Improving End-to-End Availability Using Overlay Networks

by

David Godbe Andersen

S.M. Computer Science, Massachusetts Institute of Technology (2001)
B.S., Computer Science, University of Utah (1998)
B.S., Biology, University of Utah (1998)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2005

Author . . . .
Department of Electrical Engineering and Computer Science
December 22, 2004

Certified by . . . .
Hari Balakrishnan
Associate Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by . . . .
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Improving End-to-End Availability Using Overlay Networks

by

David Godbe Andersen

Submitted to the Department of Electrical Engineering and Computer Science
on December 22, 2004, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science and Engineering

Abstract

The end-to-end availability of Internet services is between two and three orders of magnitude worse than other important engineered systems, including the US airline system, the 911 emergency response system, and the US public telephone system. This dissertation explores three systems designed to mask Internet failures, and, through a study of three years of data collected on a 31-site testbed, why these failures happen and how effectively they can be masked.

A core aspect of many of the failures that interrupt end-to-end communication is that they fall outside the expected domain of well-behaved network failures. Many traditional techniques cope with link and router failures; as a result, the remaining failures are those caused by software and hardware bugs, misconfiguration, malice, or the inability of current routing systems to cope with persistent congestion. The effects of these failures are exacerbated because Internet services depend upon the proper functioning of many components—wide-area routing, access links, the domain name system, and the servers themselves—and a failure in any of them can prove disastrous to the proper functioning of the service.

This dissertation describes three complementary systems to increase Internet availability in the face of such failures. Each system builds upon the idea of an *overlay network*, a network created dynamically between a group of cooperating Internet hosts. The first two systems, *Resilient Overlay Networks* (RON) and *Multi-homed Overlay Networks* (MONET) determine whether the Internet path between two hosts is working on an end-to-end basis. Both systems exploit the considerable redundancy available in the underlying Internet to find failure-disjoint paths between nodes, and forward traffic along a working path. RON is able to avoid 50% of the Internet outages that interrupt communication between a small group of communicating nodes. MONET is more aggressive, combining an overlay network of Web proxies with explicitly engineered redundant links to the Internet to also mask client access link failures. Eighteen months of measurements from a six-site deployment of MONET show that it increases a client's ability to access working Web sites by nearly an order of magnitude.

Where RON and MONET combat accidental failures, the *Mayday* system guards against denial-of-service attacks by surrounding a vulnerable Internet server with a ring of filtering routers. Mayday then uses a set of overlay nodes to act as mediators between the service and its clients, permitting only properly authenticated traffic to reach the server.

Thesis Supervisor: Hari Balakrishnan
Title: Associate Professor of Computer Science and Engineering

To my parents, Mary Lou Godbe and Jerry Richard Andersen,
and to my grandfather Hampton Clawson Godbe,
who always encouraged me to discover things,
instilling in me the curiosity to become a scientist,
and the impatience to become a computer *scientist.*

Acknowledgments

I am deeply indebted to my advisor, Hari Balakrishnan, for making five years of graduate school one of the best periods of my life. For each effort I put into my research and this dissertation, I think Hari put two. I arrived at MIT quite without a clue, and rather worried that the admissions committee would soon realize their drastic error in inviting me. Fortunately, it was difficult to stray *too* far afield when striving to follow Hari's example and exemplary advice. He was a better advisor than I imagined possible.

Frans Kaashoek let me steal an enormous amount of his time and wisdom as I walked a line between networks and systems. Frans's vigorous approach both to research and to life helped show me the fun that can be had in academia. I benefited enormously from the time I spent visiting PDOS group meetings and getting an infusion of hard-core systems.

In addition to so generously serving on my committee, and on my typically short notice, John Guttag taught me many of the right questions to ask about research. I know of nobody better to turn to when wondering, "What's the really *important* thing about what I'm doing here?"

I have to thank Robert Morris for two things: First, as one of the people behind the RON project, for his early guidance and frequent advice; and second, for sharing his passion for all things timing-related and letting me frequently barge in and borrow a rubidium oscillator or frequency counter when I was avoiding my real work.

Alex Snoeren, my officemate for my first three years at MIT, showed me how to swim, both in research and socially, in a very different pool than the one from which I came. He was a great mentor, office-mate, and a damn smart blackboard off which to bounce my crazy idea or dumb question of the week. He's a great friend to have. Thank you, Alex. On the topic of swimming, I owe Christine Alvarado for similar help, both as an occasional training partner and teacher, and for successfully juggling her own Ph.D. research while encouraging those around her to occasionally flee the confines of their cubicles to meet other people.

Alex, Allen Miu, Stan Rost, and Jon Salz made our office in room 512 an excellent—and loud—place to work. Stan, thank you for the music, and all the rest. Allen, thank you for making me notice the badminton courts. I'll miss our daily interaction.

Nick Feamster tolerated me as both a friend, an officemate and an apartment-mate, and did an excellent job at all of the above. I may have gotten more research done while running and cycling with Nick than I did in the office. Where my research touched BGP, Nick was right there. We built a crazy measurement infrastructure, and hopefully, we'll keep on building it. On an unrelated note, I dedicate the eradication of the word "methodology" in this dissertation both to Nick and to Michael Walfish. If I ever manage to write half as well as Walfish, I'll count myself happy. My future sushi parties, however, must be dedicated to Jaeyeon Jung. Mythili Vutukuru and Sachin Katti also displayed impressive noise-tolerance during our short time of sharing an office.

I spent my time at MIT in the Networks and Mobile Systems group, whose members' diverse research interests always let me learn something new, whether about wireless networks, streaming databases, BGP routing, or whatnot. Brett Hull, Kyle Jamieson, Bodhi Priyantha, Eugene Shih, and Godfrey Tan all deserve blame for doing cool things and being willing to teach me about them. Rohit Rao helped write the second and much improved versions of the MONET proxy and parallel DNS server, thereby greatly improving my chances of graduating. Magda and Michel, I raise a Costco membership card in your honor. Dina Katabi provided much good feedback about nearly everything I wondered about.

Eddie Kohler and David Mazières showed me the true PDOS spirit, carried on by John Jannotti, Bryan Ford, Kevin Fu, Michael Kaminsky, Athicha Muthitacharoen, Emil Sit, Benjie Chen, Jinyang Li Max Krohn, and Jacob Strauss. Sameer and Mandi, thank you for many a Christmas party, and many a good conversation. Rodrigo and Chuck, don't stop debunking peer-to-peer. Thank you to Chris Laas for never letting me BS, and to Dan, John, Frank and Stribling for Halo. Doug Decouto, John Bicket and Sanjit Biswas, Daniel Aguayo, and Ben Chambers's Roofnet showed me that good work can (and should) be inspirational and extremely cool. Doug, good wind for your sails. Thomer Gil and Alex Yip provided inspiration in research *and* cycling.

Dorothy Curtis and Michel Goraczko put up with my constant requests for more disk space, my occasionally breaking the network, and the many outrageous requests I made at ridiculous hours. I placed a similar burden on the LCS infrastructure group; I give particular thanks to Garrett Wollmann and Mary Ann Ladd, who carved through the administrative bureaucracy to let me install DSL lines, collect routing data, and access dedicated Internet connections. Jay Lepreau, Mike Hibler, and Leigh Stoller and the rest of the Emulab crew provided invaluable assistance with managing and running the RON testbed. Thanks also to the many anonymous users at MIT who used and abused the proxy to provide us with data to analyze MONET.

MIT has an amazing collection from whom to learn, and a group of professors and research scientists who shared generously with their much-demanded time. David Karger did his best to teach me some theory, even if that was like trying to teach a pig to fly. David Gifford introduced me to the joys of machine learning through computational biology. John Wroclawski and Karen Sollins somehow always made time to talk to me about things network architecture and otherwise.

My research was supported by a Microsoft Research fellowship, DARPA, the National Science Foundation, and MIT. Many thanks to all for making this research possible.

My mother, Mary Lou Godbe, bravely and patiently proof-read many of the papers I wrote, and proof-read a draft version of this dissertation, steeping herself deeper in computer science jargon than I'm sure she ever imagined.

While I'm deliberately leaving out many good friends and family from this list (the list of academic people is too long already—and I fear it's drastically incomplete!), I do owe a heartfelt thank you to Naomi Kenner for supporting me during my job search and related trauma that temporarily turned me into a stressed-out basket case. Thank you, Ny. You rock. Many thanks to my friends from the outing, cycling, triathlon, and masters swimming clubs. You kept me comparatively sane for five years. Rob Jagnow was an excellent friend and adventure partner, who also provided a sterling example of how many responsibilities one can successfully juggle if one really tries. Between Rob, Hector Briceno, Luke Sosnowski, and Robert Zeithammer, I was never lacking for friends or inspiration to do crazy things involving the mountains.

My friends and family, named and unnamed, form the richest tapestry in which anyone could hope to live. You know who you are, and please know how much I appreciate you all.

Contents

List of Figures

List of Tables

System's got more bugs than a bait store.

- Tron (1982)

Chapter 1

Introduction

Despite the huge increases in the pervasiveness of the Internet in recent years, the Internet remains too unreliable to displace special-purpose networks for many critical applications. Studies suggest that Internet availability typically ranges between 95% and 99.6% (Table 1.1). When something important is on the line, we still pick up the telephone or use dedicated networks composed of leased lines. The telephone network is not general purpose, and dedicated networks are much more expensive than the Internet. These differences motivate this dissertation: Can we realize the *availability* of special purpose, dedicated networks over a more affordable, general-purpose Internet substrate? Doing so would lead to substantial cost savings and offer interoperability benefits to applications that migrate to the Internet.

Availability is the fraction of time that a service is reachable and correctly functioning relative to the time it is in operation. Systems upon which we have come to depend typically exhibit four "9's" of availability (99.99%) or greater. For instance, in 1993, the 911 emergency telephone service had core availability (not counting subscriber line failures) of 99.994% [127]. Even normal telephone service appears to have nearly 99.99% availability [128, 85, 45]. Few networked services can match the 1 in 161,000 accident rate (99.9993%) of carrier airlines in 2002 [165].

The availability of Internet-based systems, however, falls far short of the public telephone network and other special-purpose networks. Not only is the Internet an insufficiently available substrate for near-real-time critical services such as medical collaboration and some financial transactions, and for "real-time" services such as telephony and real-time process control, but unavailability, even on timescales of a few seconds, is also expensive for businesses that conduct transactions over the Internet (see, *e.g.*, Zona Research [129]). Even brief interruptions lasting more than a few seconds can degrade users' perception of a site's performance. Thus, Internet failures are deleterious to commerce and interfere with the wider adoption of the Internet for critical applications.

Most computer systems use redundancy and replication to improve availability. In the case of the Internet, many hardware systems, such as the routers that forward data packets along Internet paths, are engineered with redundant components; routing protocols are designed to compensate for link or router failures by establishing an alternate path through the network using different physical links; and services (such as large Web sites or databases) are often replicated across multiple machines. The Internet failures we observe today do not typically occur because of a lack of redundancy in the underlying Internet; as later chapters will demonstrate, there is often considerable spare capacity and many alternate paths. Rather, these failures occur for several different reasons:

Paxson 1997 [117]	Internet paths available 98.1–99.3%
Dahlin 2003 [36]	Web server availability 93.2–99.6%
Labovitz 1999 [89]	$<$ 35% of *routes* have 99.99% availability
Chapter 5	Average path experiences 30+ minute outages 0.1%–0.4% of the time.
Chapter 7	Web server availability $\leq$ 99.7%

Table 1.1: Comparison of studies of Internet availability. Route availability is a lower bound on end-to-end availability; if no route is present, a service *cannot* be available, but the presence of a route does not guarantee availability.

- First, Internet services depend upon the proper functioning of numerous components which we discuss below; it only takes one unmasked fault to interrupt this chain and reduce availability.
- Second, some component failures affect availability because the underlying routing systems take too long to react, disrupting connectivity.
- Finally, the component's failure may be of a type to which the underlying system was not designed to react, such as misconfiguration, malice, or traffic overload.

In this dissertation, we take an *end-to-end* view of Internet availability: At any given time, can two programs *effectively* communicate when they need to? We use this end-to-end measure of availability to design systems that can react more quickly to a wider range of failures than previous systems, often improving availability by an order of magnitude. An important theme in this dissertation is the development of simple, reusable failure-masking schemes that work in concert across a range of fallible components.

The approach we present is based on *overlay networks*, a network formed between hosts on the Internet that performs its own routing and forwarding using the Internet as a transport. Overlays provide a way to implement domain-specific routing; they can be used to quickly discover and mask failures in the underlying network; and they can be used to thwart attacks. Overlay networks work well for these purposes precisely because they take an end-to-end view of the underlying network.

