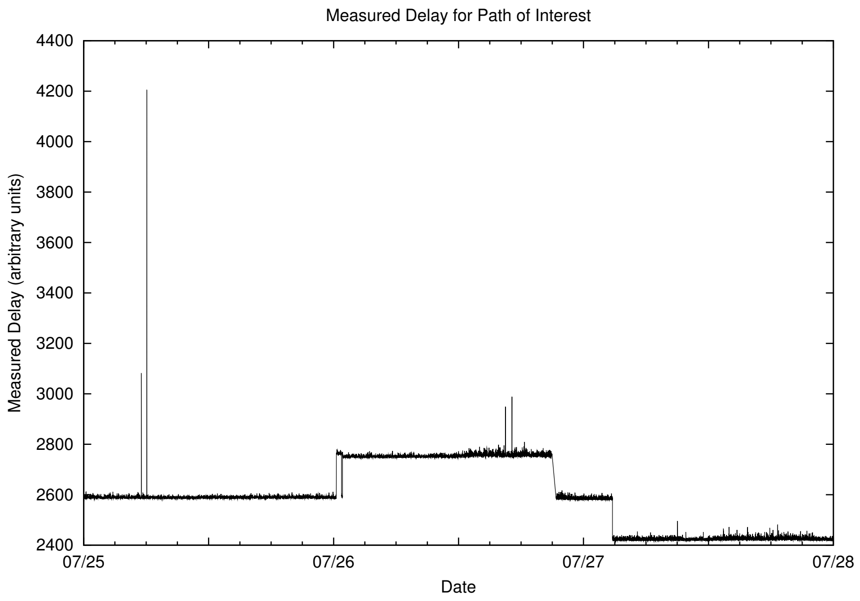


Figure 6.9: Measured delay across a path over three days.

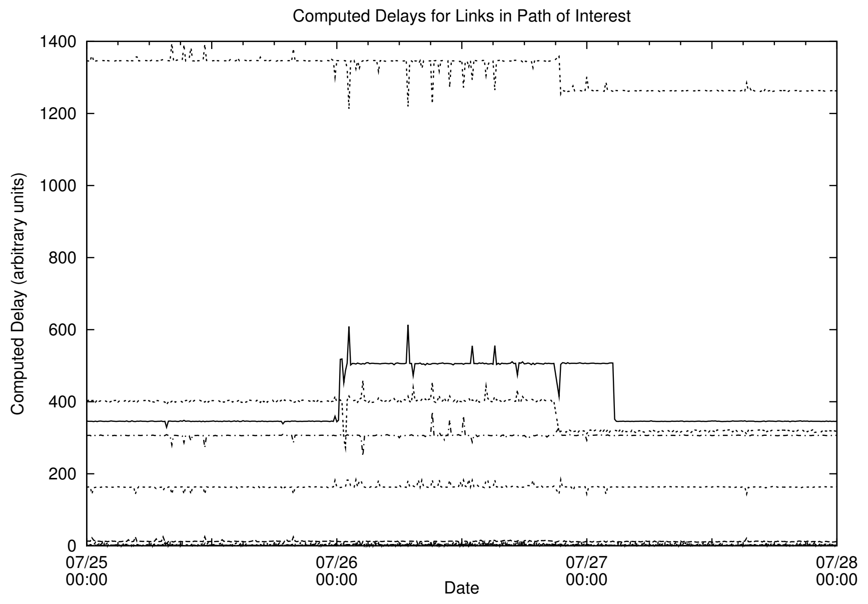


Figure 6.10: Computed propagation delays for each link in the path whose measurements are given in Figure 6.9.

however, that the measured delays in each of these modes of behavior are are nearly constant, with only six measurements (out of more than 30,000) varying more than 2% from the norm of the current mode. This implies that most changes in observed delays are the result of propagation delays of the links used and not a result of congestion (queue delay).

Figure 6.10 shows how the RCM assigns delay to the propagation delays of the links along the path measured. It shows that the rise from 2600 to 2800 is due to a change in the delay across the link represented by the solid line. The drop back to 2600 from 2800 that occurred 21 hours later is not, in fact, that the link returned to its previous propagation delay, but rather the propagation delay over the two links represented by the top two dashed lines changed. These two lines correspond to the two tunnels used by the measurement. Thus, despite the measurements returning to nearly the original level, the actual delays across the path in the MPLS network remain 200 higher than normal. The delays across the MPLS network did not return to normal levels for another six hours. The change in the delay of the link in the MPLS network was confirmed, using other means, as a reroute of the MPLS link in the network upon which the MPLS network runs.

6.4.3 Symmetry

As mentioned above, perhaps the most controversial piece of RCM's network tomography technique is the symmetry goal of minimizing $||\overline{R}p||_1 + ||\overline{R}q||_1$. Without this term, the system is under-constrained. Thus, this term has the effect of selecting among a set of otherwise equally-good solutions. A valid question is whether or not a symmetric solution is more likely than a non-symmetric one. Regardless, the solution is improved over previous techniques that forced symmetry without respect for the symmetry of the measurements. RCM allows asymmetric link delays when the measurements are asymmetric. The symmetry term has the effect of avoiding artificial symmetry not implied by the measurements.

To determine whether or not this is a reasonable goal for this network, consider the symmetry of the input and output data. If the measurements are symmetric and can be modeled by nearly symmetric link delays, then maximizing symmetry is probably a reasonable goal. On the other hand, if asymmetry in rampant in

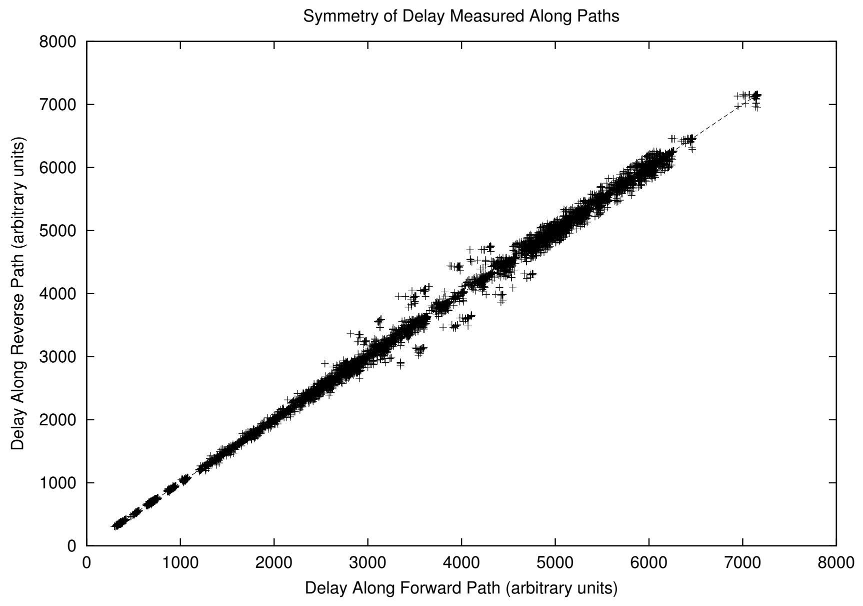


Figure 6.11: Symmetry of average measured delay across each path. Forward and reverse are arbitrary. The dashed line represents exact symmetry.

either the input or output data, then selecting the most symmetric solution may not be the best method. Note that if the output is not symmetric, then previous techniques that forced exact symmetry would also not be able to model the system well.

Figure 6.11 shows the symmetry of the measured delays for one day. This figure plots the measured delay of the forward path against the measured delay of the reverse path, where delays are averaged over twenty measurements performed over ten minutes. Because routing on this network is symmetric for most paths, the measured delays are, for the most part, nearly symmetric. When multiple paths exist with equal costs between two border routers, the forward and reverse paths are assigned one of the equal-cost paths pseudo-randomly. This can result in a forward path that differs from the reverse path, resulting in asymmetric path delays even if link delays are symmetric. ECMP is responsible for all the data points where the forward and reverse path delays differ by more than 200 units.

Figure 6.12 shows the symmetry of the computed propagation delays. The computed propagation link delays are nearly symmetric. More than half (55%) of the computed propagation delays are exactly symmetric. 93% of the computed propagation link delays are symmetric within either 5% or 2 units, although a few links are strongly asymmetric: 0.06% of the computed propagation delays are antisymmetric by more than 100 units. Thus, almost all of the output propagation delays are nearly symmetric.

Because the measured delays are symmetric except when routing is not symmetric and because the measurements can be modeled with nearly symmetric delays, maximizing symmetry appears to be a reasonable goal. This does not rule-out asymmetric delays, but combined with the fact that the two directed links comprising a connection should be allocated as a single bidirectional circuit, it makes a strong case for it. This analysis has ignored queue delays. As mentioned, the computed queue delays are small for most time periods, averaging less than one unit per link. Thus, queue delays are trivially symmetric within either 5% or 2 units in all but a handful of cases. For a network with larger queue delays, the symmetry goal may not always be correct. However, ISPs strive to design and operate their networks such that all queue delays are small.

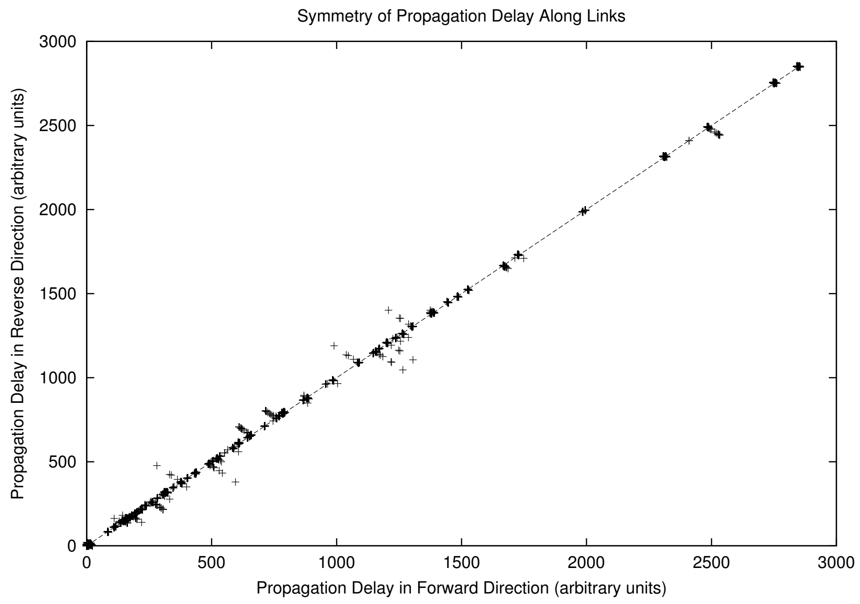


Figure 6.12: Symmetry of computed propagation delay for each link. Plot represents data from 144 time periods (an entire day). Forward and reverse direction are arbitrary.

6.5 Future Work

This chapter presented several related problems considered future work. Firstly, the current system lacks live routing information. This can be solved using any of the route monitors available[44, 83, 86]. Secondly, the current system naïvely divides time into ten minute intervals without regard for the measurement values. Future work could divide time based on epochs of similar behavior. Thirdly, RCM requires that the route taken by each measurement is known. When the network includes equal-cost multi-path[63], and traceroute[46] cannot identify the path taken by the probes, the exact route is not known. Instead, it is known to be one of a set of alternatives. Although ECMP exists on RCM's network, it is rare enough that discarding the few measurements where the exact path is not known does not adversely affect usefulness of the results. The difficulty with ECMP is that solving the link inference with ECMP is, in the general case, NP-hard.

Consider a formalization of the problem of determining link delays with no error on a network with ECMP: given a weighted graph $G = (E, V)$, a set of measurements $\{m_i\}$, let $P_i = \{p_i^j\}$ be the set of all minimum-weight paths between the source and destination of measurement m_i. Find a delay function $d{:}E \rightarrow R^+ \cup \{0\}$ such that:

$$\forall i \; \exists j \text{ s.t.} \sum_{e \in p_i^j} d(e) = m_i$$

Call this the routed multi-path weight assignment (RMWA) problem. A problem with this formulization is that P_i can contain exponentially many paths, which is unrealistic in practice. The routed k-path weight assignment (RkWA) problem is any RMWA instance where $max_i \|P_i\| \leq k$. Any instance of RkWA is an instance of RMWA, regardless of k.

Lemma 2 *The routed 2-path weight assignment (R2WA) problem is NP-complete.*

Proof:

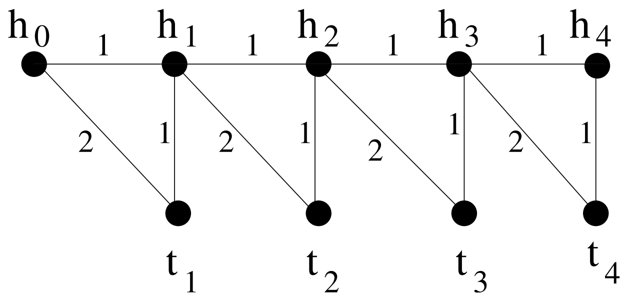


Figure 6.13: Example, for $n = 4$, of graph constructed in proof of Lemma 2.