The rest of this chapter examines the components involved in a typical Internet connection and the ways in which they fail; discusses the challenges facing end-to-end Internet availability; gives an overview of our approach to masking failures to improve availability; and discusses some of the key findings from this research.

1.1 End-to-End Service Availability

Internet services such as Web and FTP sites, email and news, and peer-to-peer systems all depend upon a number of underlying systems. In the context of this dissertation, we use *service* to refer to a particular function, such as providing the documents that make up a Web site. A *server* is an individual machine. A service may be replicated across multiple servers for redundancy. We often use *site* or *Web site* synonymously with service.[1] This section examines the paths taken by and processing performed on packets involved in a simple Internet service that responds with a text

[1] We term the function that Internet Service Providers (ISPs) perform—providing a link to the Internet—Internet *access*, not service.

```
mediaone-ma> telnet eep.lcs.mit.edu 26666

Trying 18.31.0.114...
Connected to eep.lcs.mit.edu.
Escape character is '^]'.

You have reached the service

Connection closed by foreign host.
```

Figure 1-1: A simple Internet service showing a connection from a client (`mediaone-ma`) to a server (`eep.lcs.mit.edu`) that returns a static text string to the client when contacted.

string, shown in Figure 1-1. In this example, the client host `mediaone-ma` attempts to contact a service running on the server host `eep.lcs.mit.edu`.

Figure 1-2 shows the packets that are exchanged between the client host, `mediaone-ma`, and other machines on the Internet. Before the client can reach the server named `eep.lcs.mit.edu`, it must translate that host name into an IP address such as 18.31.0.114. The client performs this translation using the Domain Name System (DNS) [100]. DNS resolution requires communicating with one or more servers maintained by different organizations. Once DNS resolution is complete, the client opens a reliable transport connection (a TCP connection) to the server and receives the data. In a larger service, the data transmission may require many packets.

These packets must travel over an Internet path similar to that shown in Figure 1-3. The interaction between all of these machines requires that several major components be functional for a connection to an Internet service to succeed:

1. Client local network and hardware
2. Client software
3. Client access link to the Internet
4. Wide-area Internet links and routing
5. DNS servers
6. Server access link to the Internet
7. Server software
8. Server local network and hardware

The client's connection to the Internet is a common source of both bandwidth bottlenecks and failures. Many smaller client networks are *singly-homed*, or only connected to the Internet through a single link. This lack of redundancy makes smaller clients particularly vulnerable to longer-term interruptions of Internet connectivity. Networks with multiple upstream Internet providers are said to be *multi-homed*. In order to use multiple connections, these multi-homed networks typically participate in Internet routing to choose between upstream links and to inform the rest of the network that they can be reached via several paths.

These components are often layered upon one another. The service model provided by the physical links and routers that make up the Internet is that of an unreliable datagram. A host sends a packet to a destination, and the network makes a best-effort attempt to deliver it. Hosts place a logical transport layer, such as TCP, on top of these unreliable datagrams to provide features such as connection

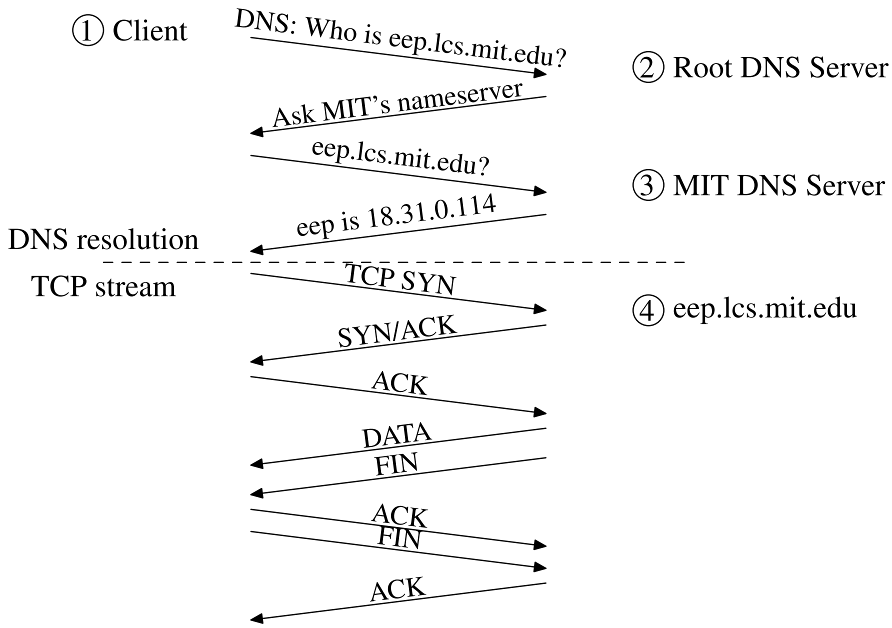


Figure 1-2: A simplified diagram of the packets exchanged to contact a trivial Internet service.

multiplexing, reliable in-order transmission, and flow control. Infrastructure services like DNS use these transport protocols to enable hosts to locate each other, and applications use transport-layer connections to communicate with each other.

The next subsections discuss three important aspects of the Internet: its logical organization, its routing protocols, and its domain name system.

1.1.1 The Organization of the Internet

The Internet is a collection of individual networks that share a common, underlying communications protocol: the Internet Protocol (IP). Each host on the Internet is assigned a numeric identifier (an IP address) that lets other hosts direct packets towards it. The Internet's routing substrate is organized

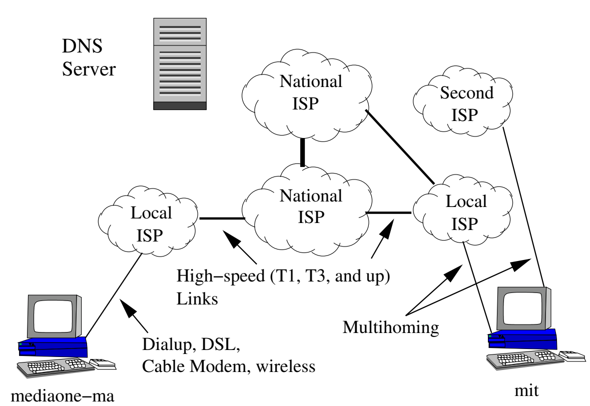


Figure 1-3: The network connections linking the example client and server.

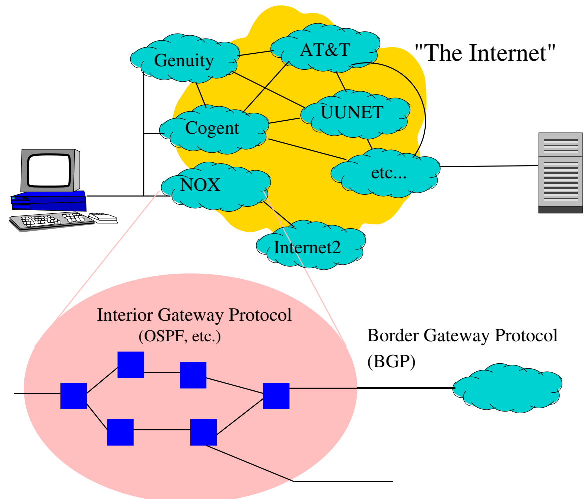


Figure 1-4: Routing domains in the Internet

as a number of different routing domains, each one an *Autonomous System*, or AS. An AS is an independently organized and operating network or collection of networks; examples of ASes include universities, large businesses, Internet Service Providers (ISPs), telephone companies, and so on.

Individuals and smaller companies usually attach to the Internet via a large national ISP, a telephone company, or an ISP local to their city or state. Home users in the US at the time of writing typically connect via modems (33.6 or 56 Kbits/s), Digital Subscriber Lines (DSL, 128 Kbits/s - 2 Mbits/s), or cable modems (128 Kbits/s - 3 Mbits/s). Corporations, universities, and service providers attach in a similar manner, though typically with higher speed T1 (1.5 Mbits/s), T3 (45 Mbits/s), or faster links. Figure 1-3 shows an example. Home users typically belong to the AS of their ISP; the ISPs themselves, along with larger companies, are each typically a separate AS.

Routing within an AS is performed by an *Interior Gateway Protocol* (IGP) such as Open Shortest Path First (OSPF) [103] or IS-IS [108]. Routing between ASes is handled by the Border Gateway Protocol (BGP), version 4 [126]. As we discuss below, there are significant differences in scalability, functionality, and failover times between the IGPs and BGP; an important focus later in this dissertation will be on masking inter-AS failures.

1.1.2 Routing

Routing attempts to ensure reachability between two IP addresses across the network; as such, it is a network's primary defense against link failures. Routing protocols propagate destination reachability information throughout the network, which tells routers where to forward packets so that they make progress toward their intended destination. When a router detects that a link has failed, it updates its routing messages accordingly, and after a period of *routing convergence*, it and all other routers will have established new routes that avoid the failed link if such a route exists and is usable.

Using BGP, ASes send out route announcements declaring that a particular network can be reached through them. These announcements propagate through the network. Eventually, by receiving these messages, the participating ASes know how to reach the destination networks either directly or by routing through a sequence of other ASes.

By treating vast collections of subnetworks as a single entity for global routing purposes, BGP is able to summarize and aggregate enormous amounts of routing information into a format that scales to hundreds of millions of hosts. Instead of providing detailed network topology information about where in its network all of its customers are located, a large ISP instead sends out a few small routing announcements that each point the way to hundreds or thousands of individual addresses. While this aggregation is necessary for the scalability of the global Internet, it means that routers propagate updates only for the entire aggregate—they do not announce failures that affect only a subset of hosts or networks within the aggregate. As a result, other parts of the routing system are unaware of many failures, and cannot attempt to react to them.

1.1.3 The Domain Name System

Internet hosts are reached by a human-unfriendly numeric IP address that is typically expressed in "dotted quad" notation, such as `18.31.0.114`. The DNS provides a lookup service that translates human-friendly, hierarchical names such as `eep.lcs.mit.edu` or `www.lcs.mit.edu` into IP addresses.

DNS resolution requires successively contacting one or more DNS servers to obtain parts of the name mapping. The base of name resolution is the root, or "." domain. To contact the host `eep.lcs.mit.edu`, the client software sends a request to a recursive resolver, which is willing to do the rest of the work for it. The resolver first contacts the root DNS servers (the addresses of which are provided along with the software), and is given a set of servers that can provide authoritative answers for the `edu` domain. The resolver contacts these servers and receives a delegation to the servers for `mit.edu` domain, and so on, until it reaches a server that can provide the address for the desired name. DNS servers rely on caching of both the final answer and the intermediate delegations to reduce both the load on the servers and the time it takes to resolve names.

DNS servers are easily replicated, and the system tends to be robust when implemented in accordance with the current best practices for redundancy and failure isolation [44]. These practices, however, are often not followed [111], either intentionally, or through subtle configuration errors. Partial DNS failures tend to cause client delays of several seconds as the client resolver searches for a working nameserver.

1.2 Challenges to Availability

In contrast to the high availability displayed by the telephone network, end-to-end Internet availability is generally in the 95-99.6% range [117, 36, 9, 89]. As Gray noted while examining a transaction processing system in 1985, "99.6% availability 'sounds' wonderful, but hospital patients, steel mills, and electronic mail users do not share this view" [62]. This section examines several factors that contribute to this reduced availability: Link failures, unexpected failure modes, performance failures, complex interdependencies between components, and malicious attacks.

1.2.1 Link Failures

A variety of existing mechanisms mask link failures. In areas of dense connectivity with redundant physical links, techniques like SONET[2] can recover from fiber-optic cable failures in under 50

[2] Synchronous Optical Network; a link-layer technology commonly used for high-speed optical links. Chapter 2 discusses SONET and other technologies in more detail.

milliseconds [17]. These techniques are often used to provide service within a metropolitan area, but are too expensive for use in the wide-area. Within an ISP, an IGP typically requires six to eight seconds to recover from link failure [69]. With tuning, this IGP recovery could be reduced to a few hundred milliseconds [6]. With recovery times this short, why do link failures remain as a cause of reduced availability? The primary reasons are an inability to find a backup link and the convergence delays within BGP.

The inability to find a backup link comes either from a lack of redundancy (*e.g.*, as is common for the access link to small customers who cannot afford redundant physical links), or because the links that are there could not be used. These links may be unavailable for two reasons: First, latent errors in the router configurations that mark links as backup or primary links may linger in an operational system until a failure brings them to light. Second, the links may be part of a shared physical infrastructure. Several catastrophic examples of this latter problem have occurred in recent years. For example, the 2001 train derailment in the Howard Street Tunnel in Baltimore, Maryland, impaired Internet service for at least four major U.S. backbone carriers, all of which used fiber optic cables running through the same physical location [42].

Even when the physical links are available, failures of links between providers or other failures that affect inter-provider routing still reduce availability. BGP can take up to several minutes to converge to a new valid route after a link failure causes an outage [86]; and, as we noted earlier, route aggregation can prevent many failures from even being reflected in BGP. We observed in another study that interruptions of end-to-end connectivity typically precede the associated BGP routing updates by at least three minutes [51].

1.2.2 Unexpected Failure Modes

Routing protocols are designed to cope with fail-stop failures (link disconnection, peer router crashes, etc.), and can generally do so, given sufficient convergence time. These protocols were not intended to cope with the myriad other failures that affect today's Internet—operator error and misconfiguration, software, hardware, and protocol bugs.

Operator error is a particularly severe cause of service unavailability [107], and the outages it causes tend to last longer than other outages. Operator error also plays a part in Internet outages. According to a study by Mahajan *et al.*, router "configuration errors are pervasive" [92], and misconfiguration appears to cause erroneous updates to 0.1-1.0% of the BGP table each day. Four percent of these misconfigurations were serious enough to cause connectivity problems.

Operator error also played a large part in several Internet-wide failures during the 1990s. One of the most well-known was the "AS7007" incident, in which a small ISP accidentally announced routes for the entire Internet—and most of the rest of the network accepted these announcements. The resulting attempt to send all of the traffic on the Internet through one completely overloaded router interrupted, for several hours, Internet connectivity to hundreds of networks [48]. Although advances in route filtering have reduced the severity of these incidents, these sorts of routing-propagated errors still occur with unacceptable frequency, in part because router configuration remains at a primitive level [50, 23].

Hardware or software failures can interrupt *data* without interrupting the flow of routing information. The routing system believes that all is well, but users' packets are unable to get through. One particularly bad incident of this type occurred in early 2004 with Level3 communications, a major ISP. The routing plane of one of their routers functioned normally, but the data forwarding plane degraded under excessive load and was unable to forward packets, causing an outage that affected thousands of sites in the US and Europe [135]. While tracking down the failures observed by our failure-masking systems, we observed numerous incidents of this sort, in which a small subset of packets (those with a particular source-destination pair, or that used certain port numbers) were unable to get through, while other traffic was unaffected.

1.2.3 Performance Failures

IP-layer protocols like BGP cannot detect problems that greatly degrade end-to-end performance, such as packet floods and persistent congestion. As long as a link is deemed "live" (*i.e.*, the BGP session is still alive), BGP will continue to advertise the faulty path; unfortunately, such a path may not provide adequate performance for an application using it.

These problems can arise from short-term problems (*e.g.*, flash crowds, in which a site experiences a huge, transient increase in traffic), or from long-term shifts in traffic patterns or the rise of new applications such as the Web or peer-to-peer file-sharing applications. At present, network operators adapt to these changes via traffic engineering (to balance the traffic load more effectively across their network links) and through network upgrades and provisioning. Unfortunately, the timescales on which these changes take effect leave long windows in which some links or routers in the network perform badly.

1.2.4 A Complex System of Fallible Components

As was explained in Section 1.1, the success of end-to-end communication depends on numerous working links, routers, DNS servers and client and server software. Each of these systems is vulnerable to its own set of failures. For example, recent studies have noted that "DNS configuration errors are widespread" [111], often reducing server redundancy or even making names unresolvable. Server failures, software and hardware bugs, and operator error still impair availability, even when the services are heavily replicated [107].

Today's Internet consists of about 18,000 different autonomous systems [146]. The overall system complexity of Internet routing and packet forwarding can lead to routing system failures. This problem was graphically illustrated in the mid-1990s, when it was found that a number of routers would either drop peering sessions or reboot when their BGP routing load grew too large. These session resets then increased the routing load on other routers in the network, leading to cascading—and possibly long-lasting—network instability. Even a few years after these problems were solved, up to 99% of the routing traffic on the Internet consisted of redundant updates that did not directly reflect changes in routing policy or topology [88]. These excessive updates were caused by a combination of router faults and the normal process of BGP exploring routes.

These cascading failures inspired router vendors to add *route flap damping*, which suppresses excessive routing updates on a per-prefix basis [167]. Emphasizing the complex interactions that can occur in a system like the Internet, this stabilizing mechanism is one *cause* of the delayed routing convergence we discussed above [93], because it can prematurely suppress routing updates during the BGP path exploration process.

1.2.5 Malicious Attacks

Distributed Denial of Service (DDoS) attacks and other malicious traffic are responsible for an increasing number of outages. In 2001, Moore *et al.* observed between three and five thousand DDoS attacks per week [102], a number that only counts certain types of attacks and probably undercounts the total number of attacks that actually occurred. More recently, large *botnets* have emerged, each a centrally-controlled network of thousands of compromised "zombie" hosts that can be directed to attack a particular target. In 2001, DDoS attacks could involve upwards of half a million packets per second, and a modern botnet can easily generate a gigabit per second of attack traffic. Even ignoring the effects on the victims, the collateral damage from such large attacks can be severe.

While technical measures have been developed to prevent [52, 112, 72] and trace [149, 38, 141] some DDoS attacks, most of these measures require widespread adoption to be successful. Unfortunately, even the simplest of these measures, filtering to prevent IP address spoofing, is not globally deployed despite years of advocacy, and DDoS attacks continue to be launched with impunity. Alarmingly, while the first DoS attacks were often launched in response to personal grudges or just "to see if they work," more and more attacks are being launched as part of extortion attempts against commercial Web sites [97]. It is likely that the severity and frequency of DDoS attacks will only increase.

In addition to brute-force DDoS attacks, several other security problems can interrupt Internet traffic. Worm programs frequently saturate Internet links, or even cause routers to crash, often by accident [27]. In some cases, the problems generated by worms have led to increased routing load [170]. Routers themselves are vulnerable to DoS attacks and/or compromise. Internet backbone routing supports little in the way of authentication, making it relatively straightforward to "hijack" network blocks and prevent traffic from reaching its intended destination [159]. The security of routing also opens other systems to attack, e.g., by hijacking the traffic destined for a name server or the root name servers [171].

1.3 Coping With Failures and Degradation

This dissertation continues a long tradition of creating reliable systems from unreliable components. The majority of problems affecting availability on modern networks represent faults that are not easily solved by improving a single component on the client-server path. The insight upon which this research builds is that the Internet is another component that can act as a building block for a redundant, highly available system. By treating an Internet path as the unit of failure masking, software systems running on end hosts can use redundant Internet paths to mask failures that interrupt end-to-end connectivity, regardless of the cause of those failures.

To perform such masking, Internet endpoints must be able to determine whether or not a path is working properly and choose between a set of paths that fail independently of each other. Important challenges addressed in this dissertation include how to obtain and make use of such redundant paths, how to measure their performance, and how to ensure that the resulting system performs with acceptable overhead. The systems presented in this dissertation address these issues with respect to two different workloads. The first workload is that of "community" communication: A small group of cooperating nodes that frequently send traffic between each other. The second workload is a Web workload, in which clients connect to a large set of external servers.

Masking failures on the Internet remains very empirical: at present, we lack good analytical models of the components affected by failures and of how failures between components are correlated. The final challenge addressed in this dissertation is one of evaluation. How can we evaluate the performance of these systems in a realistic setting, and how do the resulting systems perform in the face of real failures?

1.3.1 Finding and Using Redundant Paths

As Chapter 3 shows, the Internet graph topology offers numerous partially-disjoint paths between any two nodes. There are, however, several significant challenges in actually making use of this redundancy:

1. The Internet architecture exposes only a single path to end-hosts. How can a failure masking system access additional paths?
2. How can the failure masking system respond *quickly* to a path failure between N participants? For large values of N, a general solution scales as $O(N^2)$.

This dissertation first presents a system that takes advantage of this *in situ* redundancy by optionally forwarding packets through other hosts on the Internet. This system, Resilient Overlay Networks (RON), was able to achieve a 50% reduction in the number of end-to-end outages that affect traffic. However, it was often hampered by access link failures at singly-homed sites—a lack of physical redundancy.

Building upon the lessons we learned from RON, the MONET Web proxy system uses explicitly engineered client redundancy (*e.g.*, multiple DSL and cable modem links) in addition to exploiting the redundancy already available in the Internet. By combining these approaches, MONET was able to avoid the client link failures that are a major cause of reduced availability for smaller customers, and was able to improve the client-perceived availability of web access by nearly an order of magnitude. MONET uses a novel *waypoint selection* algorithm to determine the order in which it attempts to use the paths available to it, and the time it delays before using those paths. Waypoint selection allows MONET to function while sending fewer than 8% more packets and less than 0.5% additional bytes of network traffic.

When there are redundant paths, how should the system select which path to use? We believe that the proof lies in the pudding: a path should be used if, and only if, it will transport the desired data from its source to its destination. The systems we describe approximate this goal using in-band host-to-host probes that are likely to follow the same path as the actual data, and using the recent performance history of the paths to select which should be used.

1.3.2 Deployment Challenges and Overlay Networks

Once the system has access to multiple paths, it must address two subsequent issues:

1. What are the right metrics for path selection? Applications may have widely differing needs.
2. How can such a system be deployed? Making rapid change to the core Internet architecture is a slow and difficult process.

This dissertation argues that overlay networks meet the needs of both of these requirements. An overlay network is a network that runs on top of another network. In the case of an overlay network built over the Internet, the overlay network treats a host-to-host path provided by the Internet as a "link." The hosts in the overlay participate in an overlay routing protocol and cooperatively forward data for each other. Overlay networks have become a popular field of research in the last few years; Chapter 2 discusses some related efforts. From our perspective, overlays provide two critical capabilities in addition to their use as a mechanism for path selection: incremental deployment and specialization.

Overlay networks are deployed among cooperating end-hosts without requiring changes to the underlying Internet infrastructure, and can therefore be deployed quickly and incrementally. In contrast, changing the IP routing infrastructure is difficult because of the large installed base of IP systems that would need to be changed—its very success makes the Internet harder to change, as is demonstrated by the still-limited deployment of IPv6 ten years after it was adopted as a proposed standard [66]. While we view incremental deployability as a necessary attribute for many systems we design, we strive to design systems that provide a strong adoption incentive: not only are they incrementally deployable, but even a limited deployment provides immediate benefits to the early adopters.

The second benefit of overlays is that they avoid burdening the underlying network with features better performed at higher layers, and a bug in the overlay network is much less likely to crash the underlying IP routing infrastructure. This means that systems like MONET can choose paths based on Web server reachability without requiring that core Internet routers understand anything about the higher level protocols. Overlays also constrain their state (topological, routing, and so on) to a small number of participating nodes, instead of propagating that state throughout the Internet.

The separation of state provided by overlays means that IP layer engineering can continue to focus on providing general, highly scalable best-effort delivery, while supporting a variety of overlays that incorporate application-specific functions or do not support global scale within a single overlay. The final chapter of this dissertation provides an example of such an overlay, named Mayday. Mayday uses an overlay network in conjunction with existing, simple router-level filtering to shield servers from DDoS attacks. By moving to the overlay nodes the heavy-weight functions of authentication and deciding whether traffic is malicious, Mayday reduces the state and processing burden on high-speed Internet routers.

1.3.3 Measure, Implement, Measure, *Understand*

Finally, what are the right evaluation metrics for the resulting system? There are three challenges to the evaluation of Internet failure masking systems that must be overcome:

1. The lack of detailed failure models makes an accurate analytical evaluation difficult. While numerous studies have examined the *first-order* properties of failures, such as their duration and frequency, the evaluation of a failure masking system requires a model that captures the correlations between failures: if link A goes down, does backup link B also fail?
2. Empirical measurement requires a testbed. How should such a testbed be structured so that it observes a justifiably diverse set of Internet paths? At the time this research began, there were no publicly available large-scale Internet testbeds.
3. What workload should be used to characterize the systems? Simply measuring connectivity suffices for some systems, but in order to account for the behavior of users (*e.g.*, users avoid sites that perform poorly), other portions of the analysis must be driven by user input.

A recurring theme in this dissertation is the cyclic interplay of measurement and implementation, both within the development of the individual systems and within the larger context that drives the development of subsequent systems. Internet measurements and systems-building is an experimental research area within computer science, and one for which we lack good models of failures and the interactions between components. A key element of our work is evaluating the systems we create in a deployed experimental testbed, so that we can observe the performance of the systems under real conditions.

These evaluations primarily take place on the RON testbed, a distributed Internet testbed consisting of 36 nodes at 31 sites that we deployed over a period of four years. The nodes in the testbed are hosted at a variety of sites—educational institutions, corporations, ISPs, and private homes—and observe between them a wide spectrum of link speeds, technologies, and administrative domains.

While any testbed represents only a (biased) sample of Internet paths, the RON testbed observes a sufficiently rich set of paths that it permits us to evaluate our systems under a range of real-world conditions and to observe numerous real-world failures.