This will be proven by showing a reduction of set partitioning to R2WA. Consider a set $S = \{x_1, x_2, \ldots, x_n\}$ to partition into two sets with sum $s = \frac{1}{2}\sum_i x_i$ (an instance of set partition). Construct a graph with a head node h_0 and two nodes for each element of the set, h_i and t_i. Let $E = \cup_{i=1}^{n}\{(h_{i-1}, h_i, 1), (h_i, t_i, 1), (h_{i-1}, t_i, 2)\}$ (the third value of each triplet is the routing (OSPF) cost of the edge). An example of this graph for $n = 4$ is shown in Figure 6.13.

Let the measurement set contain four types of measurements. The first two types are measurements from h_{i-1} to t_i, one with value x_i and the other with value 0. The third type is a measurement from h_i to t_i with value 0 (forcing $d(h_i, t_i) = 0$). The fourth type is a single measurement from h_0 to h_n with value s. The path set for the last two measurement types contains only one path. The path set for the first two measurement types contains two paths: (h_{i-1}, t_i) and (h_{i-1}, h_i, t_i).

If a solution to the set partitioning problem exists, then let $X \subset S$ with sum s. If $x_i \in X$, then let $d(h_{i-1}, h_i) = x_i$ and $d(h_{i-1}, t_i) = 0$. If $x_i \notin X$, then let $d(h_{i-1}, h_i) = 0$ and $d(h_{i-1}, t_i) = x_i$. The resulting weighting makes the delay across $(h_0, h_1, h_2, \ldots, h_n)$ equal to $\sum_{i=1}^{n} d(h_{i-1}, h_i) = \sum_{i|x_i \in X} x_i = s$. Thus, the delay function solves R2WA.

If a solution to the R2WA problem exists, then let $X = \{x_i | d(h_{i-1}, h_i) = x_i\}$. Because of the two measurements from h_{i-1} to t_i, $d(h_{i-1}, h_i) \in \{0, x_i\}$. Thus, $\sum_{i|x_i \in X} x_i = \sum_{i=1}^{n} d(h_{i-1}, h_i) = s$. Therefore, the elements of X sum to s, and thus X solves the set partitioning problem.

Therefore, set partitioning is reducible to R2MA. Since set partitioning is NP-complete, R2MA is NP-complete. ■

Note that many restrictions of the problem do not change the result. The same proof will work when the problem is restricted to graphs of bounded degree (for bounds of at least four). The proof also works if the input is a weighted digraph instead of a graph. Only minor revisions to the proof are necessary if all links have the same weight.

R2WA arises naturally from a network running OSPF with ECMP. Thus, Lemma 2 states that optimally assigning link delays in the presence of ECMP is intractable in the general case. It is not clear if the structure in real networks could be exploited or if heuristic or approximate techniques are necessary. Determining link delays on a network with ECMP is left as future work.

6.6 Conclusions

This chapter describes RCM, a delay measurement and network tomography system to monitor end-to-end and link delays within a network using a single measurement host. RCM's measurement system avoids the need for many measurement hosts by using tunnels to collect a diverse set of unicast measurements. This makes RCM easier, cheaper, and faster to deploy, operate, and maintain.

RCM's network tomography technique uses the measured delays to determine the propagation and queue delay for each link. RCM's network tomography technique works on an arbitrary topology without needing to assume route symmetry or delay symmetry. RCM does not require a measurement host at any internal nodes. It requires a few dozen probes of each path instead of thousands, while still being able to determine both the propagation and queue delay of each link.

RCM cannot deal with equal-cost multi-path routes. It was shown that a generalized, exact form of the problem with ECMP is NP-complete. If path selection is done by looking at source and destination IP addresses, then the path is consistent across measurements and may be measurable using traceroute[46]. In a tunneled network where ECMP is common, heuristic techniques may be necessary.

RCM is deployed and operational on an AT&T's network, from which results were presented. The computed delays correspond with expected behavior. In particular, the slack variables are rarely used, the delays match well with routing miles, the delays change little when geographic knowledge is removed, and the delays are stable over large time frames. In addition, when measured delays shift due to a change in one or more link delays, the system is able to determine which link delays changed.

Chapter 7

Firewall Configuration and Anonymous DNS[1]

From 1980 to 2000, the Internet grew from around 200 hosts to over 100 million[45]. As more and more corporations connected their internal networks to the Internet, hackers discovered that their internal networks were not secure. While this poses limited risk if only employees are able to access the network, once a company connects its network to the Internet, it becomes vulnerable to attacks from anyone on the Internet. To combat this without disconnecting their network from the Internet completely, corporations chose to limit their connectivity by deploying secure Internet gateways[15], which came to be known as “firewalls”. Firewalls examine the traffic between two or more networks and selectively drop packets based on a set of rules. Recently, users have have started running software on their own host that acts as a firewall protecting access to just that host, which have come to be known as “personal firewalls”.

Firewalls are an important first-line defense mechanism to limit network access to hosts and arguably the simplest and most popular form of network defense. How prevalent are firewalls? How securely are they configured? These questions provide important information about firewall deployment on the Internet and could unveil potential security concerns. This chapter seeks answers to these questions and many other questions related to the deployment of these security defense mechanisms.

There are two basic philosophies when configuring a firewall: block by default and block by exception. In block by default, the default behavior is to not allow access to IP addresses behind the firewall. The firewall allows access from the Internet to only particular services on particular hosts. This is common in higher-security networks, such as large corporate and government networks. In block by exception, only access to certain services is disabled. This is more common in lower-security networks, such as universities and residential ISPs. In these cases, the network operator is not expected to block services, and, in fact, even limited blocking may be considered undesirable by network users. In block by default, the firewall administrator decides which services are useful enough to specifically allow through the firewall. In block by exception, the firewall administrator decides which services pose a high enough security risk to block them at the firewall. Block by default is preferred by security experts and is more common in practice. An interesting question one may ask is what percentage of IP addresses are beyond firewalls that block by default.

Another question related to the firewall analysis is to study the behavior of publicly accessible services. How many hosts are running public servers for different services? Do servers often run multiple services?

[1]This chapter represents work with Dawn Song, published as “A Security Study of the Internet: An Analysis of Firewall Behavior and Anonymous DNS” as a technical report at Carnegie Mellon University[9].

To increase security, it is recommended that administrators split public services across hosts. If a host runs many services, compromising any of them compromises the entire host. By splitting the services between multiple hosts, the likelihood that any one host is compromised is decreased and the impact of gaining access to a host is decreased, since taking the host offline affects only the services running on that host and services that depend on those services. Economic realities, however, are such that operating more hosts increases hardware and administrative costs. Thus, many installations run multiple services on hosts to reduce costs.

Corporate networks are often separated from the Internet in naming as well. That is, hostnames valid within a corporate network may not resolve using domain name service (DNS) outside that corporation. This is known as "split DNS". This chapter will focus on split reverse DNS, where, for example, 192.0.2.12 may reverse resolve to "inside.example.com" within a corporation, but reverse resolve to "host-192.0.2.12.example.com", or some equally uninformative hostname, on the Internet. This provides a layer of obfuscation, as it makes it more difficult for an outside attacker to determine whether 192.0.2.12 is in use, much less what, if any, services it provides. Split DNS is somewhat analogous to firewalls for DNS: the Internet is allowed only partial access to internal DNS, usually limited to resolving external servers.

Split DNS is commonly implemented as assigning mechanically-generated hostnames to IP addresses. This chapter focuses on mechanically-generated hostnames derived from the IP addresses, such as the example above. Such hostnames will be referred to as "anonymous DNS" because the hostname reveals little about the IP address. Split DNS is not the sole reason for anonymous DNS. For example, ISPs commonly do not give meaningful hostnames anywhere to the IP addresses of their residential customers. This chapter examines the prevalence of anonymous DNS and how much information is actually contained in reverse lookups. Most companies using anonymous DNS generate hostnames for all IP addresses they own, regardless of whether or not that IP address is in use.

Anonymous DNS affects attempts to estimate the number of hosts on the Internet that use DNS. Before firewalls were widely deployed, ping[73] could be combined with sampling to estimate the number of hosts on the Internet. As firewalls grew in popularity, ping became increasingly inaccurate. This caused projects like ISC Internet Domain Survey[45] to use DNS instead. Anonymous DNS makes using DNS inaccurate. This chapter quantifies the error resulting from using DNS to count hosts.

In summary, this chapter specifically addresses the following seven questions:

1. How prevalent are firewalls? What percentage of IP addresses are behind firewalls?
2. What do firewalls block? What services are commonly allowed and blocked by firewalls?
3. How much do firewalls block? What percentage of IP addresses are beyond firewalls that are block by default?
4. How is firewall behavior correlated? Given a firewall blocks or allows a service, what other services is that firewall likely to block or allow?
5. What accessible services do hosts run?
6. How are running servers correlated? Given a server running a publicly available service, what other publicly available services is it likely to run?
7. How prevalent are anonymous hostnames? What fraction of hosts and IP addresses have anonymous hostnames?

To the best of the author's knowledge, this work is the first attempt to systematically study the behavior of firewalls and anonymous DNS on the Internet. This chapter proposes methodologies and describes findings to address these questions. Many results are surprising and provide new insights for understanding the deployment of security defense mechanisms on the Internet.

The rest of the chapter is organized as follows: Section 7.1 reviews previous work. Section 7.2 details the methodology used. Section 7.3 presents the results, including answers to the questions posed above. Section 7.4 gives the conclusions.

7.1 Previous Work

Most previous work which examined behavior of hosts on the Internet focused on the version of services running on systems. Early previous work includes multiple RFCs: 669[29], 679[31], 701[28], 702[30], 703[32], 751[54], 844[22], 847[102], and 876[90]. RFC 669, 679, 701, 702, and 703 are periodic surveys looking at telnet server behavior to observe the deployment of the new telnet protocol. RFC 751 is a survey that looks at how many mail daemons allow e-mail to unknown recipients and how often a particular method to send e-mail is available. RFC 844 looks at the deployment of ICMP. RFC 846 looks at the deployment of TCP. With the exception of RFC 751, all of these exclusively dealt with the rate of the deployment of a new service or protocol. None of these surveys included more than 400 hosts, because the Internet did not have more than 400 hosts. That is small-scale when looked at in modern terms.

More recent work by Provos and Honeyman[80] studied the deployment of a new version of SSH[105] after the discovery of a vulnerability in one version of the protocol. They scanned 400,000 IP addresses at University of Michigan daily to determine the version of SSH server running on the systems that responded (around 2,300 of the IP addresses). Again, they were focused on the version of servers running, rather than looking at responsiveness of systems.

Several projects have used DNS to estimate the number of hosts on the Internet. They include Netsizer[67], ISC Internet Domain Survey[45], and RIPE Region Hostcount[81].

7.2 Methodology

To analyze firewall behavior, IP addresses were probed for their responsiveness to particular services. Although some firewalls contain specific rules that allow particular external hosts special access to their corporate network, this chapter focuses on the behavior of a request from a general host. This is the access that most hosts will see and it is not clear how these special hosts could be reasonably detected anyway. Two types of responsiveness are considered. The first category is a host responding to the service request. This does not mean that the host necessarily runs the service, simply that no firewall blocks the request. The second category is a host listening for a service. This means that no firewall blocks the request and the host offers initial access to the service. The specific packet responses implying these two response types are service-specific and are detailed below.

Testing firewall behavior is composed of four pieces: selecting the services to test, selecting the IP addresses to test, testing the IP addresses, and separating firewall responses from end-host responses. Because there are too many services to test all of them, a representative subset must be selected. Even if the set is limited to routed or allocated IP addresses, there are too many IP addresses to test all of them. Thus, IP addresses must be sampled using a representative subset of sufficient size to limit the error. Once the services and IP addresses are selected, the selected IP addresses must be tested for responsiveness to those selected services. This requires defining what responsive means in the context of each service.

One unusual, and perhaps unexpected behavior, is that some firewalls respond to services for IP addresses beyond them. The goal is to test if IP addresses are responsive to the service. If the firewall acts

Service	Use	Type	Protocol/Port
Chargen	Debugging	Historical	TCP/19
DNS	Name Lookup	Current	UDP/53
Echo	Debugging	Historical	TCP/7
Finger	Identification	Historical	TCP/79
FTP	File Transfer	Current	TCP/21
Gnutella	File Sharing	P2P	TCP/6346
HTTP	WWW	Current	TCP/80
Ident	Identification	Historical	TCP/113
Kazaa	File Sharing	P2P	TCP/1214
NetBIOS	File Server	Local	TCP/137
NNTP	USENET	Current	TCP/119
Ping	Measurement	Current	ICMP
POP3	E-mail	Current	TCP/110
SMTP	E-mail	Current	TCP/25
SSH	Terminal	Current	TCP/22
Telnet	Terminal	Current	TCP/23
UDP	Measurement	Current	UDP/33437

Table 7.1: Services selected to measure. "Historical" means that although the service is still in use, it is much less popular than it used to be.

on behalf of the IP address, the IP address appears responsive, even though the firewall does not allow the packet to reach the IP address. To remove these false positives, firewall responses must be detected and removed.

7.2.1 Service Selection

It is not possible to test all possible services. There are 65,536 TCP ports and 65,536 UDP ports, too many to test with any level of detail in a reasonable time. Even limiting the search to ports with published services, there are almost 2000 of those listed in FreeBSD's services file[79], still too many to test responsiveness of for a reasonable number of IP addresses. Thus, a cross-section of services were selected. The seventeen services selected to test are listed in Table 7.1. These services were selected for a variety of reasons:

1. Chargen (character generator)[75] is almost exclusively used for bad purposes. Security experts suggest blocking it and there are few reasons not to block it.

2. DNS (domain name service)[62] is required for most other services to be useful. It is also a UDP service, unlike most of the services selected.

3. Echo[76] is similar to Chargen, but not as useful for bad purposes. Like Chargen, there are few reasons not to block it.

4. Finger[107] is considered bad as it reveals user names. security experts suggest blocking it, but, unlike Chargen and Echo, it provides a useful service to end-users.

5. FTP (file transfer protocol)[78] is a popular service used primarily to publish files for public download. FTP was selected for its popularity.

6. Gnutella[40] is a peer-to-peer (P2P) file sharing service. Given the potential for legal problems due to some uses of P2P services, Gnutella was expected to be blocked more often than other services.

7. HTTP (hypertext transfer protocol)[38] is used to transfer web pages. For some users, the Internet is web pages. HTTP was selected for its popularity, as almost every companies runs at least one HTTP server.

8. Ident (identification)[48] identifies the local user owning a particular TCP port. The most common use is by FTP servers. Some FTP servers connect back to the client using the Ident service to learn the identity of the user attempting to access the server. If firewalls simply block the packets, users are forced to wait for these Ident connections to timeout before using such FTP servers. For this reason, its behavior was expected to be unusual.

9. Kazaa[50] is another peer-to-peer (P2P) file sharing service. It is included for the same reasons as Gnutella. Kazaa provides a service similar to Gnutella.

10. NetBIOS[42] underlies Microsoft Windows networking. It uses multiple ports. Port 137 is the naming service, selected because it was expected to be the least intrusive. Because Microsoft Windows networking is primarily used for local file sharing, remote access was expected to be blocked by most firewalls.

11. NNTP (network news transfer protocol)[49] is used to transport USENET articles. It was selected as an established service run by many companies, but less popular than FTP or HTTP.

12. Ping is a service popularly used for measurements and debugging. It is considered relatively low-risk to allow this to pass a firewall. It runs on ICMP[73], unlike any of the other services. For these reasons, its behavior was expected to be different.

13. POP3 (post office protocol v3)[66] is used by users to retrieve their e-mail. As the POP3 service that runs on port 110 is usually not encrypted, it is considered more of a security problem than alternatives, such as IMAPS and POP3S.

14. SMTP (simple mail transfer protocol)[53] is used to deliver electronic mail between mail servers. Like HTTP, almost every company operates at least one publicly-accessible mail server.

15. SSH (secure shell)[105] provides remote terminal access to a host. It provides encryption and key-based authentication. It was expected that firewalls will block most SSH accesses, forcing users to use a VPN or go through a small set of SSH servers to gain access to internal hosts.

16. Telnet[77], like SSH, provides remote terminal access. Although Telnet can be encrypted, it is usually not. For this reason, security experts suggest using SSH instead of Telnet. It was expected to be blocked more than SSH.

17. The UDP "service" is not a true service. If a UDP packet is sent to a non-listening port, most hosts will respond with an ICMP port unreachable message. This is commonly used by traceroute[46] and related technologies for measurement and debugging. Allowing traceroute past a firewall is considered relatively low-risk and is useful for debugging network problems.

One service conspicuously missing is SNMP[11]. SNMP was excluded for two reasons: SNMP is intrusive and an SNMP server should not respond to unauthorized packets. Because SNMP is a network management service, attempting access is often viewed as an attack and is therefore more intrusive. In addition, SNMP packets contain a community string, which acts somewhat like a password. If the community string does not match any community string configured to have access, the SNMP server does not respond to the request. In such cases, it is difficult to distinguish a firewall blocking the packets and not having a proper SNMP community string. Because measuring SNMP is difficult and intrusive, it was not tested.

7.2.2 Sample Selection

It is not reasonable to test all four billion IP addresses. Even limiting testing to allocated or routed IP addresses leaves too many IP addresses to test. Instead, a random set of IP addresses must be selected.

The primary goal of the measurements is to analyze the prevalence of various behaviors and how it varies by services. Thus, IP addresses were randomly selected from the entire IP address space. Large parts of the address space are not used. A local BGP feed indicates that less than 30% of the IP address space is globally routed. However, aggressively limiting the IP addresses to active address space is prone to error. Although IP addresses were selected from the IP address range, only IP addresses within /8 CIDR blocks indicated as allocated according to ARIN's reverse lookup databases were tested. All other IP addresses were considered to be non-responsive. ARIN lists 124 /8 blocks as in use, after excluding multicast and private address space. Although this is somewhat conservative, aggressive filtering would introduce inaccuracies without meaningfully decreasing the set of IP addresses to test.

The number of IP addresses selected determines the accuracy of the results and how well rare behaviors are measured. Because responsiveness is unusual, Poisson confidence intervals[2] give the error bounds. Consider a Poisson test that yields x positives. From statistics, the $1 - \alpha$ confidence interval for $E[x]$ is given by $(\frac{1}{2}\chi^2_{\frac{\alpha}{2}}(2x), \frac{1}{2}\chi^2_{1-\frac{\alpha}{2}}(2x+2))$, where χ^2 is the chi-square distribution. For 423 positive responses, the 95% confidence interval is approximately $(383.6, 465.3)$, an error of less than 10%. If at least 908,400 IP addresses on the Internet would respond positively to some test, then a sample of two million IP addresses is expected to contain 423 such IP addresses. Testing two million IP addresses means that the error should be less than 10% for any behavior exhibited by at least 908,400 IP addresses on the Internet.

Selecting IP addresses from the space uniformly does not sample firewall behaviors uniformly. However, this chapter is more interested in the sampling access behavior of IP addresses more than sampling blocking behavior of firewalls. This is exactly what is measured by selecting random IP addresses. Part of the reason for this focus is that firewalls may have different rules for different IP addresses. For example, a firewall may allow public access to the network with the web servers, at least for HTTP, but block HTTP access to the rest of the network. It is not clear what the behavior of the firewall would be considered is such a case. By sampling IP addresses, this ambiguity does not arise.

7.2.3 Testing Method

Most of the services selected to test run over TCP. To test a TCP service, a TCP SYN packet was sent to an IP address to test if that host was responsive to that service. Any TCP response, including a TCP RST (connection refused) was considered responsive. Any TCP response that had the SYN and ACK flags set was considered listening. A TCP RST packet was sent for each TCP packet received that was not itself a RST, telling the tested host to discard the connection attempt.

In the case of a listening host, no attempt was made to determine what service was actually running on the port, simply that some service was listening to the port. In addition, no testing was done on other ports to determine if the services was accessible or blocked above the network level. For example, a host using

TCP wrappers[100] accepts connections before checking the allow and deny lists. Such a host would be recorded as listening, despite the fact that the connection may be closed before any data is exchanged.

An alternative method to test responsiveness would be to send a TCP ACK or TCP SYN/ACK packet instead of the TCP SYN packet. For firewalls that do not maintain state but allow outbound connection, TCP ACK packets may be allowed through the firewall. For firewalls that do maintain state, such packets should be dropped, as there would be no associated connection. Thus, testing using a TCP ACK may detect stateful firewalls and may find more hosts. However, a TCP service is only accessible if a TCP connection can be established. It is not possible to establish a connection with a host if TCP ACK packets elicit responses but TCP SYN packets do not. Thus, using TCP SYN packets better measures accessibility.

Not all of the services selected are TCP. In particular, DNS and UDP use UDP, and Ping uses ICMP. These services were tested using different techniques.

The DNS service was tested by sending a PTR request for 1.0.0.127.in-addr.arpa. This is a request for the hostname of 127.0.0.1, the localhost address. Most DNS systems include this entry. Knowing this IP address's hostname has no obvious security implications, as IP packets sent to this IP address are destined for the host that sent them. Any IP address that responds with a DNS packet (including a DNS error response) was considered responsive and listening. Any IP address that responds with an ICMP port unreachable was considered responsive but not listening, since that is the behavior of an unprotected systems not listening to part 53.

The UDP service specifies its testing method. A UDP packet was sent to a port 33437, a "high-numbered" port used by traceroute. An IP address responding with an ICMP port unreachable message was considered responsive but not listening. An IP address cannot be considered listening to the UDP service.

Ping was tested by sending an ICMP echo request packet (type 8). An IP address responding with an ICMP echo reply (type 0) was considered responsive and listening. An IP address responding with an ICMP port unreachable was considered responsive but not listening[2].

Sending a packet to test for a service may also induce an ICMP error response. Common ICMP error responses include ICMP time exceeded, ICMP host unreachable, ICMP filtered, and ICMP network unreachable. These may be responses from firewalls or may be responses from other network elements. Such error messages mean the host did not respond because the packet did not reach its destination. Therefore, with the exception of ICMP port unreachable, all ICMP error responses were ignored in the analysis.

These techniques effectively attempt an inbound connection. Depending on the definition of "connected", inbound connection attempts cannot find all the hosts connected to the Internet. A host behind a firewall that blocks all incoming connections or some other device that obscure its existence, such as a network-address translation (NAT) box, would not be detected. It is not possible to use inbound connection attempts to determine which of the IP addresses beyond a firewall are active and which are not if the firewall blocks all incoming connections. It is not possible to determine how many hosts are beyond NAT boxes using inbound connection attempts. Techniques that look at inbound or outbound traffic[4] may be able to detect active hosts beyond such devices, but are still unable to detect hosts which do not access the Internet. Inbound connection attempts identify the hosts connected to the Internet that are publicly accessible to some degree, a subset of all hosts connected to the Internet.

Because it is impossible to detect a host behind a firewall that blocks all incoming connections using these methods, it is not possible to analyze what percentage of hosts are behind firewalls that block a given service. Thus, the analysis will focus on hosts either not behind a firewall or protected by a firewall that allows public access to at least one service on that host, regardless of whether or not the host runs that

[2]Although an ICMP port unreachable response to a Ping may seem impossible, a few IP addresses responded in this way.

service. Although this is equivalent to assuming that hosts which cannot be detected do not exist, it is unclear how this assumption could be reasonably avoided.

7.2.4 Firewall Responses

Some firewalls respond to packets when they drop them. Naïvely, such responses would be counted the same as responses from a host at that IP address. When a firewall responds, it does not indicate a host is at that IP address, much less whether or not that host is listening for the particular service being tested. Firewall responses can account for more than 30% of all responses and 78% of measured listeners. Unless firewall responses are detected and removed, they skew the results.

Firewalls may be configured to respond to packets for a variety of reasons. One example is to respond to Ident[48] TCP SYN packets with TCP RST. Some FTP servers still attempt to connect back using Ident to clients to determine the user connecting to them. If the user's firewall simply drops such packets, users must wait for the Ident connection to timeout. However, if the user's firewall responds with TCP RST packets, the connection immediately fails, avoiding the response delay.

To detect firewalls responses, pairs of IP addresses close to responding IP address were tested for being aliases of each other using IPid verification described in Section 3.4.1. This tests if the IP identification (IPid) field in the IP header[74] of two IP addresses are correlated. Most hosts use a common counter to set the IPid of response packets. If pairs of close IP addresses were tested and the IPid field values were correlated, then the responses were likely to be from the same host. This could be due to IP aliasing or a firewall intercepting the packets and sending responses itself. Routers and web servers are often given many IP aliases. Web servers most commonly have multiple IP addresses to support virtual hosting under HTTP 1.0 or to support multiple HTTPS servers. Generally, a system administrator will give such a web server adjacent IP addresses. Thus, the further two IP addresses are away, the less likely they are to be the same due to IP aliasing. Routers have an IP address for each interface. Routers are commonly given IP addresses at the beginning (.1) of the subnet in which they belong, although the end (.254 for a 24-bit network mask) is also common. To avoid falsely concluding that web server responses come from firewalls, the selected IP addresses should have large separation and multiple pairs of IP addresses should be tested. To avoid falsely concluding that responses from routers come from firewalls, the selected IP addresses should have different ending bit patterns and multiple pairs of IP address should be tested.