At a high level, the work in this dissertation was conducted in cycles of measurement, analysis, and implementation. The initial measurements help drive the design of a particular system, which is then refined through implementation and additional measurements. In the end, that system and measurements thereof can help provide a deeper understanding of the Internet environment, and shed light on additional problems. An example of this feedback loop is noted above: the RON system was guided by initial measurements that suggested the potential of alternate-path routing, and the subsequent evaluation of the complete RON system led us to develop MONET, which corrected for the classes of failures not addressed by the first system.

1.4 Contributions

This dissertation explores the hypothesis that the myriad failures interrupting Internet connectivity can be addressed on an end-to-end basis by layering routing and probing protocols atop the existing network substrate. We present several key findings that we believe are applicable to the design of a variety of Internet-like and Internet-based systems:

1. **Overlay routing can *improve* upon the underlying network.** RON demonstrates that not only are overlay networks a convenient deployment mechanism, but this additional layer can actually enhance the availability and performance of the Internet. To our knowledge, RON is the first wide-area network overlay system that can detect and recover from path outages and periods of degraded performance within several seconds. These recovery times are nearly an order of magnitude shorter than BGP's wide-area failure recovery times, *when* BGP is able to route around the failure.

   The work in this dissertation shows that overlays are not only a good interim deployment mechanism, but that they are a good way to provide many features once thought to be exclusively in the domain of the underlying network.

2. **End-to-end measurements can guide path selection.** Because Internet failures have a variety of causes and manifestations, end-to-end indications of path usability form a strong basis for choosing between different paths. By having end hosts decide whether or not a path is suitable, a failure making system can detect and avoid problems—such as those caused by operator error or malfunctioning hardware—that evade conventional routing systems.

3. **It is possible to provide higher availability between *actively* communicating hosts than between *potentially* communicating hosts.** In line with the previous observation about end-to-end probes, the systems described in this dissertation attempt to detect and mask failures between *actively* communicating end-points. RON restricts its probing and routing to a small group of cooperating nodes; MONET uses in-line probes that only verify that a particular path is functioning. By limiting the scope of their recovery, these systems avoid the challenge faced by Internet routing protocols of ensuring that all $O(N^2)$ potential node-pairs on the Internet can concurrently communicate. This narrowed focus enables failure masking systems to respond more quickly and to a wider variety of failures than conventional approaches permit.

4. **Waypoint selection can quickly locate working paths** without requiring excessive probing traffic. Chapter 6 presents our formulation of the waypoint selection problem, a generalization of the server selection problem that benefits from the knowledge of past accesses using particular paths. By attempting to use different Internet paths in a good order, a failure masking system improves its ability to find a working path without spending excess time attempting non-working paths.

5. **Real-world failures are frequent and often avoidable**. Typical Internet paths provide under 99.7% availability. These failures occur at a variety of locations in the network (Chapter 7 and [51]). Fortunately, these failures can often be masked through the use of simple techniques such as an overlay network or the use of a cheap additional Internet access link.

Finally, most of our analysis is based upon the **real-world evaluation of a production system**. RON has been adopted into at least one commercial product [106], and the MONET prototype has been running in our lab for nearly two years. To evaluate RON and MONET and to collect data about Internet failures, we deployed the RON testbed, which became a useful research artifact in its own right, being used by researchers from eight different institutions and by numerous projects within MIT [10]. The testbed's model of heterogenous, contributed network sites with centrally managed, homogenous systems software became the model of choice for the large-scale PlanetLab testbed [122]. A key aspect of the testbed was its diverse network locations, which ensured that the systems running on it saw a wide variety of network conditions and events. The testbed provided a framework for the extensive real-world evaluation and measurement of RON and MONET, which forms the final contribution of this research.

RON and MONET effectively mask many failures, but this dissertation opens several problems in its wake. While chapter 9 discusses work that has examined the scalability of RON-like systems, the *stability* of a network in which many overlay networks interact has yet to be examined in depth [80]; is it possible for these systems to coexist without oscillations and inefficiency? Second, while two recent studies found similar failure and failure avoidance statistics to those in this dissertation [64, 179], these studies all took place on testbeds that may only partially reflect the characteristics of the whole Internet. The software systems presented in this dissertation are available on the Web, and we hope that a wider deployment can help answer these questions.

We also leave for future work one final question: Knowing that many Internet failures *are* avoidable, and knowing that overlay networks are an effective way to provide application-specific functionality across the network, how would one change the core Internet architecture to provide improved service to its users?

The remainder of this dissertation is structured as follows: Chapter 2 examines related work in improving Internet and Web availability, protecting against denial of service, and overlay networks. Chapter 3 describes the general measurement methods used to evaluate the research in the remainder of the dissertation, including a description of the RON testbed.

Chapters 4 and 5 describe the design, implementation, and evaluation of RON. Chapters 6 and 7 build upon our results with RON and proactive packet replication and present MONET, a Web proxy system that exploits overlay routing, physical multi-homing, and server replication to increase a client's ability to access Web sites by nearly an order of magnitude.

The RON and MONET chapters examine failures in the core of the Internet and failures at the client access link that require explicit engineering to avoid, but they do not address failures caused by

Denial-of-Service attacks specifically targeted at a victim. Chapter 8 presents Mayday, an overlay-based system that acts as a distributed filtering system to shield Internet servers from DoS attacks. Chapter 9 concludes with a discussion of future work and open issues raised by this dissertation and the lessons that we have drawn from the RON and MONET systems with respect to network architecture.

Runners just do it - they run for the finish line even if someone else has reached it first.

- Author Unknown

Chapter 2

Related Work

The work in this dissertation builds upon a very broad base of related research into Internet reliability and performance. This chapter first presents several studies of Internet performance that show the current state of availability for Internet systems. It subsequently discusses two critical subsystems of the Internet: routing and DNS. It then presents a summary of work to improve networked availability and to prevent and detect Denial-of-Service attacks. This chapter concludes with a brief survey of related work in overlay networks.

2.1 Availability and Performance of Internet Systems

Problems with Internet availability have been recognized for many years. This section examines several pivotal studies of the effects that failures have on routing, host-to-host availability, and service availability. Each of these studies measures something slightly different; most studies of routing availability and host availability will slightly overestimate end-to-end service availability: routing availability is necessary, but not sufficient, for host-to-host reachability. Service availability depends both on host-to-host reachability and DNS and software availability. Availability also varies with the sites being measured: Rexford's recent examination of routing stability found that highly popular (and thus, presumably, well-maintained) sites had considerably better performance than average [130]. Keeping these factors in mind, we first examine studies of routing availability, then host-to-host reachability, and finally service availability.

Labovitz *et al.* examine routing availability, finding that 10% of all Internet routes were available less than 95% of the time, and that fewer than 35% of all routes were available more than 99.99% of the time [89]. Furthermore, they find that 40% of all path outages take more than 30 minutes to repair and are heavy-tailed in their duration; only 40% of routing failures are repaired in 10 minutes [87]. Much of the routing information flowing in 1998 was pathological, providing no useful topological updates [88], and much of this bad traffic was generated by small service providers.

Paxson studied Internet failures and routing pathologies by running periodic traceroutes between 37 distributed sites on the Internet [117]. His study noted that "major routing pathologies" disrupt two-way connectivity between hosts between 1.5% and 3.4% of the time, and that this percentage has not improved with time. The Skitter project collects similar end-to-end probes along with numerous traceroutes to help map the Internet topology and identify critical paths [20]. Compared to Labovitz's data, which studied routing failures, both Paxson's data and the data presented in this dissertation suggest that "average" failures are repaired somewhat more rapidly—in several minutes—than are BGP-reflected routing failures, which may require three to 15 minutes to repair. In earlier work, we found that many end-to-end failures are not reflected in BGP traffic [51].

Dahlin *et al.* measure two-way path failures using traceroutes and HTTP probes [36], analyzing their own measurements and several earlier datasets, including those of Paxson. Five percent of the failures detected by their active probes last longer than 10,000 seconds (2 hours, 45 minutes). The failure durations are heavy-tailed and can last for as long as 100,000 seconds before being repaired. While routing appears to be more stable for popular sites [130], many failures still occur at the client access link and in the middle of the network (Chapter 7). These findings do not augur well for mission-critical services that require a higher degree of end-to-end communication availability. The authors of the traceroute and HTTP trace study note that their probing method may underestimate failures near the destination.

Considerable work has examined the performance and availability of hosts and services; this overview touches only briefly on work that focuses on the availability of networked services. Numerous techniques replicate services across multiple back-end nodes; for a more extensive discussion, see the related work in [14]. Oppenheimer studied the causes of failures for Internet services, finding that operator error was a dominant cause [107]. Systems such as BFS exploit redundancy inside a local cluster service to increase availability [24].

2.2 Internet Infrastructure

Three of the most important parts of the Internet infrastructure are the physical links that carry data, the routing substrate that selects a path over the physical links, and the Domain Name System that provides a human-friendly naming layer. Because their proper functioning is critical to the success of almost all Internet-based applications, we must first understand how they succeed, and how they can fail.

2.2.1 Link Management and Intradomain Routing

Physical link redundancy is often managed using link-level techniques such as SONET (Synchronous Optical Network). When fiber deployments are dense enough to permit it, SONET links are often deployed in a redundant ring configuration. When the link is disrupted by a single failure, every point remains connected, and the devices on the ring can recover from failures in under 50 milliseconds [17]. SONET-level redundancy is common within large metropolitan areas in which connectivity is dense and failures due to link disruption (from, *e.g.*, road and utility construction) are common.

While SONET provides fast failover for local failures, it is not commonly used to guard against failures on long-distance links. Such links are often managed using ATM (Asynchronous Transfer Mode) switching or MPLS (Multi-protocol Label Switching), which operate underneath the IP layer. MPLS can pre-compute alternate paths to use in the case of failure; using this "fast reroute" feature, an MPLS switch can mask a failure in around 100 milliseconds.

Lower-speed parallel links can be "bonded" using techniques like multi-link PPP (Point-to-Point Protocol) [144], or by configuring some form of equal-cost load balancing on the links, often using an IGP such as IS-IS [108] or OSPF [103]. An analysis by Sprint engineers found that IS-IS intra-provider routing updates required 6-8 seconds to recover from a link failure, a number that could be reduced to 3 seconds with aggressive IS-IS timer tuning [69]. A recent Internet-Draft suggests that IS-IS failover could be tuned to occur in the range of a few hundred milliseconds [6].

2.2.2 Interdomain Routing

In 1987, the Internet had just passed 10,000 hosts, and had under 1,000 constituent networks. Its *backbone* links were primarily 56 and 9.6 Kbits/s. The original ARPANET was managed by a single administrative entity, reducing the need for exterior routing protocols. Early ARPANET routing was more dynamic than today's backbone routing, responding in real-time to the delay and utilization of the network. By 1989, the ARPANET evolved to using a delay and congestion-based distributed shortest path routing algorithm [82].

The diversity and size of today's Internet (hundreds of thousands of networks, over 100 million hosts, with no central routing authority) necessitated the deployment of protocols that perform more aggregation and less frequent updates. As a result, though interior routing within systems is still sometimes performed in an ARPANET Shortest Path First (SPF)-like fashion using OSPF or other protocols, routing *between* systems is performed using a different protocol: BGP, the Border Gateway Protocol, version 4 [126]. BGP's routing decisions are based primarily reducing the number of Autonomous Systems that packets must traverse; its decision process does not consider performance, even at high levels of packet loss. BGP's design emphasizes scalable operation over all else.

Because BGP permits customers to announce routes through multiple ISPs, an oft-cited "solution" to achieving fault-tolerant network connectivity for a small or medium-sized customer is to *multi-home*, *i.e.*, to connect to several upstream providers. The idea is that an outage in one ISP would leave the customer connected via the other(s). However, this solution does not generally achieve fault detection and recovery within several seconds because of the degree of aggregation used to achieve wide-area routing scalability. To limit the size of their routing tables, many ISPs will not accept routing announcements for fewer than 8192 contiguous addresses (a "/19" net block). Small companies, regardless of their fault-tolerance needs, do not often require such a large address block, and cannot effectively multi-home. One alternative may be "provider-based addressing," where an organization gets addresses from multiple providers, but this requires handling two distinct sets of addresses on all of the hosts within the organization. The MONET system presents one mechanism that clients can use to provide fail-over between multiple address sets; connection migration, discussed later in this chapter, provides another.

Inter-domain routers in the Internet may take tens of minutes to reach a consistent view of the network topology after a fault, primarily because of routing table oscillations during BGP's rather complicated path selection process [86]. During this period of "delayed convergence," end-to-end communication is adversely affected. While part of the convergence delays can be fixed with changes to the deployed BGP implementations, long delays and temporary oscillations are a fundamental consequence of the BGP path vector routing protocol.