Five pairs of IP addresses were tested for each IP address. Each pair includes the original IP address and an IP address with the same three leading octets. Keeping the same three leading octets makes it likely that a firewall protecting the original IP address is also protecting the other. This does not detect firewalls protecting smaller networks, but these are believed to be less common and such firewalls skew the results much less than, say, a firewall protecting a /8 network. The alternative IP address in the five pairs differed from the original IP address by 7, 13, 27, 44, and 69 respectively.

7.2.5 Anonymous DNS Detection

To test for anonymous hostnames, first the hostnames must be collected. Unlike testing for the DNS service, hostnames were collected following the standard DNS hierarchy[62]. The hostnames were collected by sending PTR queries for the `in-addr.arpa` address corresponding to the IP address. The set of IP addresses tested for anonymous DNS is the same set of IP addresses tested for responsiveness, allowing for comparison between the two results.

There are two common ways to implement anonymous DNS: give the same dummy hostname, such as "unknown" or "nothing" for all IP address or derive the hostname from the IP address. To test for dummy hostnames, the first piece of the hostname (everything before the first dot) was compared against a set of known dummy names. The set was constructed by looking at first pieces that occurred in two or more host names and determining whether or not it is anonymous or not. The list of dummy first pieces constructed was: “nothing”, “walmart”, “unknown”, “no-dns-yet”, “unassigned”, “dummy”, “validip”, “UNKNOWN”, “internal”, “internalhost”, “unused”, “unk”, “unspecX”, “racon”, “host”, “UNUSED”, “Unknown”, “PLEASE”, “EARLY-REGISTRATION”, and “client”. For “unspecX”, the individual forms of it are not listed but “X” was replaced by the first two octets of the IP address (there were five such replacements). These could have been considered derivatives of the IP address, but would not have been labeled as such by the method described below. Thus, they were included as “dummy” instead.

To test for host names derived from IP addresses, the hostnames were compared against five ways to encode the IP address. The five tested ways to encode the IP address 10.45.127.11 in the hostname are[3]:

1. The hostname contains the entire IP address as a substring, either forwards or backwards. Examples include “10-45-127-11”, “Yahoo010045127011”, “dialup-10.45.127.11”, and “h10s45a127n11”.

2. The hostname contains only the last two or three octets of the IP address, in forward or reverse order. Examples include “ws127011”, “PA011127”, “011-127-045”, and “rdu11-127-45”.

3. The hostname contains the entire IP address or all but one octet in an unusual order. Examples include “11.Red-10-45-127”, “ca-avignon-3-247.w10-45”, “h11-10-45-127”, and “host10451270b”.

4. The hostname contains the IP address in hexadecimal, in either forward or reverse order. Examples include “p0a2d7f0b”, “0A2D7F0B”, and “0B.7F2D0A”.

5. The hostname contains the entire IP address in hexadecimal or all but one octet in an unusual order. Examples include “p2d7f0b”, “0b.7f.0a2d”, and “0a2d7f-11”.

7.3 Results

Table 7.2 shows the raw results. The raw columns give the number of responders and listeners found for each server using the testing method described in Section 7.2.3. The firewall responders count is the number of responding IP addresses that were detected as responses coming from a firewall, according to the firewall detection algorithm of Section 7.2.4. The firewall listening count is the number of those responses that were considered listening. The host responders and listeners columns give the raw numbers without the firewall responses. After this correction, firewall responders and listeners are treated the same as no response from that IP address.

7.3.1 Effect of Large Domains

Some of the counts of Table 7.2 are affected by large domains. The practice of a large domain can represent a large fraction of a particular behavior when that behavior is otherwise unusual. This happens with firewall listeners counts. One surprising result is the large number of firewalls which responded with listening replies to Ident, SMTP, FTP, and Telnet. Many of these IP addresses (1836 of 3103 for SMTP, 1312 of 2746 for FTP, 1974 of 2816 for Telnet, and 3893 of 4254 for Ident) are USA military networks in the .mil top level domain, which includes all of 55.0.0.0/8. Hand testing five IP addresses reported as firewall listeners for SMTP revealed that nearby IP addresses returned the same SMTP header, indicating that the requests are

[3]Example hostnames correspond to real hostnames, with the represented IP address changed

	Raw		Firewall		Host	
Service	Responders	Listeners	Responders	Listeners	Responders	Listeners
Chargen	26388	667	4632	124	21756	543
DNS	22080	3049	3025	264	19055	2785
Echo	26577	725	4833	139	21744	586
Finger	26444	880	4927	217	21517	663
FTP	33361	8513	6836	2746	26525	5767
Gnutella	28172	385	5555	118	22617	267
HTTP	27087	9024	4679	1212	22408	7812
Ident	40655	5423	12749	4254	27906	1169
Kazaa	28018	823	5229	214	22789	609
NetBIOS	16233	176	3646	38	12587	138
NNTP	31959	1904	8890	332	23069	1572
Ping	27399	27327	1080	1079	26319	26248
POP3	32792	4060	8953	691	23839	3369
SMTP	31730	8143	7408	3103	24322	5040
SSH	34465	4287	9059	972	25406	3315
Telnet	28697	8180	6717	2816	21980	5364
UDP	22306	0	1751	0	20555	0

Table 7.2: Summary of per-service results. Host counts exclude firewall responds.

processed by the same host. For FTP, ten IP addresses reported as firewall listeners that were manually tested all closed the connections without sending even a header (as if running TCP wrappers). Based on hand-testing IPid behavior, nearby IP addresses yield responses from the same host. Thus, a firewall is sending listening responses, either directly or by redirecting the requests to another host. The USA military also represents a large fraction of the NNTP firewall listeners: 59 of 332.

Fortunately, large domains primarily affect firewall responses. Table 7.3 shows domains (partially anonymized) that represented more than 10% of firewall response or host listening counts for some service. This chapter focuses primarily on host responses, where no single domain represents more than 10% of the responses. The largest effect by a single domain for host responses is UDP, where one domain represents 7.9% of the host responders. Thus, large domain effects do not greatly affect the results.

7.3.2 How Prevalent are Firewalls?

Of the 56,297 IP addresses that responded to some service, only 4,023 responded to all services. Thus, at least 93% of hosts attached to the Internet are behind a filtering device of some type. Because this excludes hosts behind firewalls that block all incoming connection attempts, the true percentage is even higher than 93%. Clearly, firewalls are an important consideration when designing protocols and developing models for the Internet

7.3.3 What do Firewalls Block?

Table 7.4 shows what percentage of the hosts that respond to at least one service respond to each of the services. The first two columns correspond to the last two columns of Table 7.2, except that they are

Domain	Service	Type	Cnt	Pct
A.mil	POP3	FW responder	4182	47%
A.mil	NNTP	FW responder	4103	46%
A.mil	SSH	FW responder	4176	46%
A.mil	Ident	FW responder	4022	32%
A.mil	FTP	FW responder	2055	30%
A.mil	Telnet	FW responder	1993	30%
A.mil	SMTP	FW responder	1960	27%
A.mil	Ident	Host listener	274	23%
B.com	NetBIOS	Host listener	32	23%
C.net	DNS	Host listener	449	16%
D.mil	NNTP	Host listener	241	15%
E.edu	NetBIOS	Host listener	20	15%
F.net	NetBIOS	Host listener	18	13%
B.com	Gnutella	Host listener	33	12%
C.net	HTTP	Host listener	852	11%
G.com	Kazaa	Host listener	61	10%

Table 7.3: Domains, partially anonymized, contributing more than 10% of the IP addresses exhibiting some behavior for a service (raw numbers and firewall listening numbers omitted for brevity and clarity).

expressed as percentages of hosts that respond to at least one service. The total servers column gives the 95% confidence interval for the number of globally-accessible servers running each service, based on the number of host listeners found.

Unexpectedly, Ident is the most responsive service[4]. That is, Ident is the service most rarely blocked by firewalls. This is most likely due to FTP servers attempting to use Ident to determine the user attaching to them. As mentioned above, if a firewall simply drops these packets, the user must wait for the FTP server to timeout, which can be thirty seconds or longer. To avoid this, firewalls must be configured either to allow such requests or to respond to the packets themselves[5].

The least responsive service is NetBIOS. That is, NetBIOS is the service most commonly blocked by firewalls. As NetBIOS is a network file service usually meant to be accessed only locally, remote access is rarely desirable. The estimated number of NetBIOS servers is likewise small.

7.3.4 How Much do Firewalls Block?

There are two basic firewall configuration philosophies: block by default and block by exception. One way to estimate how common these two configuration philosophies are is to look at how many services to which hosts responded. IP addresses beyond firewalls configured to block by default are responsive to few services. IP addresses beyond firewalls configured to block by exception are responsive to many services. Figure 7.5 shows the distribution of the number of services to which hosts responded. 31,927 of the tested IP addresses

[4]Technically, Ident is only the most responsive of the services tested. For brevity, this chapter will be written as if the seventeen tested services are the only services.

[5]Another option would be to allow Ident queries only from hosts with which FTP connections have been established. To the author's knowledge, no commercial firewall provides a simple way to do this.

Service	% of Hosts Responding	% of Hosts Listening	Total Servers (95% Confidence Interval)
Chargen	38.6%	1.0%	1,070,000 – 1,268,000
DNS	33.8%	4.9%	5,761,000 – 6,207,000
Echo	38.6%	1.0%	1,159,000 – 1,365,000
Finger	38.2%	1.2%	1,317,000 – 1,536,000
FTP	47.1%	10.2%	12,067,000 – 12,709,000
Gnutella	40.2%	0.5%	507,000 – 646,000
HTTP	39.8%	13.9%	16,406,000 – 17,152,000
Ident	49.6%	2.1%	2,369,000 – 2,659,000
Kazaa	40.5%	1.1%	1,206,000 – 1,416,000
NetBIOS	22.4%	0.2%	249,000 – 350,000
NNTP	41.0%	2.8%	3,211,000 – 3,547,000
Ping	46.8%	46.6%	55,687,000 – 57,053,000
POP3	42.3%	6.0%	6,993,000 – 7,483,000
SMTP	43.2%	9.0%	10,527,000 – 11,126,000
SSH	45.1%	5.9%	6,879,000 – 7,365,000
Telnet	39.0%	9.5%	11,213,000 – 11,832,000
UDP	36.5%	0.0%	0 – 8,000

Table 7.4: Percentage of hosts responsive to at least one service that respond and listen to each service, as well as the estimated total number of servers for each service.

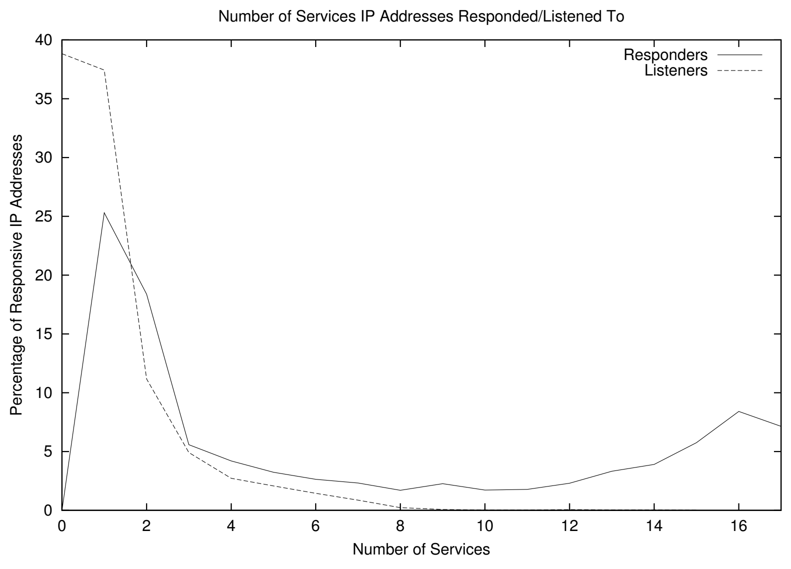


Figure 7.5: Distribution of the number of services IP addresses responded to and listened to.

are responsive to five or fewer services. Presuming that a firewall configured to block by default would allow no more than five services, 56.7% of IP addresses are beyond a firewall configured to block by default.