2.2.3 DNS

The Domain Name System [100] provides a translation service for human-readable names (*e.g.*, "eep.lcs.mit.edu") to IP addresses (18.31.0.114). Most common Internet services depend on the proper functioning of DNS. Studies of Web availability show that DNS delays are one of the major components in slow Web transfer times [32].

Fortunately, there is room for improvement. Jung *et al.* find that 23% of lookups go unanswered, and 13% receive an error [76]. Many of these unanswered queries are retransmitted several times. As a result, more than half of all DNS packets are junk queries that provide no useful information to the client. The fraction of Internet traffic used by DNS packets has been consistently going down since 1990 as the amount of traffic handled by other protocols (*e.g.*, HTTP, peer-to-peer file-sharing traffic, NNTP, and SMTP) has increased dramatically. We use this to our advantage when designing MONET's algorithms for resilient DNS lookups, which trade a small increase in traffic for a large reduction in delays due to failures.

The CoDNS [113] system is designed to mask DNS lookup delays by proxying DNS requests through peers. It is primarily concerned with delays and failures introduced by a local DNS resolver shared by many hosts on the same network. When CoDNS does not hear a DNS response from its local nameserver within a short static timeout (200 to 500ms, typically), the CoDNS resolver forwards the query to a peer node. When a majority of recent requests get resolved through a peer node, CoDNS instead immediately sends all queries both locally and through the peer.

2.3 Improving End-to-End Availability

2.3.1 Alternate-Path Routing and Multi-homing

The Detour measurement study [140] was one of the earliest to observe the potential for performing host-to-host alternate path routing on the Internet. Using Paxson's and their own data collected at various times between 1995 and 1999, Detour observed that path selection in the wide-area Internet is sub-optimal from the standpoint of end-to-end latency, packet loss rate, and TCP throughput. This study showed the potential long-term benefits of "detouring" packets via a third node by comparing the long-term average properties of detoured paths against Internet-chosen paths.

While RON (Chapter 4) shares with Detour the idea of routing via other nodes, it differs from Detour in two significant ways. First, RON seeks to prevent disruptions in end-to-end communication in the face of failures. RON takes advantage of underlying Internet path redundancy on time-scales of a few seconds, reacting responsively to path outages and performance failures. Second, RON is designed as an application-controlled routing overlay; because each RON is more closely tied to the application using it, RON more readily uses application-specific path metrics and path selection policies.

Several studies and products demonstrate that allowing clients to choose between multiple paths to the server increases reliability. NATRON extended the overlay routing ideas of RON by using NAT to reach external hosts [177]. The Detour project had made a similar NAT-based proposal [33].

SOSR (Scalable One-hop Source Routing) demonstrates that one-hop overlay routing achieves excellent reliability with low overhead [64], which agrees with the results we present in Chapters 5 and 7. In the SOSR system, a client that detects a path failure selects one or more intermediate nodes and routes its requests through them. Requests go directly from this intermediate to the destination. The authors found that selecting four random intermediaries was sufficient to obtain most of the benefits of indirect routing, focusing on longer failures between 30 seconds to six minutes. The techniques are practical, and are implemented in a NAT-based prototype. Clients are configured to use a SOSR node as their IP gateway, and their packets are automatically routed through the overlay to the destination node.

Commercial products like Stonesoft's "Multi-Link Technology" [157] and Fatpipe's "Redundant Array of Independent Lines" technology [49] send multiple TCP SYNs to servers to multi-home clients without BGP. RadWare's "LinkProof" pings a small set of external addresses to monitor connectivity on each link, failing over if a link appears to be down [63]. RouteScience [133] and SockEye [150] use end-to-end measurements to select outbound routes for networks with multiple

BGP-speaking Internet links. These techniques are similar to some of those discussed in Chapter 6 for improving the availability of generic Web sites.

Akella *et al.*'s recent measurements show that multi-homing with two local links can improve latency by about 25% [3]. In follow-up work, they considered a NAT-based system and compared the effectiveness of passive monitoring and active monitoring. Some sort of route control is important to optimize response time, but the improvements are insensitive to the exact mechanism and measurement algorithms chosen [5]. These results complement our findings: MONET focuses primarily on strategies for achieving the reliability benefits of multi-homing (the worst five percent of responses), while these studies focus on latency improvements.

Akella *et al.*'s more recent study of five days of pings between 68 Internet nodes found that most paths have an availability of around 99.9% [4]. These numbers are consistent with our estimates of link failure rates discussed in Chapter 7. The remainder of that chapter examines the contribution of other sources of failure and extends this analysis to a much wider set of hosts.

Another benefit of multi-homing is that systems can load balance between multiple links, achieving a larger aggregate throughput. Guo *et al.* examine several multi-homing strategies designed to balance load on a per-packet or per-flow basis [65]. Our experience combining cable and DSL links with MONET suggests that such balance is achievable *and* desirable. Goldenberg *et al.* evaluate multi-homing strategies when the per-link traffic price is non-uniform [58].

2.3.2 Caching and Prefetching

Web caching is extensively used to reduce traffic volumes and improve client latency. Caches can act alone, or in concert with a cooperative hierarchy of other caches [47, 25]. Cooperative cache peering increases the effective number of users and the amount of storage that the cache has, permitting it to achieve higher hit rates, but these benefits fall off quickly with moderate cache hierarchy sizes [175]. The availability benefits of caching are less widely studied. Dahlin and Chandra found that caching can be used to improve availability, but only when combined with extremely aggressive prefetching of documents [36]. Systems like CoDeeN [110] use more involved techniques to fetch content through a mesh of cooperating proxies, blending the client-driven notion of a cache hierarchy with the structure of a content delivery network (See Section 2.3.3).

While Dahlin *et al.* examined the effects of prefetching and Web objects, Cohen and Kaplan studied the effects of prefetching the means to obtain those objects, namely, DNS responses and TCP connections [32]. Their trace-based simulations show that these small prefetches can reduce document transfer times substantially, and work even for uncacheable objects. Prefetching DNS and TCP connections is particularly effective at reducing the tail of the latency distribution, a major goal of the MONET system described in Chapter 6.

2.3.3 Content Replication

Content replication aims to avoid both failed Internet paths and overloaded servers and access links, improving the availability of *content* instead of improving the availability of a particular host. Content Delivery Networks (CDNs) like Akamai [2] and Coral [55] cache content at intermediate nodes to which clients are redirected using DNS or server redirects. These systems are *server-driven* - the server operators configure their Web servers to direct clients to the CDN. A large CDN like Akamai may consist of thousands of nodes co-located with ISPs around the world.

CDNs deliver replicated popular content and prove particularly effective in the face of flash crowds [153, 154], but without additional reliability mechanisms like those we consider, they do not effectively mask failures involving uncached content. The techniques we discuss in this dissertation would work well in helping CDNs increase their availability. Though information about it is scarce, we believe that Akamai's Akarouting [98], like RON, selects indirect hops through Akamai's CDN to reach origin servers.

2.3.4 Server Selection

CDNs use one approach to directing clients to a particular replica, but their design choices are strongly influenced by the huge scope of their replication networks, in which it is often possible to direct a client to a server immediately up-stream from it. Server selection research aims to determine how to select a content source when the assignment problem is not quite as straightforward.

SPAND [142] passively measures Internet transfers and stores the results in a shared repository. Future transfers can use this history to choose a good server replica for fetching data. Both RON and MONET benefit from SPAND's ideas of a shared performance database to help aggregate performance information across transfers and, when possible, across clients.

Smart Clients argues that load balancing and server selection should reside in the client applications for better flexibility and performance [178]. This approach downloaded mobile code to the client; the approaches we discuss in Chapter 6 achieve many of the same reliability benefits without changes to name resolution and without mobile code.

Dykes *et al.*'s comparison study of server selection techniques [43], concludes that picking the first Web server to respond to a SYN is more effective than using statistical records of transfer rates from the servers, particularly when networks are heavily loaded. They found that most load balancing algorithms reduced pathologically long delays by 7% to 20% over random selection. These findings inspired the techniques we use to pick a good client-server path in MONET.

Earlier work by Crovella *et al.* suggested that latency-based server selection outperformed both random selection and static selection based on the geographic location or the network hop-count of the remote servers [34].

A different approach to server selection argues for using multiple servers concurrently. Dynamic TCP parallel-access [131] load balances between multiple servers by downloading different parts of the same file in parallel from different Web servers. Parallel downloading provides load balancing at the expense of more complex reassembly machinery and more aggressive network utilization. Parallel schemes do not work well for small files and short transfers. As with connection migration, discussed below, parallel downloading schemes provide an answer to handling mid-stream failures during long-running connections.

2.3.5 Disconnection and Reconnection

Most of the techniques we study in this dissertation attempt to preserve or establish an end-to-end connection in the presence of faults. When faults are persistent or non-maskable, or when clients are mobile, connections may be interrupted. Several projects aim to enable clients to reconnect after they are disconnected or to change their attachment to the network. These techniques complement those we discuss in this dissertation by addressing another consequence of long-term failures.

The migrate framework [147] uses *sessions* layered atop conventional TCP or UDP connections. Within this framework, clients can disconnect and reconnect, possibly much later. Other work on mobility aims to allow clients to reconnect TCP sessions [148], or change IP addresses by relaying their packets through a designated third party [121].

These techniques can also be implemented on a per-application basis. For example, SFS [94] streams NFS-like operations over TCP to provide secure remote filesystem access. It transparently reconnects its TCP connection if it breaks, performing a new lookup of the remote host address. All of these connection migration techniques work in concert with the failure masking systems described later in this dissertation.

2.4 Denial of Service

DoS attacks have been increasing in frequency, severity, and technical sophistication in recent years. They have moved from single-host attacks to coordinated attacks by thousands of machines, termed distributed DoS, or DDoS attacks [39]. Moore *et al.* analyzed spoofed packet DoS attacks in 2001 by watching "backscatter" traffic sent by victim hosts back to spoofed random attack sources and observed between three and five thousand attacks of this type per week [102]. The problems have only become worse since then. In the rest of this section, we briefly present some of the most pertinent related work in DDoS prevention and detection. We refer the interested reader to an extensive taxonomy of DoS attacks and responses by Mirkovic and Reiher [99].

2.4.1 DoS Prevention

The most basic mechanism for reducing the severity of DoS attacks is ingress filtering, which prevents hosts from spoofing the source of their IP packets [52]. Ingress filtering has been deployed with moderate success, but its deployment remains far from complete. The drawback to ingress filtering is that it requires cooperation from most networks; prevention of IP spoofing is, unfortunately, difficult to achieve in the core alone [112].

Ingress filtering makes it easier to detect the source of an attack and to filter it, but it is not sufficient to prevent attacks by attackers with large numbers of sacrificial machines at their disposal. Attackers can obtain thousands of these machines through worm programs or other types of automated exploits [99]. With these *zombie* machines, they can flood victims with un-spoofed traffic.

Mazu Networks [95] and Arbor Networks [13] provide DoS detection and prevention by creating models of "normal" traffic and detecting traffic that violates the model. If an attack is detected, Mazu's tools suggest access lists for routers. If the Mazu box is installed in-line with the network, it can shape traffic to enforce a previously good model. Asta Networks' Vantage analyzes NetFlow data to detect DoS attacks on high-speed links and suggest access lists to staunch the flood [15]. These access lists must be deployed manually, and provide reactive, not proactive, assistance to the victim of a DoS attack. In Chapter 8, we discuss probing attacks that are effective against schemes using access-list based filters.

Pushback provides a mechanism for pushing rate-limiting filters to the edges of an ISP's network [72]. If attack packets can be distinguished from legitimate traffic (as in the case of a SYN flood), Pushback's mechanisms can effectively control a DoS attack. In the general case, Pushback will also rate-limit valid traffic. If the source of the traffic is widely distributed over the network, Pushback is less effective. In any event, Pushback is effective at reducing collateral damage to other clients and servers that share links with the DoS target, but this scheme requires new capabilities of routers, slowing deployment.

2.4.2 DDoS Detection and Traceback

ICMP traceback messages were proposed as a first way of tracing the origins of packets [18]. Under this scheme, routers would periodically send an ICMP message to the destination of a packet. This message would tell the recipient the link on which the packet arrived and left, allowing the recipient of a sufficient quantity of ICMP traceback messages to determine the path taken by the packets.

To avoid out-of-band notifications, Savage *et al.* use probabilistic in-line packet marking to allow victims to trace attack packets back to their source [141]. In this scheme, routers occasionally note in the packet the link the packet has traversed; after sufficient packets have been received by the victim host, it can reconstruct the full path taken by the attack traffic. Dean *et al.* improve the performance and robustness of probabilistic packet marking by treating the path reconstruction problem as an algebraic coding problem [38].