An alternative method to estimate the percentage of IP addresses behind a firewall configured to block by default is to look at a random port. The best examples of this are Gnutella and Kazaa, because they are newer and run on ports above 1024, which some operating systems treat specially. 31,707 of the tested IP addresses respond to neither Kazaa nor Gnutella. Presuming that a firewall configured to block by default would block both, 56.3% of the IP addresses are behind firewalls that block by default. These two methods agree that about 56% of IP addresses are behind a firewall configured to block by default.

The average number of services that an IP address responds to is 6.83. The median is four, lower than the average. The distribution is bimodal, with maximums at one and sixteen. The maximum at one represents hosts responsive to a single service. This is an implied statement by the administrator that only one service produces enough value for the security risk it represents, perhaps because the host only runs that service. The maximum at sixteen represents hosts that are accessible using all services except one. This represents a conclusion that only one service is worthy of blocking, a damning statement about that service. This could be either because the value of that service is low or the security risk for that service is high. Table 7.6 shows how often these two behaviors are observed for each service.

Service	Responded all but this		Responded only to this	
	Count	Percent	Count	Percent
Chargen	59	0.1%	224	0.4%
DNS	566	1.0%	1017	1.8%
Echo	24	0.0%	172	0.3%
Finger	42	0.1%	137	0.2%
FTP	88	0.2%	476	0.8%
Gnutella	33	0.1%	287	0.5%
HTTP	271	0.5%	1023	1.8%
Ident	42	0.1%	3488	6.2%
Kazaa	57	0.1%	436	0.8%
NetBIOS	2578	4.6%	175	0.3%
NNTP	28	0.0%	115	0.2%
Ping	364	0.6%	3906	6.9%
POP3	17	0.0%	236	0.4%
SMTP	116	0.2%	1258	2.2%
SSH	40	0.1%	287	0.5%
Telnet	227	0.4%	590	1.0%
UDP	183	0.3%	417	0.7%

Table 7.6: Number of IP addresses with unusual response behavior for each service. This table only refers to host responses, not firewall responses. Percentages are the fraction of hosts that respond to at least one service.

NetBIOS is, by far, the most common service to block if blocking only one service. Recall from Table 7.2 that NetBIOS is also the least common service to elicit responses. Part of the reason many IP addresses are responsive to every service except NetBIOS is that some large ISPs block only it. Such ISPs contribute a large part of the numbers: bbtec (921), Comcast (294), Road Runner (243), and AOL (153). Excluding these ISPs given still leaves 968 IP addresses for which only NetBIOS is blocked, 70% more than DNS, the second-most-common service to block if blocking only one service.

Ident and Ping are the two most common services to allow if allowing only one service. Neither of these have single domains representing a large fraction of the IP addresses responding only to it. They are allowed

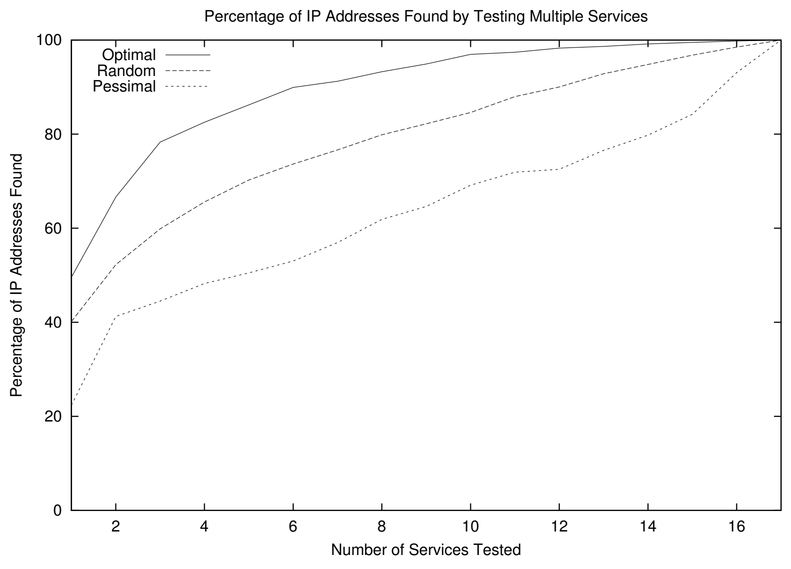


Figure 7.7: IP addresses found by using multiple services with optimal, random, and pessimal service selection.

for different reasons. Ping is allowed because it is useful for measurement and debugging and it has a low security risk. Ident is allowed because some FTP servers try to use it. Ident is unsurprising, because, as noted above, it is also the most common service allowed through a firewall. Ping, in contrast, is the sixth-most-common service allowed through a firewall, making it surprising that many IP addresses only respond to it.

Since most IP addresses are responsive to only a few services, it is difficult to determine what hosts are accessible from the Internet. On a large set of IP addresses, testing a large set of services is intrusive and time consuming. Thus, to determine which hosts are accessible from the Internet, as few services as possible should be tested. One idea is to use only one service. 27,906 of the tested IP addresses respond to Ident, more than any other service. This represents 49.6% of the IP addresses that responded to at least one service. Thus, testing only one service will, at best, find approximately half the IP addresses that would be found if many services were tested. Obtaining a more reasonable estimation, such as 90% of the accessible hosts, requires testing multiple services.

Figure 7.7 shows what fraction of responsive IP addresses would be found if a subset of services were tested. If services are selected optimally, only six services are necessary to obtain more than 90% of the IP addresses: DNS, FTP, Gnutella, HTTP, Ident, and Ping. This set of services is interesting for two reasons. Firstly, DNS elicits the second-smallest number of responses, yet it is included. Secondly, NNTP, POP3, and SSH are excluded, despite the fact that they elicit more responses than DNS, Gnutella, and HTTP. This implies that behaviors between services are correlated. This will be discussed in more detail below. If services are selected randomly, the number of services required to obtain at least 90% of the IP addresses jumps to 12. Even services are selected poorly, then sixteen services (all but one) must be selected. Random selection and poor selection perform badly because a few services have many IP addresses that respond exclusively to that service.

Service	Letter	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
Chargen	A	x	34	89	77	69	77	41	61	74	33	78	40	68	60	69	72	48
DNS	B	42	x	43	46	36	44	45	53	44	26	45	40	54	45	46	36	63
Echo	C	89	35	x	80	72	81	42	63	76	34	81	42	69	61	71	69	50
Finger	D	79	38	81	x	76	75	46	69	74	36	89	51	75	67	80	63	59
FTP	E	49	20	51	53	x	48	34	35	49	20	62	21	50	49	66	49	35
Gnutella	F	72	33	76	69	63	x	39	53	87	30	70	39	59	52	61	61	49
HTTP	G	39	35	40	43	46	40	x	51	39	27	45	34	58	53	45	40	32
Ident	H	39	28	40	44	31	36	34	x	36	25	42	26	51	42	42	30	27
Kazaa	I	68	33	71	68	64	86	38	51	x	29	70	39	59	52	60	58	49
NetBIOS	J	71	46	73	77	62	71	61	84	69	x	75	47	87	76	75	53	48
NNTP	K	71	33	73	80	79	68	43	60	69	31	x	40	70	67	76	63	53
Ping	L	29	23	30	36	21	30	25	29	30	16	32	x	29	24	31	23	37
POP3	M	58	37	59	63	61	54	52	68	55	34	66	35	x	78	69	48	42
SMTP	N	50	30	51	55	57	46	46	54	47	29	62	28	75	x	64	46	37
SSH	O	53	29	54	60	71	50	37	50	49	26	64	34	61	59	x	49	43
Telnet	P	70	29	68	60	69	64	41	45	61	24	68	31	55	54	63	x	45
UDP	Q	53	56	55	64	54	57	37	47	58	24	63	56	54	49	61	50	x

Table 7.8: Blocking correlation of IP addresses responding to two services, expressed as percentages. The upper-right value of 48 corresponds to $C_A^Q = 48\%$.

7.3.5 How is Firewall Behavior Correlated?

In order to quantify the correlation between firewall behavior, define the blocking correlation from one service to another to be how much more likely, relatively, an IP addresses is to not respond to service x given that it does not respond to service y. By formula, let R_x be the set of IP addresses responding to the service x and $R = \bigcup_x R_x$, the set of IP addresses that responded to at least one service. For firewall behavior correlation, what matters is the IP addresses for whom access to a service are blocked. Thus, let $\overline{R}_x = R/R_x$, the set of IP addresses that respond to at least one service, but not to service x. Then, the blocking correlation of a with b, C_a^b, is given by:

$$C_a^b = \frac{P[x \in \overline{R}_a | x \in \overline{R}_b] - P[x \in \overline{R}_a]}{1 - P[x \in \overline{R}_a]} = \frac{\|R\| \|\overline{R}_a \cup \overline{R}_b\| - \|\overline{R}_a\| \|\overline{R}_b\|}{\|\overline{R}_b\| \|R_a\|}$$

A negative C_a^b would mean that it is more likely to respond to a if it does not respond to b. This definition of correlation is closely related the correlation coefficient. In particular, the correlation coefficient of responsiveness to two services a and b is $\sqrt{C_a^b C_b^a}$. Table 7.8 shows the correlation between services. The bottom quartile is between 16% and 39%. The top quartile is between 66% and 89%.

High correlations reveal which services administrators consider equal, in terms of balance between security risk and usefulness to remote users. The highest correlation is 89%. Chargen is 89% correlated with Echo and Echo 89% correlated with Chargen. These services are often lumped with Discard, Daytime, and Time under the term “tiny services,” as they are simplistic services. Less correlation was expected, with Chargen blocked more often than Echo. However, this is not the behavior: most IP addresses that respond to one service respond to the other. Surprisingly, Finger is 89% correlated with NNTP, while NNTP is 80% correlated with Finger. That means that most IP addresses that respond to NNTP will also respond to

Finger (although they may not run either). Both NNTP and Finger are older, somewhat less popular services primarily run on UNIX-like operating systems, which may the cause of the high correlations.

Low correlations reflect administrators having dissimilar opinions about the security risk or (legitimate) usefulness to remote users. The lowest correlations are with NetBIOS. For example, Ping is 16% correlated with NetBIOS, the lowest correlation. This means that knowing that Ping was blocked to an IP address did not make it much more likely that NetBIOS was blocked. Because few IP addresses were responsive to NetBIOS, the likely reason that the other services have low correlations with NetBIOS is that NetBIOS is one of the first services blocked when configuring a firewall. In contrast, Ping is considered a relatively safe protocol to allow. Ignoring NetBIOS, the lowest correlations are between FTP and DNS and between FTP and Ping, at 20% and 21% respectively. This is partially a result of neither Ping nor DNS using TCP, which FTP and most of the services tested use. Indeed, many services have low correlations with Ping and DNS. However, UDP does not use TCP either, and FTP is only 35% correlated with UDP. This can be partially explained by the fact that many firewalls having special configuration options for Ping and DNS packets. However, that does not explain the mystery of why FTP is in both pairs. Although FTP had many responses, Ident had more and SSH, SMTP, and POP3 all had only slightly fewer.

One surprise from the raw results is the small number of responses to DNS. One theory of why DNS rarely induces responses is that it uses UDP for transport (at least, as commonly used and as tested), while most of the services tested employ TCP. Since TCP is connection-based, it is far simpler for firewalls to understand TCP communications than UDP communications. As a result, UDP may be considered less secure than TCP. DNS's behavior is not a function of simply not using TCP, since Ping uses neither TCP nor UDP, and is the sixth-most responsive service. However, it may be a result of UDP itself. If the reason DNS is blocked is that it uses UDP, than the UDP service, which uses UDP, should be highly correlated with DNS. UDP is 56% correlated with DNS and DNS is 63% correlated with UDP. Although DNS is not as correlated with any other service, UDP is more heavily correlated with many other services, most notably Finger and NNTP. Thus, blocking UDP made it notably more likely to block DNS, the reverse is not true. Therefore, DNS is blocked for reasons beyond just being UDP.

These correlations imply a dependency, where it is more likely to block service x if service y is blocked. This can be represented by the blocking dependency digraph defined as follows: Let the set of nodes correspond to the set of services. Let there be an arc from x to y if $C_x^y \geq 75\%$. Thus, an arc implies that an IP address is 75% less likely to respond to service x if it does not respond to service y. Figure 7.9 shows the blocking dependency digraph, which has 30 arcs on twelve of the services (nodes without arcs are excluded).