The probabilistic traceback schemes require that a large amount of data be received by a victim before path reconstruction can be performed. To allow traceback of even a single packet, the Source Path Isolation Engine (SPIE) system records the path taken by *every* packet that flows through a router [149]. SPIE uses a dense bloom-filter encoding to store this data efficiently and at high speeds. While it provides exceptional flexibility, SPIE requires considerable hardware support.

2.5 Overlay Networks

This dissertation makes extensive use of overlay networks to increase end-to-end availability. Overlays are an old concept in networking—the Internet itself was initially deployed as an overlay on top of the telephone network, using long-distance telephone links to connect Internet routers. Modern overlays operate similarly, using the Internet paths between end-hosts as “links” upon which the overlay routes data, building a network on top of the network. By taking routing to end-hosts, overlays permit designers to implement their own routing and packet management algorithms on top of the Internet. Instead of requiring years of upgrades to Internet routers, overlays can be used to deploy new functionality almost immediately. They also present developers with a flexible and powerful platform on which to create services.

The most immediate benefit of using an overlay network instead of modifying Internet protocols or routers is that the overlay provides a quick and easy deployment path that lacks many of the technical and political hurdles of a router-level deployment. There is a growing feeling among many Internet researchers that IP and the IP routing infrastructure have become ossified by virtue of its huge success. Changing the hundreds of millions of currently deployed IP-speaking devices poses a considerable challenge—witness the extremely long projected roll-out times for IPv6. Instead of changing the IP layer, many researchers now design protocols that run on top of IP in an overlay.

Another, more subtle, benefit of overlays is that they avoid burdening the underlying network with features better performed at higher layers—and a bug in the overlay network is much less likely to crash the underlying IP routing infrastructure. Functions such as content routing require that the content routers possess deep knowledge of the application protocols that run through them. This set of protocols is likely to change frequently; augmenting core routers with application-specific knowledge would burden them with processing needed only by a small fraction of the traffic that passed through them. Many routers handle IP options and other rare events on a “slow path” that has only a fraction of the forwarding speed of normal packet processing. This differentiation has historically made such routers vulnerable to additional denial-of-service attacks. Adding even more esoteric processing requirements to routers presents both a performance and a security headache for operators.

Finally, overlays provide access to resources far beyond the constrained environment of high-speed routers. An overlay can take advantage of the glut of processing, memory, and permanent storage available in commodity hardware to perform tasks that would ordinarily be well beyond the ability of a conventional router, such as expensive cryptographic operations, file caching, or database lookups. The ability to perform these tasks enables the creation of powerful new facilities such as scalable, distributed publish-subscribe systems and content distribution networks; performing them in an overlay keeps slow and expensive operations off of routers' critical paths.

Overlays have been used off and on for many years. Early deployment of multicast protocols took place on an overlay called the MBone [46], but overlays were not generally regarded as a distinct area of research on their own. This changed in the late 1990s when two types of overlays bloomed: routing overlays, and storage and lookup overlays, also termed "structured" overlays.

2.5.1 Routing Overlays

Routing overlays attempt to enhance or replace the routing performed by the underlying Internet, hopefully providing new functionality or improved service. Virtual Private Networks (VPNs) are an example of simple routing overlays that provide better network-level security and authentication. Routing overlays like RON were designed to provide better resilience to network failures than could Internet routing protocols. The X-Bone [163] attempted to provide generic overlay support to facilitate the deployment of overlays like the MBone.

A number of projects have used routing overlays to implement multicast, a service for which Internet-wide deployment within routers has proven elusive. Overcast [75], Yoid [54], and End-System Multicast [68] all group nodes into logical content distribution networks and use them to provide multicast services. By aligning the logical topology of the multicast nodes with the physical topology of the underlying Internet, these projects typically try to minimize the number of redundant data copies sent on any given Internet link, while simultaneously attempting to minimize the extra latency incurred by clients.

Other overlays have been used to provide anonymous communication, censorship-resistant publication [29], and numerous additional features. Mixnet-based anonymizing overlays like Tarzan [57], Tor [41] and FreeHaven [40] are designed to prevent observers from determining the identity of communicating hosts. The principles used in these overlays, primarily Chaumian Mixnets [26], can be directly used in a system such as Mayday (Chapter 8) to provide greater protection against certain adversaries.

2.5.2 Structured Overlays

Structured overlays like Chord [156], Pastry [134] and Tapestry [180] focus on techniques to harness the power of large distributed collections of machines. Storage and lookup overlays have become a substrate upon which a number of large distributed systems projects are based. Like the systems presented in this dissertation, a major goal of many of these projects is to create an overlay network that is highly resilient to the failure of individual links and nodes. Structured overlays typically replicate content across a number of individual nodes. These functions are analogous to those performed by a content distribution network, but data in a structured overlay is often mutable, not merely a cached copy of the original object. Researchers have used structured overlays to implement RON-like services [180], high bandwidth data dissemination [83], distributed filesystems [35], and more.

Now to the kitchen. Assemble the mise en place. *The concept is simple: wash, chop, and measure all ingredients (or software, as I like to think of it), and gather all hardware before you start cooking*

- Alton Brown

Chapter 3

Method

This chapter describes the general approach used in the research presented in the rest of this dissertation. In general, this research follows a cycle of measurement, analysis, design, and implementation, with an emphasis on real-world measurements of both network characteristics and the performance of the systems we built. Most of the Internet measurements used to understand and validate the remainder of this dissertation were taken over a period of three years on a testbed that we deployed. This chapter discusses the measurement/implementation cycle, the metrics we used to evaluate our systems, and the details of our experimental wide-area Internet testbed.

3.1 Approach

Internet failure avoidance is an empirical field. While several measurement efforts have noted first-order failure statistics (how often, how long, and so on), the systems community lacks models that capture the second-order effects: when a failure occurs between points A and B, does it also occur between A and C? B and C? Unfortunately, masking failures depends precisely upon avoiding such correlated failures. The research presented in this dissertation relies heavily upon extensive measurements of Internet paths both to design and to evaluate the our systems.

Many of the systems described herein were initially motivated by a perception of a problem, typically poor Internet availability or performance. To better ground our understanding of the problem to be corrected, we began with an initial *measurement and analysis phase* to examine the feasibility of the approaches under consideration. Oftentimes, we had several preliminary hypotheses for solutions to the problem, and designed the initial measurement experiments to help narrow down the solution space. For example, before designing the RON system in Chapter 4, we deployed a 4-node measurement system that sent small "ping" packets[1] and performed TCP transfers between the hosts at regular intervals. Our analysis of this initial data suggested that the RON approach was promising, so we proceeded to the *design* phase.

The design of our systems has two goals: to mask failures and to be easily instrumented. We wanted to create *useful* networked systems that increase availability. To do so, needed to collect sufficient data from the operation of these systems to evaluate their performance and explore alternative algorithmic choices. In some cases (such as the design of MONET's DNS resolver discussed in Chapter 6), the instrumentation goal lead to somewhat different design decisions for the system

[1] Using ICMP, the Internet Control Message Protocol.

than originally envisioned. Usually, however, these measurement hooks provided a way to continually evaluate the performance of the running system and help it adapt to long-term changes in network conditions.

Our goal was to create failure masking systems that are robust with respect to the cause of the failure. The majority of the systems discussed in this dissertation have been implemented and deployed in real Internet environments, from which we collected traffic and application traces to evaluate the system's performance. In some cases, we then directly analyzed these traces, and in others, we used the traces as input to a subsequent simulation phase during which we explored the effects of alternate design decisions. This performance evaluation typically concluded one round of our approach, and served as input to the next: by understanding the behavior of our system in the real world, we gained further insight into the failure characteristics of the Internet, and a base from which to create better solutions.

3.2 Measurement and Validation

This section describes several tactics for measurement that we developed and used to perform the research described in this dissertation. We used some of the techniques, such as the emphasis on basing the analysis around a database, from the beginning; others emerged through trial and error. The tactics in this section were derived from three goals:

1. **Confidence**: Have confidence in the validity of the measurements;
2. **Long-term utility**: Create an archive of useful data for later experiments;
3. **Efficiency**: Reduce wasted analysis effort and wasted time.

3.2.1 Measure More Than You Think You Need

When instrumenting a system, we intentionally collected more data than we believed we needed, and collected overlapping data via different mechanisms. For example, the MONET measurements in Chapter 7 include both application-level traces of when the Web proxy opens and closes connections, and `tcpdump` packet-level traces of the actual connections.

Over-measuring data provides three benefits:

1. Compensation for missing or corrupted data
2. Cross-validation
3. Data for subsequent experiments

Compensating for missing data is important when dealing with application-layer data whose accuracy may be degraded by programming errors or by increased timing uncertainty. More canonical sources of data such as packet traces are often better in this respect, but may still suffer from an error, *e.g.*, in specifying the set of packets to capture or the length of captured packets.

The second benefit is important for calibrating the measurements. In many Internet measurements, there is no ground truth against which to compare; as Paxson notes, it is important in these situations to be able to check the accuracy of measurements through some kind of self-consistency checks or by comparing multiple measurements [118].

Finally, designing, deploying, and collecting measurement data is time consuming. By collecting measurements with rich detail (*e.g.*, by using the synchronized clocks in the testbed described later in this chapter to record one-way transit times instead of collecting only round-trip times), the resulting data is more useful in subsequent, possibly unrelated experiments.

3.2.2 Develop the System and its Analysis Tools Concurrently

An important part of designing systems for measurement is performing the measurement in step with the system development. This helps eliminate bugs early, when they are easy to locate, and helps protect against collecting months of invalid data.

Not all design flaws and bugs result in obvious problems; many cause subtle performance changes or taint the resulting measurements in ways not apparent to a user of the system. During the evaluation of MONET, for instance, our analysis revealed that when modifying our Web proxy, we had neglected to upgrade one control path, leading to degraded performance under rare circumstances.

Creating the analysis framework during development also ensures that the system records the right things. This property is less important for simple Internet measurements such as our preliminary RON study, which primarily recorded only round-trip times and loss rates, but it was essential when instrumenting the more complex MONET system, which dealt with a wide array of connection types, requests, and servers.

3.2.3 Be Willing to Scrap Early Measurements

Legacy data formats and errors can cause an unnecessary headache for research data. An advantage of developing the analysis framework early is that one can avoid collecting voluminous data before finding bugs. A corollary to this is that discarding these "prototype" measurements once or twice can substantially ease the rest of the analysis. While losing several days of data is unpleasant, it is often preferable to having having an analysis framework burdened by special-case code to deal with the flaws in the early data.

3.2.4 Automate and Reuse

Automating the analysis process as early as possible and as much as possible makes it easier to try "what-if" scenarios, to adapt to errors, and to incorporate new or larger datasets. Strive for being able to type "make" and have the entire analysis run from data processing to graph production. Automation increases efficiency, and the ability to easily correct for errors and updates facilitates producing correct results.

Data analysis is susceptible to code and tool reuse. In our research, we have not generally required the power of dedicated analysis packages such as SAS or Matlab; we spent relatively more time turning collected data into a meaningful set of numbers than performing statistical analyses of the resulting numbers. Therefore, much of the analysis code consisted of scripts written in Perl or Ruby. Many of these scripts have very similar structure. We adopted a convention of having all lines in text files start with a standard UNIX timestamp (*e.g.*, 1088832560.621396), and developed standard utilities to split, merge, and gather subsets of these files. As noted below, using a database system with common schemas facilitates tool reuse.

3.2.5 Database Systems

The analysis in this dissertation makes extensive use of the MySQL relational database for processing data. While a database may appear on the surface to be excessively complex and add unnecessary overhead to the analysis process, the overhead it does add is rapidly offset by several important benefits:

1. **Data Management**: The data repository becomes more self-documenting, with clearly defined column names and relationships. Deleting data and adding new data is greatly simplified.
2. **Data Reuse**: Once the data is in the database, it is easy to run later analyses to address new questions. In Chapter 5, we describe the evaluation techniques for the RON system. We use a small extension of the same analysis framework to answer the questions posed in several other research efforts [12, 11, 51].
3. **Joint Analysis**: When all data is in a central, well-documented database, it is easy to examine the relationships *between* those datasets (*e.g.*, in our examination of the relationship between routing messages and end-to-end connectivity [51]).
4. **Data subsets**: With a database, running "test" analyses on sub-parts of the data is easy by simply changing a SQL "SELECT" statement or two. This ease of use facilitates tool development and debugging, as well as exploratory studies to derive an intuition for the data.

A sensible database schema can improve storage efficiency without compromising ease of use. While text files are easy to process without additional library support, they are often an inefficient way to store and process the data. Binary storage formats such as that used for CAIDA's Skitter data [20] are efficient, but they are not self-documenting and they require extra code to convert them into a usable form. A database can store measurement data compactly without impairing accessibility.