The cliques[6] in the blocking dependency digraph show clusters of services that administrators have similar opinions of. There are five completely connected cliques in this digraph. The largest one is Echo, Chargen, and Finger. These are less-common services that primarily run on UNIX-like operating systems. The second clique is Finger and NNTP. Again, both run on UNIX-like operating systems, but USENET, which is based on NNTP, remains somewhat popular, although not as popular as it once was. Given the similarity of the reasons for these two cliques, it would be reasonable to expect Echo and Chargen to also be highly correlated with NNTP. Indeed, Echo and Chargen are highly correlated with NNTP, and NNTP is somewhat highly correlated with them (71% and 73%), but not enough to be included in the digraph. Another clique is Kazaa and Gnutella, which is unsurprising given their similarity. A somewhat more surprising clique is POP3 and SMTP, both e-mail protocols, albeit different in application: SMTP is used to send e-mails and POP3 is used to retrieve e-mails. This difference in applications means that SMTP is unauthenticated (generally) while POP3 requires authentication. Thus, one might expect that administrators would view their security requirements differently. However, most appear to not. The last clique is the most

[6]Clique is normally used when discussing undirected graphs. For a directed graph, define a clique to be a maximal set of vertices which induce a complete digraph. Trivial cliques of one node are ignored.

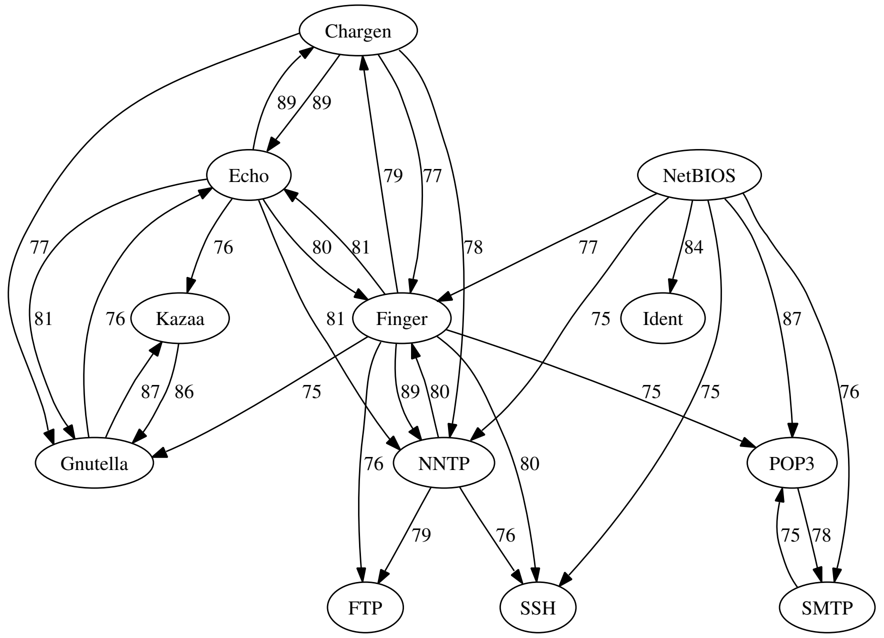


Figure 7.9: Blocking dependency digraph of firewall blocking. An arc service x to service y implies that $C_x^y \geq 75\%$.

puzzling: Gnutella and Echo. One might expect Kazaa to also be in this clique, given its close relationship with Gnutella, but Echo is only 71% correlated with Kazaa. The reasons for this clique are unclear.

Disconnected or weakly-connected nodes correspond to services with relatively unique profiles. Five services are missing from the digraph: DNS, HTTP, Ping, Telnet, and UDP. Three additional services have no outbound arcs: FTP, Ident, and SSH. These represent services that are blocked relatively independently from other services. Ignoring UDP, which has no listeners, when services are ranked by the number of listening IP addresses, these services are seven of the top ten, including the top four. Thus, part of their low correlation comes from the fact that firewalls must be configured to not block access from the Internet to public services running on hosts.

7.3.6 What Accessible Services do Hosts Run?

Table 7.4 shows the percentage of IP addresses that run each service as well as the estimated total number of servers. Not surprisingly, Ping was the most common listening service, as it is the only service which the operating system normally does the listening. Excluding Ping, HTTP, FTP, Telnet, and SMTP are the four next-most common. HTTP, FTP, and SMTP are the three basic services of the Internet: file transfer, e-mail transfer, and web servers. DNS was only the ninth most-common service, despite the fact that DNS is required for almost every other service to properly work. The fourth most common service, Telnet, is vulnerable to password sniffer when used with encryption, which is the common implementation. It is surprising that Telnet is still in large use, given the availability of a secure alternative with more functionality, SSH.

The observant reader may wonder why the number of HTTP servers differs greatly from Netcraft's numbers[55]. This chapter estimates 16-17 million HTTP servers on the Internet, while Netcraft's number is 50 million. Netcraft's number corresponds to the number of websites, not the number of HTTP servers.

Service	Letter	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
Chargen	A	x	1	76	34	3	54	2	10	19	63	10	0	5	3	5	3	0
DNS	B	7	x	14	0	6	-3	7	1	-4	-3	-1	3	8	6	6	1	0
Echo	C	82	3	x	27	3	49	2	11	20	51	8	0	4	3	4	3	0
Finger	D	42	0	30	x	4	53	2	13	19	61	14	0	8	4	7	5	0
FTP	E	38	12	33	36	x	36	33	30	10	51	81	0	52	54	61	42	0
Gnutella	F	26	0	22	21	2	x	2	11	21	63	9	0	4	2	4	1	0
HTTP	G	36	22	36	24	47	49	x	18	34	72	33	4	57	51	42	21	0
Ident	H	21	0	21	23	6	48	2	x	19	62	13	0	9	6	12	5	0
Kazaa	I	22	0	21	18	1	49	2	10	x	46	9	0	4	2	4	1	0
NetBIOS	J	16	0	12	13	1	32	1	7	10	x	5	0	3	2	3	1	0
NNTP	K	29	0	22	35	21	52	6	18	23	61	x	-1	21	24	30	18	0
Ping	L	18	49	33	11	-6	-15	23	-21	23	-32	-70	x	-13	-18	2	11	0
POP3	M	34	10	27	42	29	50	23	28	24	69	46	0	x	52	36	10	0
SMTP	N	32	12	29	30	47	43	31	29	15	75	81	-1	81	x	54	22	0
SSH	O	32	7	21	35	33	58	16	34	20	74	65	0	35	34	x	24	0
Telnet	P	27	2	29	47	39	32	14	25	10	30	66	1	16	23	40	x	0
UDP	Q	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	x

Table 7.10: Listening correlation of services, expressed as percentages. The second value on the first line, 1, corresponds to $LC_A^B = 1\%$.

A single HTTP server can host million of web sites. A single web site may be hosted on dozens of HTTP servers. The net effect is the number of web sites is much larger than the number of HTTP servers, at least on the Internet today.

7.3.7 How are Running Servers Correlated?

Although somewhat unrelated to the question of firewall behavior, another interesting question is how often two services run on the same host. This requires looking at the correlation of IP addresses listening to two service (the listening correlation). Table 7.10 shows the listening correlations of the services, where the listening correlation is defined akin to blocking correlation (let L_x be the set of IP addresses listening to a service):

$$LC_a^b = \frac{P[x \in L_a | x \in L_b] - P[x \in L_a]}{1 - P[x \in L_a]} = \frac{\|R\|\|L_a \cup L_b\| - \|L_a\|\|L_b\|}{\|L_b\|\|\overline{L}_a\|}$$

Note that an IP address is consider "listening" only if it is both listening and not blocked from the public Internet. A host may be running services that are not publicly available. Moreover, the host may be running local filters such as TCP wrappers that make the service unusable from the public Internet, even though hosts can connect to the service.

The bottom quartile of listening correlation is -70% to 1%. Recall that a negative listening correlation means that an IP address is less likely to listen to one service if it listens to the other. The top quartile is 32% to 82%. The listening correlations are lower than the blocking correlations, reflecting that an open firewall will make the IP addresses beyond it respond to many services, but running many services on a system is administratively difficult.

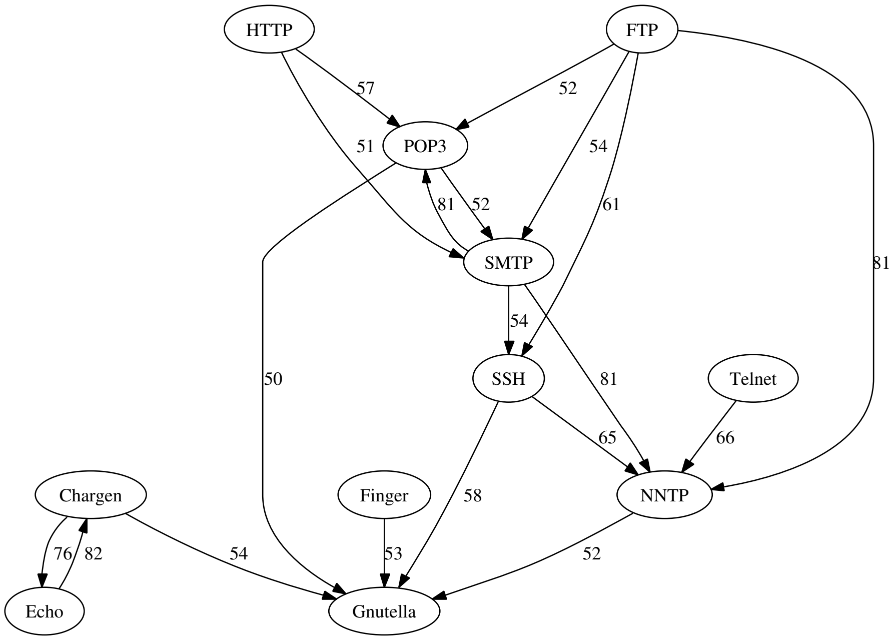


Figure 7.11: Dependency digraph of listening to services. An arc from service x to service y implies that $LC_x^y \geq 50\%$. That is, an IP address if IP addresses that listen to service y are at least 50% less likely to not run service x. NetBIOS was omitted for clarity. All displayed nodes, except Telnet, have arcs to NetBIOS (which has no out-arcs).

The highest correlations reveal interesting behavior. The largest correlation, 82%, is Echo with Chargen. Interestingly, Chargen is only 76% correlated with Echo. Recall that the blocking correlations were equal in both directions for these two services. Thus, firewalls are configured more similarly than hosts for these two services. The asymmetry of their listening correlations are likely a result of Chargen being more open to nefarious uses. Three pairs of services had listening correlations of 81%: FTP to NNTP, SMTP to NNTP, and SMTP to POP3. These three are highly asymmetric relationships: NNTP is 21% correlated with FTP and 24% correlated with SMTP, and POP3 is 52% correlated with SMTP. That is, most systems running NNTP servers also run FTP and SMTP servers, and most servers running POP3 run SMTP servers, but the reverse is not true. As any host running POP3 must receive mail somehow and that is commonly done over SMTP, the last pair is not unexpected. The relationships with NNTP are more surprising, but likely a result of the majority of NNTP servers running UNIX-like operating systems designed to easily handle running many services.

Listening correlations can also reveal which services are run by servers and which ones are run by end-systems. For example, correlations with Ping, run by every host, are all approximately 0, meaning running Ping gives little information about what services run on that host. However, listening to other services greatly *decreases* the likelihood of listening to Ping. In particular, Ping is -70% correlated with NNTP, implying NNTP servers are less likely to respond to Ping than non-servers.

Listening correlations are somewhat biased by the default configuration of operating systems. Because default installations and near-default installations are common, decisions by operating system distributors such as RedHat affect numerous hosts.

Like blocking correlation, a listening dependency digraph can be constructed to illustrate high correlations: Let each node in the digraph correspond to a service. Let an arc from x to y be included in the

Hostname Type	# IP Addresses	# Responding	Pct Responding
Based on IP address	94635	21189	22.4%
Dummy hostnames	3108	73	2.3%
Meaningful hostname	35753	10532	29.5%
Successful lookups	133496	31794	23.8%
Non-existent domain responses	735225	21107	2.8%
Lookup error	99761	2933	2.9%
All IPs	968482	55834	5.8%

Table 7.12: Prevalence of anonymous hostnames, non-existent hostnames, and DNS errors. The percentage is the percentage of IP addresses with that type of hostname that responded to at least one service.

digraph if $LC_x^y \geq 50\%$. Figure 7.11 shows the listening dependency digraph of the services tested. The graph contains 18 edges on ten nodes. NetBIOS is excluded from the display for clarity. All of the displayed nodes, except Telnet, have an out-arc to NetBIOS. NetBIOS has no out-arcs.

The cliques in the listening dependency digraph correspond to services commonly running together on a host. The dependency digraph of listening has two cliques: POP3 and SMTP and Chargen and Echo. Both of these pairs were cliques in the blocking dependency as well. The smaller number of cliques here, as compared to the blocking dependency digraph, reflects administrators' tendency to place services on separate servers, whether that be for security reasons, platform-incompatibility, load distribution, or some other reason.

7.3.8 How Prevalent are Anonymous Hostnames?

Reverse queries were sent to determine the hostnames of the 968,482 IP addresses tested for responsiveness. 133,496 of these lookups were successful in finding a hostname. The resulting hostnames were examined to determined if they represented anonymous hostnames.

Table 7.12 shows how often anonymous DNS occurs in both the entire set of IP addresses tested and those IP addresses responding to at least one of the services tested. Hostnames not based on the IP address will be called "meaningful". Note that there are other ways to encode part of the IP address in the hostname that are not included herein. As such, this overestimates the number of meaningful hostnames. The majority of hostnames are anonymous. Of 133,496 successful lookups, 97,743 (73.2%) of the hostnames returned are based, in part, on the IP address or are dummy hostnames.

Accessibility from the Internet makes it only slightly more likely that a host has a valid hostname. A bare majority of the IP addresses accessible from the Internet: 31,794 (56.9%) of the 55,834 tested IP addresses that respond to at least one service have hostnames[7]. Excluding errors (most of which were server failure messages from a name server), the percentage rises to 60.1%. Even responsive IP addresses that do have hostnames often have anonymous hostnames: 21,262 (66.9%) of the IP addresses that have hostnames and respond to at least one service have hostnames that are not meaningful.

That said, it is much more likely that an IP address will respond if it has a hostname: 23.8% versus 5.8%. Having a meaningful hostname increases the probability further. However, IP addresses with meaningful hostnames were still only 29.5% likely to respond to at least one service.

[7]463 (0.8%) of the tested IP addresses that responded to at least one service responded with a different IP address than the one sent. For this section's purpose, such IP addresses were considered non-responsive, as it is unclear to what IP address the responsiveness should be assigned. Because this is a small fraction, this exclusion does not qualitatively effect the results.

It may seem surprising that many inactive IP addresses have valid hostnames. It is difficult to decisively conclude that the data show this, as not responding to any service does not necessarily mean that the IP address is not active. The IP address may be beyond a firewall that blocks all inbound connection attempts. In fact, one would expect that a network which employs split DNS is more likely to be protected from the Internet by a restrictive firewall.

The data do show that more aggressive filtering of the input set based on ARIN data would have skewed the responsiveness results. 141,969 of the DNS lookups fail with a non-existent domain response from ARIN, indicating that no DNS server is responsible for reverse queries on that IP address. Of these, 419 IP addresses were responsive. These IP addresses represent 58 different assigned network blocks; none of them are within unassigned space. It is not clear why these IP addresses do not have any DNS servers associated with their reverse lookup. Regardless, aggressive filtering based on the top level domain database for in-addr.arpa would reduce the number of IP addresses tested by only 14.7%, while causing testing to miss 0.7% of the responsive IP addresses.

7.4 Conclusions

This chapter presents a study of firewall configurations, firewall behaviors, and anonymous DNS on the Internet. It includes a methodology for such a study and surprising findings in the analysis.

Most hosts on the Internet are behind a firewall. Specifically, this study showed that more than 93% of the hosts on the Internet are behind a firewall or filtering device of some sort. Thus, almost every packet delivered on the Internet passes through at least one such device. This makes giving an exact definition of "connected to the Internet" difficult at best.

This difficulty is reflected in looking at how many services must be examined to identify a large fraction of the IP addresses "connected to the Internet". One of the goals of this study is to identify such IP addresses to compare against DNS census techniques. Although testing seventeen services was expected to be somewhat overkill, it takes surprisingly many services to find a large fraction of the IP addresses that were found by the seventeen services tested. If services were selected randomly, it would take twelve services to, with high probability, find at least 90% of the IP addresses. Even if services were selected optimally, six services must be tested. If more services were added to the seventeen tested, these numbers would only increase further. Any similar attempt to look for most of the IP addresses responsive from the Internet must test multiple services to obtain reasonable result.

One of the largest surprises of this study is the amount to which firewalls respond. Firewalls represent more than 20% of the listeners for nine of the seventeen services tested and more than 20% of the responders for eight of the seventeen services tested. Clearly, any similar study of IP address behavior would also need to detect and remove firewall responses to avoid large errors. This was somewhat expected for the Ident service, but large numbers of firewall responses are common to about half the services tested.

Filtering devices differ in the level and type of filtering they employ. Approximately 56% of IP addresses are protected by a firewall that blocks by default and 37% of IP addresses are behind a firewall that blocks by exception. In terms of security, blocking by exception is generally considered a worse policy, as it depends on knowing which services are bad. From a security standpoint, it is much better to assume by default that services are bad and only allow access to services on specific hosts where the service is useful enough to accept the security risk. The high percentage (37%) of IP addresses behind a block-by-exception firewall raises a security concern.

Another interesting item to look at is how firewall behavior is correlated between services. For some pairs of services, such as Finger and NNTP, firewall behavior is heavily correlated. High correlation between two services implies that the perceptions of the two services are similar. In some cases, correlation is not symmetric: blocking A makes it more likely to block B, but not vice-versa. For example, not responding to POP3 makes it 87% less likely that an IP address will respond to NetBIOS, but not responding to NetBIOS makes an IP address only 34% less likely to respond to to POP3. This behavior implies one of the services is perceived either as more risky or less valuable. In this case, NetBIOS is both riskier and less valuable than POP3, in terms of remote access.

Although not the primary motivation, the study naturally detects if hosts are listening to ports as well, if no filtering devices blocks the queries. 39% of hosts accessible from the Internet listen to no services, implying that a lot of hosts are accessible that likely do not need to be. It is surprising to find out that certain insecure services are still prevalent. For example, Telnet is much more popular than SSH. There are approximately eleven million Telnet servers and approximately seven million SSH servers.

The services an IP address listens also exhibit correlated behavior. Unlike responding, where many hosts responded to almost all the services tested, few hosts listened to more than three or four of the services. That does not mean, however, that there are not examples of highly correlated behavior. For example, most hosts than run either SMTP or POP3 run both. This relationship is also not always symmetric. For example, listening to NNTP makes an IP address 81% more likely to listen to FTP, while listening to FTP makes an IP address only 21% more likely to listen to FTP. When most servers running service A run some other service B, any security problem in service B will affect most A servers. In this case, if FTP servers could be compromised, then most NNTP servers are compromised as well. This leads to some unexpected security implications, where a security flaw in service B causes problems with A servers despite the fact that the services are not obviously related.

One of the goals of this study is to explore the accuracy of DNS methods of counting active IP addresses. At first, DNS methods look good, as an IP address with a hostname is more than eight times more likely to respond to an IP address without one. However, an IP address with an anonymous hostname is 24% less likely to respond than one with a meaningful name, and IP addresses with dummy hostnames are less likely to respond than those without a hostname. That said, even IP addresses which are responsive on the Internet, which should be more likely to have hostnames, do not have a hostname 43.1% of the time. Assuming that every hostname, even anonymous ones, corresponded to an actual host, DNS methods undercount the true number of hosts by at least 15%.

This study leads to a new area of Internet measurement – a security study of the Internet, including how security defense mechanisms are deployed and configured on the Internet. Such a study provides important information for vulnerability assessment and assists in designing new defense mechanisms. Hopefully, this study will encourage new research and efforts in security measurement. It is planned to continue to monitor how security-relevant behaviors on the Internet change over time.

Chapter 8

Conclusions

The Internet has grown from four hosts in late 1969 to hundreds of millions of hosts today. This growth in size has been accompanied with a corresponding growth in importance to both scientific work and economy activity. This thesis presented ways to measure and monitor today's Internet to better understand its behavior and the behavior of those on it.

One basic piece of information to know about a network is its topology. Without knowing the hosts on a network and how they are connected, it is difficult to place other information about the network in context. This thesis presented techniques used by an ongoing, long-term project to measure the topology of the Internet and how it changes over long time spans. Similar topology measurement techniques formed the basis of a venture capital-funded start-up that developed a product to measure the topologies of corporate networks, giving network administrators a snapshot of their topology. The combination of organic network growth and the decentralized nature of IP network means that some corporations have a poor or limited feel for their network's topology. The topology measurements can also provide insight on the effect of remote events. In the case of the 1999 NATO bombing of Yugoslavia, it could detect when network elements were disrupted. One problem with this technique is that it is difficult to determine which network elements should be monitored. This problem is addressed by geolocation techniques.

One of the problems with topology measurements is that a host may be represented by multiple nodes under different IP aliases of the host. This thesis presented ways to resolve these IP aliases by determining the IP addresses belonging to a single host. Previous methods and implementations were small-scale, designed to work on a few thousand IP addresses, and it was not known how effective they were. Algorithms using the same basic techniques of previous work, but designed to test hundreds of thousands IP addresses, were presented and compared against previous methods. These algorithms were then run on approximately 400,000 IP addresses for their performance and accuracy. Of the three techniques, the IPid technique performed best, but was expensive in terms of packets and time required. The UDP technique required few packets and ran quickly, but was only half as effective. The rate limit technique was almost as expensive as the IPid technique but performed worse than the UDP technique. However, each technique found IP aliasing not found by the other two techniques. Thus, applications with tight time constraints should use UDP, applications with looser time constraints should use UDP and IPid, and applications requiring high accuracy must use all three.

The Internet is, at its heart, a set of interconnected hosts that forward packets. Routing is the process of deciding how those hosts should forward packets. Routing is multilevel and compartmentalized, making global measurements difficult. The parallel traceroutes done to measure the Internet's topology give routing information for many destinations, but measure the routing decisions made for each destination at only a few routers. This gives an incoherent set of routing decisions that is difficult to analyze. This thesis proposed a new way to measure these routing decisions by attempting to determine the inbound paths to a particular destination. Although some techniques to measure inbound paths do exist, they yield minimal information. The proposed technique measures more than sixty times the number of routing decisions measured by previous techniques, at the cost of being less accurate.

One problem facing the Internet is denial-of-service attacks. Because packets are forwarded almost purely based on the destination, the true source of a packet need not be the source listed in the packet unless the source wants to receive replies. In an attack, the goal is to disrupt service, which, quite often, does not require receiving replies. Thus, the attacker can forge (“spoof”) the source of the attack packets. This thesis described a scheme to find the source of an attacking stream of packets. This controversial scheme uses some of the same denial-of-service techniques used by attackers. Although not ideal and not designed for multiple attack streams, which have become popular, it is designed for the current Internet. Other, more desirable techniques that require altering the Internet were presented, leading to much later work on defeating or locating denial-of-service attacks.

Another problem facing network operators is ensuring that their networks are operating properly. One aspect of this is monitoring link and end-to-end delays. This can obviously be done by placing measurement hosts next to each router, but that is expensive and difficult to maintain. This thesis described RCM, a delay measurement and path monitoring system that requires only a single measurement. RCM uses tunnels to create virtual measurement hosts next to routers. This means RCM is cheaper, faster, and easier to deploy and maintain. RCM uses a novel network tomography technique to determine link delays from the measured delays. RCM’s network tomography technique requires few probes per path, does not need to address internal routers, does not require hosts at every router, is able to model asymmetric networks, and can distinguish between propagation and queue delays. RCM is implemented and operational on an AT&T’s network. For that network, RCM determines link delays that match network configuration information without RCM knowing the information.

Firewalls and filtering devices have become common on the Internet. It is unclear exactly how prevalent filtering devices are or how many packet are filtered. This thesis presented a large-scale analysis of filtering. One surprising result is that at least 93% of the IP addresses are behind a filtering device of some sort, meaning that almost every network communication must pass through a filter. The exact level of filtering varies by service, with filtering of two different services being highly dependent, especially when those services are similar.

Firewalls and filtering devices have the effect of separating the Internet. The naming system is also often separated, with IP addresses resolving to different hostnames within and without local networks. This thesis presented the results from an examination of the hostnames of almost a million IP addresses. 73% of hostnames are determined from the IP address or are dummy names. Since IP-derived hostnames are commonly created for every IP address on a network without regard to whether or not there is a host at the IP address, hostnames are a poor indication of the existence of a host. Thus, the prevalence of IP-derived hostnames implies that host-count techniques based on DNS are inaccurate. Even if every IP address with a DNS hostname corresponded to an actual host, 15% of IP addresses which respond from the Internet do not have a hostname, and would therefore be incorrectly counted as inactive by DNS-based host-counting techniques.

This thesis presented ways to measure and monitor multiple aspects of today’s Internet. It includes analysis of the accuracy of the presented measurement techniques, as well as experimental results from those measurement system on the Internet. Being able to measure the current Internet means that the Internet can be better understood, including its limitations and deficiencies. Hopefully, this will lead to improved network design, network protocols, and network performance.

Bibliography

[1] M. Adler. Tradeoffs in probabilistic packet morking for IP traceback. In *Proceedings of 2002 Symposium on Theorem of Computing*, 2002.

[2] L. Bain and M. Engelhardt. *Introduction to Probability and Mathematical Statistics*. PWS-Kent Publishing Company, 1991.