This basic analytical pattern has been very useful for our own research; the interested reader may also wish to read Paxson's "Strategies for Sound Internet Measurement" article [118] and Allman *et al.*'s "Scalable System for Sharing Internet Measurements" [8].

3.3 Failures and Performance Metrics

We define two kinds of failures that occur within the network. *Link failures* occur when a router or a link connecting two routers fails because of a software error, hardware problem, or link disconnection. A link failure does not need to interrupt end-to-end connectivity if it is *masked* quickly enough by either link redundancy or routing.

In contrast, a *path failure* is a disconnection of two end-points because the path between them is unavailable. Path failures may occur because of unmasked link failures, or for a variety of other reasons, including denial-of-service attacks, software and hardware failures, or configuration errors. This dissertation focuses primarily upon path failures, since these are the failures that are visible to end-systems.

Applications perceive failures in one of two ways: *outages* or *performance failures*. We define an outage as a path failure that creates an average loss rate over 30% that lasts for several minutes. At these high loss rates, the performance of most protocols, including TCP, degrades by several orders of magnitude, rendering the path effectively useless. Performance failures are less extreme; for example, throughput, latency, or loss-rates might degrade by a factor of two or three.

The most important metric we studied is end-to-end *availability*. Availability is best defined as the fraction of time that a system was actually available relative to the time it was intended to be available [136]. In this dissertation, we tend to make two simplifying assumptions about Internet availability and traffic: First, we assume that Internet paths are intended to be available 100% of the time; any "planned" downtime is still downtime if it interrupts traffic. Second, because many of our measurements were probes spaced in time, we assume that the fraction of packets or requests that successfully reach their destination provides a good approximation of availability.

Latency and *throughput* are two important performance characteristics of networked systems. Unless otherwise specified, we use the term “latency” to mean the round-trip time between two hosts. When examining measurements of Internet paths, we typically present latency exclusive of host processing time. When examining measurements of Internet services or Web sites that are outside of our control, we include processing time, because client-side mechanisms are unlikely to reduce server delays. The throughput of a path or connection refers to that achieved by a bulk TCP transfer using the path.

3.4 The RON Testbed

The majority of our operational results come from measurements from the *RON testbed*, which now consists of 36 operational machines at 31 different sites in eight countries, with nodes obtaining Internet service from at least 12 different providers (see Table 3.1). In addition to its use in our own research, the RON testbed has proved useful to other researchers over the past several years. Projects that have benefited from the testbed include overlay networks [9, 56], distributed hash tables (DHTs) [156, 134], DHT-based peer-to-peer file systems [35, 104], network analysis experiments [73, 158, 160, 67], and experiments to correlate end-to-end path failures with BGP routing messages [51].

The rest of this chapter provides an overview of the RON testbed and describes its key features and limitations. Based on our measurement experience and discussions with other testbed users, we have concluded that the most important strength of the testbed is in its *heterogeneous network connectivity*. This was an important factor in our choice of sites for the RON research described in Chapter 4. We strived to deploy nodes at companies, homes, ISP co-location facilities, and international sites. Currently, only six of the 31 host sites are U.S. universities.

The second reason for the success of the testbed in facilitating interesting research is the *homogeneity of the hardware and software platforms*. All our sites run identical versions of FreeBSD; this homogeneity reduces the hassles of deploying software on the testbed.

Because our original motivation was to determine the effectiveness of our failure masking techniques and not general testbed development, the RON testbed grew incrementally in *bottom-up fashion*. This turned out to be a strong advantage, because it allowed us to debug deployment problems more easily than if we had been forced to deal with a large number of problematic machines all at once.

Finally, a major operational win in the testbed arises from the *ease of testbed configuration and management*. Through collaboration with the Netbed project [174] at the University of Utah, we now have a set of simple tools that make it possible for a single person to manage the entire testbed: I have managed ours for several years as a “side” activity to my actual research.

3.5 Testbed Overview

The RON testbed runs on relatively homogenous, centrally managed hosts that reside in a heterogenous Internet environment, hosted by volunteers. This approach has predictable (and relatively low) up-front costs of about US $2,000 per node, and low on-going costs. It is easy to develop software in this homogenous environment where the testbed developers carefully control the operating system and installed libraries. Having the testbed run on machines similarly controlled it less subject to the whim of the old, out-of-date, and ill-managed machines that people are likely to donate to a testbed.

Previous Internet testbeds have ranged from dedicated IP-layer networks like the Internet2 [71], CAIRN [22], and DARTNet [164] to software-only distributions like NIMI [119] that run on Internet-connected machines.

Dedicated network testbeds are useful for low-level network protocol research because they provide an environment in which everything above the hardware is mutable. This flexibility, however, comes at significant cost. Dedicated network links are much more expensive than commodity Internet links, and maintaining these networks requires a large investment of time and expertise.

At the other end of the spectrum, projects such as NIMI [119] and the ABone [1] used no dedicated hardware, relying on shared UNIX accounts donated by host sites. The cost of the monetary savings is that these projects run in a heterogenous hardware and software environment, on machines that are in active use by other users and that are administered by external entities. The NIMI and ABone software must therefore isolate hosted experiments from the host machine. The heterogeneity creates a larger burden on software developers, and the stringent security concerns restrict the types of experiments that these testbeds can host.

3.5.1 Hardware and Software

Hardware and software homogeneity, to the extent that we can achieve it, enhances our ability to maintain the testbed. Organizations hosting nodes may choose between 1U rack-mount machines and mini-tower boxes. The rack-mounted units are vastly more popular, accounting for all but four of the deployed nodes. Because we have deployed the machines over three years, we have been able to increase the nodes processor, disk, and memory capacities over time. The type of hardware used, however, has been identical (Pentium III and 4 chips, the same series of Seagate SCSI drives, and Intel network interfaces), so the driver configuration on the machines has remained the same.

About half of the testbed nodes are equipped with CDMA (cellular) time receivers. These receivers[2] provide approximately $10\mu s$ time resolution, similar to GPS, but can operate inside most machine rooms without an external antenna. They provide time synchronization and facilitate one-way path delay estimates.

The machines currently run FreeBSD version 4.7. The testbed interface is a non-root account on all of the nodes. Select trusted users have root access on some nodes, and we install setuid binaries for users when necessary.

Standard accounts on the testbed are not virtualized—users see the base OS and the real network. Users are granted access to all nodes in the testbed, and must manually coordinate experiments and cooperatively share resources with each other. To date, the only resource abuses we have encountered were accidental and short-lived, but this is mostly a result of having a small and well-behaved user community. Low-overhead experiments execute concurrently and continuously, but we ask users with higher bandwidth requirements to check with us before running large experiments, providing some human-mediated resource allocation.

Software homogeneity precludes some kinds of research (*e.g.*, experimenting with different TCP implementations), but has significant benefits. Paxson moved his early measurement infrastructure from a multi-platform effort [116] to FreeBSD and NetBSD only, noting that he and his colleagues experienced problems with site configuration differences even with a small number of distinct platforms [119]. In our testbed, two sites donated machines with an OS already installed, and despite

[2] The Praecis Ct from EndRun Technologies.

Name	Location	Upstreams
Aros	Salt Lake City, UT	UUNET, ELI
AT&T	Florham Park, NJ	AT&T
CA-DSL	Foster City, CA	Pacific Bell
CCI	Salt Lake City, UT	ViaWest/L3
* CMU	Pittsburgh, PA	Many via PSC
Coloco	Laurel, MD	Many
* Cornell	Ithaca, NY	Applied Theory
Cybermesa	Santa Fe, NM	Espire
Gblx-ams	Amsterdam, NL	Global Crossing
Gblx-chi	Chicago, IL	Global Crossing
Gblx-jfk	New York, NY	Global Crossing
Gblx-lon	London, UK	Global Crossing
Greece	Athens, Greece	GEANT
Intel	Palo Alto, CA	AT&T+?
Korea	KAIST, Korea	
Lulea	Lulea, Sweden	
MA-Cable	Cambridge, MA	AT&T
Mazu	Boston, MA	Genuity
* **MIT**	Cambridge, MA	Genuity
NC-Cable	Durham, NC	
Nortel	Toronto, Canada	AT&T Canada
* NYU	New York, NY	Applied Theory
PDI	Palo Alto, CA	Qwest
PSG	Bainbridge Island, WA	Genuity, Verio
PWH	Sunnyvale, CA	Many
Sightpath	Near Boston, MA	UUNET+
sg	Singapore	
* UCSD	San Diego, CA	Many
* Utah	Salt Lake City, UT	Qwest
Vineyard	Cambridge, MA	Savvis, Qwest
VU-NL	Amsterdam, Netherlands	

Table 3.1: Testbed sites. Asterisks indicate U.S. universities on the Internet2 backbone. Bold indicates a BGP feed. If known, we list the site's upstream ISPs.

upgrading the machines to our standard software, we have encountered problems from root partition sizes and old configuration files. The Netbed boot CD (Section 3.5.3) has helped correct most of the problems with such legacy systems, and provides a vehicle for future changes to the filesystem layout or to the installed operating system.

3.5.2 Deployment

The first testbed nodes were deployed in February, 2001. There are currently 36 active nodes at 31 sites[3], listed in Table 3.1. About 90% of the nodes are up and running at any given time; several nodes, particularly those outside the United States or in unattended data centers, experienced long time-to-repair. These nodes reside in a variety of locations, from private residences with DSL connections, to large, well-connected companies and schools. Table 3.2 shows the breakdown of sites by the type of institution hosting the node, and Table 3.3 shows the types of network connections.

Seven testbed sites provide the RON nodes full BGP feeds from their border routers. An additional site provides customer routes.

Figure 3-1 also plots the locations of the RON nodes deployed in the U.S.. The network topology snapshot in Figure 3-2 shows that there are a large number of AS-level paths available to the testbed nodes, even when examining only 12 of the hosts. The network topology is derived from

[3]sg suffers firewall problems that make it only partly available to researchers.

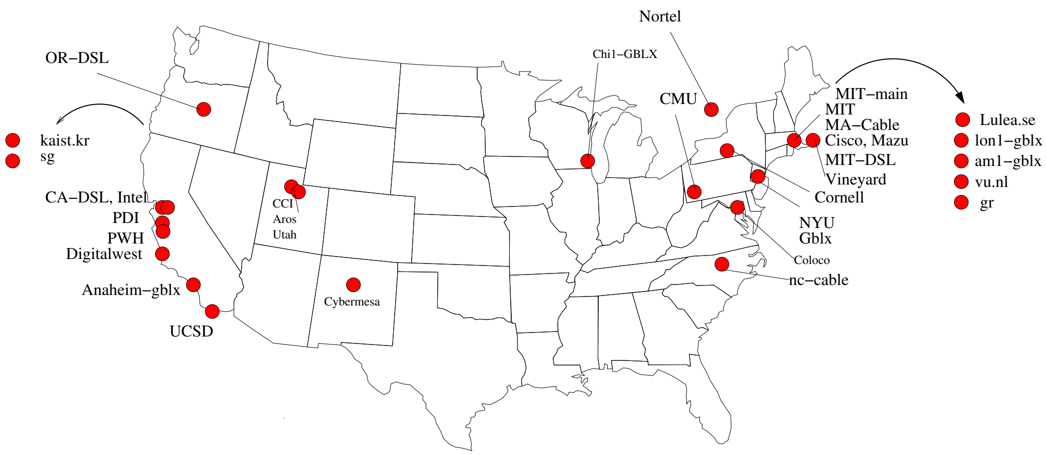


Figure 3-1: The locations of the RON testbed nodes

Site type	Number
US Private Residence	3
US Company	13
US University	6
International Company	3
International School	5

Table 3.2: RON Testbed site breakdown.

Connection type	Number
DSL	1
Cable Modem	2
T1	2
$\leq$ T3	3
100 Mbits/s *	8
OC3+	6
International	8

Table 3.3: Connectivity breakdown. Some sites listed as 100 Mbits/s may have T3s or fractional T3s. All International links are high speed to a local backbone, but connection speeds to the rest of the world vary.

`traceroute`[4] data, examining one path from each source to each destination. The network topology graph in Figure 3-2 actually understates the complexity of the network somewhat, because the connectivity varies with time, and this is a static snapshot; however, it provides a reasonable indication of the complexity of the underlying network.

3.5.3 Management

The nodes are managed through the Netbed [174] testbed. Users apply for accounts at Netbed, and request access to the MIT RON testbed nodes. If granted, the users are given accounts on all of the testbed nodes. Software updates and node installation are handled through a Netbed-based software update and CD-ROM based boot system.

Access to the testbed is granted only to "mostly trusted" researchers. The testbed is a research tool only; we have not invested time in sophisticated isolation techniques for untrusted users. Our goal is to keep the population of testbed users large enough to ensure that the testbed is useful, but small enough that a well-defined Acceptable Use Policy (AUP) and social pressure can ensure good behavior.