[3] P. Barford, A. Bestavros, J. Byers, and M. Crovella. On the marginal utility of network topology measurements. In *ACM SIGCOMM Internet Measurement Workshop 2001*, Nov. 2001.

[4] S. Bellovin. A technique for counting NATted hosts. In *ACM SIGCOMM Internet Measurement Workshop 2002*, Nov. 2002.

[5] Y. Breitbart, C. Y. Chan, M. N. Garofalakis, R. Rastogi, and A. Silberschatz. Efficiently monitoring bandwidth and latency in IP networks. In *INFOCOM*, pages 933–942, 2001.

[6] H. Burch. Eliminating machine duplicity in traceroute-based internet topology measurements. Technical Report CMU-CS-02-146, CMU, May 2002.

[7] H. Burch and W. Cheswick. Mapping the Internet. *IEEE Computer*, 32(4), Apr 1999.

[8] H. Burch and W. Cheswick. Tracing anonymous packets to their source by selective denial-of-service probes. In *USENIX Lisa*, 2000.

[9] H. Burch and D. Song. A security study of the Internet: An analysis of firewall behavior and anonymous DNS. Technical Report CMU-CS-04-141, CMU, July 2004.

[10] Catrography. National Geographic, Jan 2000.

[11] J. D. Case, M. Fedor, M. L. Shoffstall, and C. Davin. Simple network management protocol. RFC 1157, May. 1990.

[12] CERT. Tcp syn flooding and ip spoofing attacks. CERT advisory CA-96.21, Sept 1996.

[13] CERT. smurf ip denial-of-service attacks. CERT advisory CA-98.01, Jan 1998.

[14] CERT. Results of the distributed-systems intruder tools workshop. The CERT Coordination Center, Dec 1999.

[15] B. Cheswick. The design of a secure Internet gateway. In *USENIX Summer Conference*, pages 233–237, Anaheim, California, 1990.

[16] W. Cheswick and H. Burch. http://research.lumeta.com/ches/map/movie.mpeg.

[17] W. Cheswick, H. Burch, and S. Branigan. Mapping and visualizing the Internet. In *Proceedings of 2000 USENIX Annual Technical Conference*, June 2000.

[18] W. Cheswick, H. Burch, S. Branigan, and F. Wojcik. Internet mapping project. http://research.-lumeta.com/ches/map/index.html.

[19] W. Cheswick, J. Nonnenmacher, C. Sahinalp, R. Sinha, and K. Varadhan. Modeling Internet topology. Technical Report 113410-991116-18TM, Lucent Technologies, 1999.

[20] Cisco Technology. http://www.internetindicators.com.

[21] K. Claffy. private communication.

[22] R. Clements. Who talks ICMP, too? – survey of 18 february 83. RFC 844, Feb. 1983.

[23] M. Coates and R. Nowak. Network tomography for internal delay estimation. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2001.

[24] J. Cohen. Drawing graphs to convery proximity: an incremental arrangement method. *ACM Transactions on Human-Computer Interaction*, 4(3):197–229, 1997.

[25] Cypergeography. http://www.cybergeography.org/.

[26] D. Dean, M. Franklin, and A. Stubblefield. An algebraic approach to IP traceback. *ACM Transactions of Information and System Security*, 5(2):119–137, May 2001.

[27] S. Deering and R. Hinden. Internet protoc,l version 6 (IPv6). RFC 2460, Dec. 1998.

[28] D. W. Dodds. August, 1974, survey of new-protocol telnet servers. RFC 701, Aug. 1974.

[29] D. W. Dodds. November, 1974, survey of new-protocol telnet servers. RFC 669, Dec. 1974.

[30] D. W. Dodds. September, 1974, survey of new-protocol telnet servers. RFC 702, Sep. 1974.

[31] D. W. Dodds. February, 1975, survey of new-protocol telnet servers. RFC 679, Feb. 1975.

[32] D. W. Dodds. July, 1975, survey of new-protocol telnet servers. RFC 703, Jul. 1975.

[33] T. Doeppner, P. Klein, and A. Koyfman. Using rotuer stamping to identify the source of IP packets. In *CCS '00*, 2000.

[34] N. Duffield. Simple network performance tomography. In *ACM SIGCOMM Internet Measurement Conference*, Oct 2003.

[35] P. Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.

[36] D. Farinacci, T. Li, S. Hanks, D. Meyer, and P. Traina. Generic routing encapsulation (GRE). RFC 2784, Mar. 2000.

[37] P. Ferguson and D. Senie. Network ingress filtering: Defeating denial of service attacks which employ ip source address spoofing. RFC 2667, Jan 1998.

[38] R. Fielding, J. Gettys, J. Mogul, H. Frystk, L. Masinter, P. Leach, and T. Berners-Lee. Hypetertext transfer protocol – HTTP/1.1. RFC 2616, Jun. 1999.

[39] A. Frick, A. Ludwig, and H. Mehldau. A fast adaptive layout algorithm for undirected graphs. In *Proceedings of Graph Drawing '94*, 1995.

[40] Gnutella. http://gnutella.wega.com.

[41] R. Govindan and H. Tangmunarunkit. Heuristics for Internet map discovery. In *INFOCOM (3)*, pages 1371–1380, 2000.

[42] N. W. G. in DARPA, Internet Activities Board, End-to End Services Task Force. Protocol standard for a NetBIOS service on a TCP/UDP transport: Concepts and methods. RFC 1001, Mar. 1987.

[43] J. Ioannidis and S. Bellovin. Implementing pushback: Router-based defense against DDoS attacks. In *Proceedings of Network and Distributed System Security Symposium*, 2002.

[44] IP Route Analytics, Packet Design. http://www.route-explorer.com/.

[45] ISC. ISC internet domain survey. http://www.isc.org/index.pl?/ops/ds/, 1998 - 2004.

[46] V. Jacobson. traceroute. ftp://ftp.ee.lbl.gov/traceroute.tar.Z, 1989.

[47] C. Jin, H. Wang, and K. Shin. Hop-count filtering: An effective defense against spoofed DDoS traffic. In *CCS '03*, 2003.

[48] M. S. Johns. Identification protocol. RFC 1413, Feb. 1993.

[49] B. Kantor and P. Lapsley. Network news transfer protocol. RFC 977, Feb. 1986.

[50] Kazaa. http://www.kazaa.com.

[51] T. Kernen. traceroute.org. http://www.traceroute.org.

[52] K. Keys. iffinder. http://www.caida.org/tools/measurement/iffinder/, 2000.

[53] J. Klensin. Simple mail transfer protocol. RFC 2821, Apr. 2001.

[54] P. D. Lebling. Survey of FTP mail and MLFL. RFC 751, Dec. 1978.

[55] N. LTD. Netcraft web server survey. http://www.netcraft.com.

[56] Lumeta Corporation. Internet mapping project. http://www.lumeta.com/mapping.html.

[57] Lumeta Corporation. http://www.lumeta.com/.

[58] B. Lyon. The opte project. http://www.opte.org/.

[59] D. McRobb, K. Claffy, and T. Monk. Skitter: CAIDA's macroscopic Internet topology discovery and tracking tool. http://www.caida.org/tools/skitter/, 1999.

[60] Merit Networks. Routing assets database. http://www.radb.net/.

[61] MIDS. http://www.mids.org/.

[62] P. Mockapetris. Domain names – implementation and specification. RFC 1035, Nov. 1987.

[63] J. Moy. OSPF version 2. RFC 2328, Apr. 1998.

[64] T. Munzner. H3viewer. http://graphics.stanford.edu/~munzner/h3/.

[65] T. Munzner. H3: Laying out large directed graphs in 3d hyperbolic space. In *Proc. of 1997 IEEE Symposium on Information Visualization*, pages 2–10, October 20–21 1997.

[66] J. Myers and M. Rose. Post office protocol - version 3. RFC 1939, May. 1996.

[67] Netsizer. http://www.netsizer.com, 2000.

[68] J. Pansiot and D. Grad. On routes and multicast trees in the Internet. *ACM Computer Communication Review*, 28(1):41–50, Jan. 1998.

[69] K. Park and H. Lee. On the effectiveness of probabilistic packet marking for IP traceback under denial of service attack. In *IEEE INFOCOM 2001*, pages 338–347, 2001.

[70] V. Paxson. End-to-end routing behavior in the Internet. *IEEE/ACM Transactions on Networking*, 5(5):601–615, 1997.

[71] Peacock Maps. http://www.peacockmaps.com.

[72] J. Postel. User datagram protocol. RFC 768, Aug. 1980.

[73] J. Postel. Internet control message protocol. RFC 792, Sept. 1981.

[74] J. Postel. Internet Protocol. RFC 791, Sept. 1981.

[75] J. Postel. Character generator protocol. RFC 864, May 1983.

[76] J. Postel. Echo protocol. RFC 862, May. 1983.

[77] J. Postel. Telnet protocol specification. RFC 854, May. 1983.

[78] J. Postel. File transfer protocol. RFC 959, Oct. 1985.

[79] T. F. Project. Freebsd v5.3. http://www.freebsd.org/, Nov. 2004.

[80] N. Provos and P. Honeyman. Scanssh - scanning the internet for ssh servers. UMich Technical Report CITI-TR-01-13, Oct. 2001.

[81] RIPE. Ripe region hostcount. http://www.ripe.net/ripencc/pub-services/stats/hostcount/index.html.

[82] E. Rosen and Y. Rekhter. BGP/MPLS VPNs. RFC 2547, Mar. 1999.

[83] RouteDynamics, Ipsum Networks. http://www.ipsumnetworks.com/.

[84] S. Russel and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, 1994.

[85] S. Savage, D. Wetherall, A. R. Karlin, and T. Anderson. Practical network support for IP traceback. In *SIGCOMM*, pages 295–306, 2000.

[86] A. Shaikh and A. Greenberg. OSPF monitoring: Architecture, design and deployment experience. In *USENIX Symposium on Networked System Design and Implementation*, Mar 2004.

[87] Y. Shavitt, X. Sun, A. Wool, and B. Yener. Computing the unmeasured: An algebraic approach to Internet mapping. Technical Report 2000-15, DIMACS, 23 2000.

[88] J. Shewchuck. An introduction to the conjugate gradient method without the agonizing pain. unpublished paper.

[89] M. F. Shih and A. O. Hero. Unicast inference of network link delay distributions from edge measurements. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, May 2001.

[90] D. Smallberg. Survey of smtp implementations. RFC 876, Sep. 1983.

[91] D. Song and A. Perrig. Advanced and authenticated marking schemes for IP traceback. In *IEEE INFOCOM 2001*, pages 878–886, 2001.

[92] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies with Rocketfuel. In *ACM SIGCOMM 2002*, Aug. 2002.

[93] R. Teixeira, K. Marzullo, S. Savage, and G. Voelker. In search of path diversity in isp networks. In *ACM SIGCOMM Internet Measurement Conference 2003*, Nov. 2003.

[94] I. G. Tollis, G. D. Battista, R. Tamassia, P. Eades, and G. D. Battista. *Graph Drawing*. Prentice-Hall, 1999.

[95] W. Townsley, A. Valencia, A. Rubens, G. Pall, G. Zorn, and B. Palter. Layer two tunneling protocol "L2TP". RFC 2661, Aug. 1999.

[96] L. Trefethen and D. B. III. *Numerical Linear Algebra*. SIAM, 1997.

[97] Y. Tsang, M. Yildiz, P. Barford, and R. Nowak. Network radar: Tomography from round trip time measurements. In *IMC'04*, pages 175–180, Oct. 2004.

[98] D. Tunkelange. Jiggle: Java interactive graph layout environment. In *Proceedings of Graph Drawing '98*, 1998.

[99] J. Vardi. Network tomography: Estimating source-destination traffic intensities from link data. *J. of the American Statistical Association*, pages 365–377, 1996.

[100] W. Venema. TCP wrapper: Network monitoring, access control, and booby traps. In *Proceedings of the 3rd UNIX Security Symposium*, pages 85–92, Sept. 1999.

[101] W. Wei, B. Wang, D. Towsley, and J. Kurose. Model-based identification of dominant congested links. In *ACM Internet Measurement Conference*, 2003.

[102] A. Westine, D. Smallberg, and J. Postel. Summary of smallberg surveys. RFC 847, Feb. 1983.

[103] Wired, Dec 1998.

[104] A. Yaar, A. Perrig, and D. Song. Pi: A path identification mechanism to defend against DDoS attacks. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, 2003.

[105] T. Ylönen. Secure login connections over the internet. In *Proceedings of 6th USENIX Security Symposium*, pages 37–42, Jul. 1996.

[106] B. Ziegler. Hacker tangles Panix web site. *Wall Street Journal*, September 12 1996.

[107] D. Zimmerman. The finger user information protocol. RFC 1288, Dec. 1991.