3.5.4 Data Collection

As a basic service on the RON nodes, we collect periodic latency and loss samples, topological information using `traceroute`, and, at some sites, BGP routing data. Unlike the RON system itself, this general measurement service on the testbed does not use passive data collection or permit users to insert their own measurements.

Roughly once per second (the exact period is randomized), every node sends a small UDP packet to another randomly selected node, which echoes the packet back. These samples provide latency and loss information. If our probing daemon observes multiple losses to a particular host, it initiates a traceroute to record the path along which the failure occurred. Once per day, we traceroute from every node to every other node to provide a topology snapshot. We have established a BGP peering session with six of our hosts border routers, and we record and archive all BGP routing updates received at the hosts for later analysis. This data is freely available upon request.[5]

3.6 Closely Related Testbeds

Emulab, the Netbed local cluster, provides a "network in a bottle" via a centralized, reconfigurable cluster of nodes [174]. Researchers can create networks of up to 100 nodes with arbitrary bandwidth and delays between the nodes. By using the same Tcl-based configuration as the popular network simulator *ns* [162], Emulab simplifies the transition from simulated experiments to running real code in an emulated environment. Emulab users can wipe disks, install operating systems, and test non-IP protocols. The price of this power is that researchers cannot test their protocols in a real wide-area environment. We now use the Netbed facilities to manage the RON testbed, providing a complete path from simulation to emulation to wide-area testing.

Planetlab is a "global overlay network for developing and accessing new network services." [122] Planetlab aims to grow to thousands of distributed nodes that can be used for both research *and*

[4] `traceroute` is a program that determines the path taken between two Internet hosts, one hop at a time. It sends packets that expire en-route to their destination, generating an error message from the router that terminated them. The first packet expires after one hop, the second after two, and so on, resulting in a list of routers through which the data has traveled.

[5] Our BGP Web site, `http://bgp.lcs.mit.edu/` provides an interface to obtain snapshots of the data.

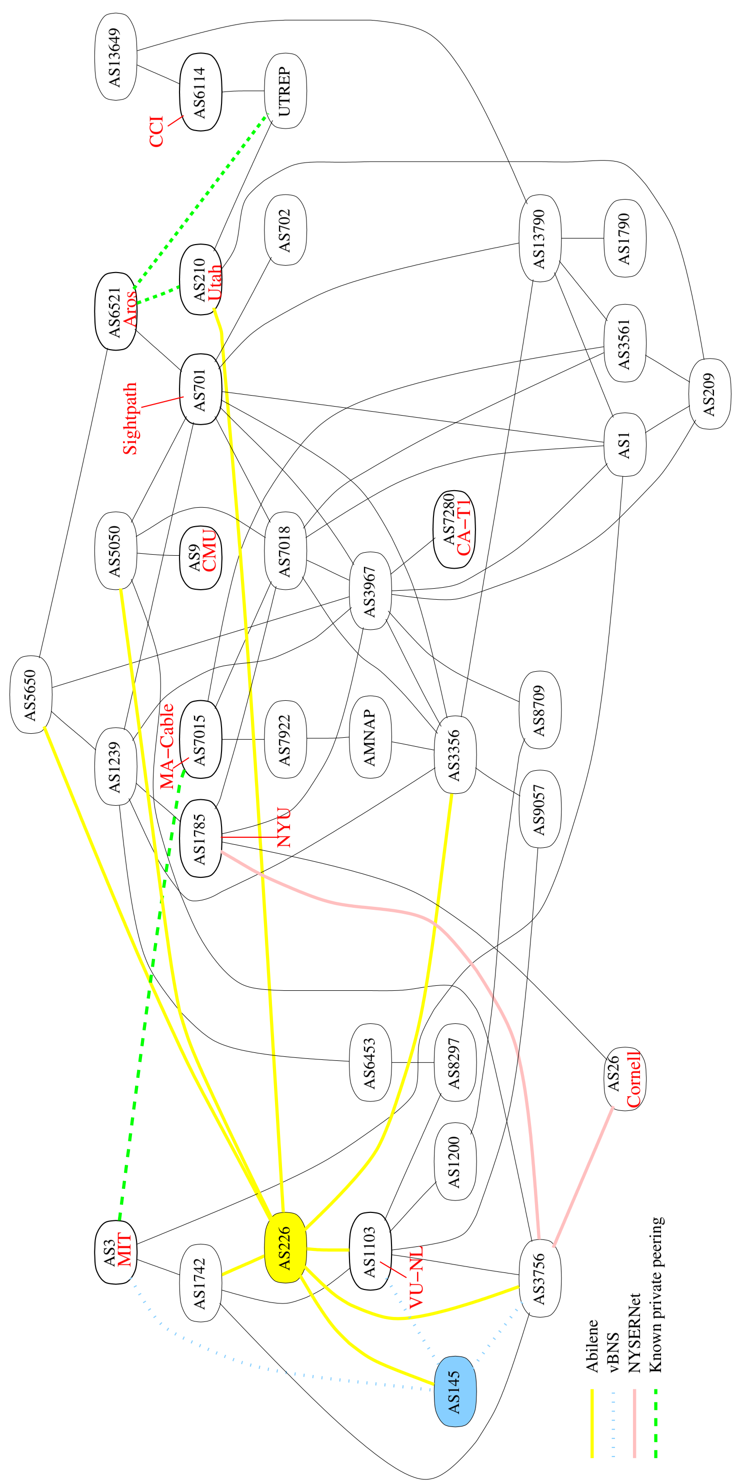


Figure 3-2: The AS-level connectivity graph for 12 of the RON nodes. The paths connecting these 12 sites traverse 36 different autonomous systems, presenting a rich set of paths from which to choose, even between a small subset of the testbed. Known research networks and private peering links are highlighted.

long-term service deployment. Like RON testbed nodes, Planetlab nodes are more autonomous than Emulab nodes, running a single base operating system image. Planetlab nodes can be deployed alone or in local clusters.

Planetlab diverges from the RON testbed in its goal of being open to service development and deployment by (eventually) almost anyone. The RON testbed is primarily an experimentation platform, and researchers who wish to provide services to the Internet are encouraged to migrate their services elsewhere. In contrast, Planetlab embraces the idea of being a substrate for future Internet services. The cost of this openness is some loss of control by researchers, who give up more privileged access to the nodes for security. We hope to eventually host PlanetLab services on the RON nodes, while still making the nodes available for lower-level experiments.

3.7 Summary

The MIT RON testbed is an ongoing research artifact that has been useful to many projects. We believe that its success is attributable in large part to its diverse network connectivity, part, has been due to its diverse network connectivity, reasonable size, and easy to use, homogenous interface. The development and ongoing maintenance of this testbed presented us with a number of challenges, and we hope that the design principles and lessons we have learned will benefit the designers of future testbeds.

The rest of the research presented in this dissertation makes extensive use of measurements obtained using the RON testbed in various stages of development. Whenever possible, the traces taken on the testbed are publicly available.

Do what you can, with what you have, where you are.

- Theodore Roosevelt

Chapter 4

Resilient Overlay Networks

A *Resilient Overlay Network* (RON) is an overlay network that allows distributed Internet applications to detect and recover from path outages and periods of degraded performance within several seconds.

RON is intended to improve availability for a class of Internet applications characterized by small groups of hosts communicating with each other as part of a distributed application. Examples include conferencing, remote login and management, connections between data centers, and virtual private networks. The RON software can be used stand-alone within an application, or arbitrary, unmodified application traffic can be encapsulated and forwarded within a RON using the RON IP Tunnel application described in Section 4.8.

Figure 4-1 shows the design of RON. RON nodes cooperatively forward packets for each other to avoid failures in the underlying Internet, forming an application-layer overlay network. RON nodes are deployed at various spots in the Internet. From there, each RON node monitors the quality of the underlying Internet paths between it and the other nodes, and uses this information to intelligently select paths for packets. Each direct Internet path between two nodes is called a *virtual link*. To discover the topology of the overlay network and to obtain information about virtual links in the topology to which it is not directly connected, each RON node participates in a routing protocol to exchange information about a variety of quality metrics that characterize the different virtual links.

4.1 Design Goals

The design of RON meets three principal design goals:

1. Failure detection and recovery in less than 20 seconds for networks of up to 40 or 50 nodes;
2. Tighter integration of routing and path selection with the application; and
3. Expressive policy routing.

Each of these goals is discussed below.

4.1.1 Fast Failure Detection and Recovery

As noted earlier, today's BGP-based wide-area Internet routing system does not protect applications from outages for many minutes, and does not react to performance failures that can significantly degrade an application's utility.

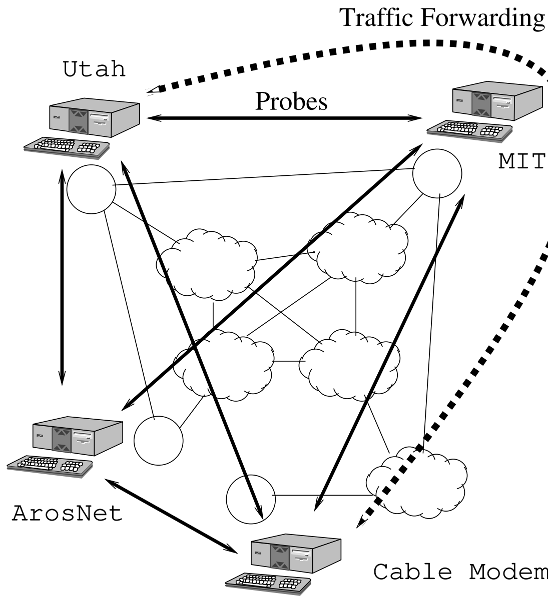


Figure 4-1: The general approach used in the RON system. Nodes send probes to determine the network characteristics between each other. Using their knowledge of the network, they potentially forward traffic through other nodes. In this example, traffic from Utah to the Cable Modem site is sent indirectly via MIT.

RON's goal is to detect and recover from outages and performance failures within 20 seconds, regardless of their cause, assuming that an alternate path exists. As noted in Chapter 1, Internet routing often requires three minutes to circumvent failures. RON's primary design goal is to improve this recovery time by an order of magnitude.

RON is aimed at "community" applications in which other nodes are willing to contribute their own network resources in order to ensure that other nodes in the community can reach each other. These applications include both group communication applications such as audio and video conferencing, and applications such as virtual private networks in which communication occurs between pairs, but the other members of the community have incentives to assist their peers. The number of participants in such applications is often small—under 50—and RON takes advantage of these small sizes to more aggressively detect and mask failures.

4.1.2 Tighter Integration with Applications

Failures and faults are application-specific notions: network conditions that are fatal to one application may be acceptable to a more adaptive application. For instance, a UDP-based Internet audio application not using good packet-level error correction may not work at all at loss rates larger than 10%. At this loss rate, a bulk transfer application using TCP will continue to work because of TCP's adaptation mechanisms, albeit at lower performance. However, at loss rates of 30% or more, TCP becomes essentially unusable because it times out for most packets [109]. RON allows applications to independently define and react to their own *path metrics* that describe the quality of an Internet path.

Path metrics include latency, loss, throughput, and combinations thereof. Applications may prefer different metrics or combinations of metrics (*e.g.*, latency over throughput, or low loss over latency)

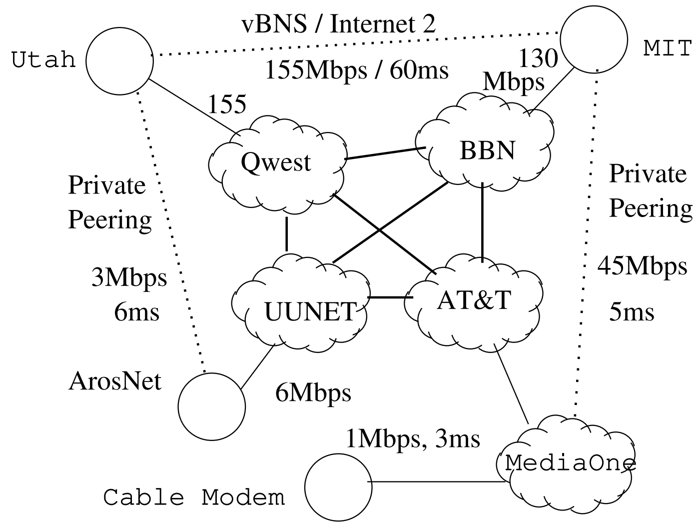


Figure 4-2: Internet interconnections are often complex. The dotted links are *private* and are not announced globally.

in their path selection. A routing system may not be able to optimize all of these metrics simultaneously; for example, a path with a one-second latency may appear to be the best throughput path, but this degree of latency may be unacceptable to an interactive application. RON allows application writers to easily construct their own metrics based upon their own measurements and have the routing system make decisions based upon those metrics.

4.1.3 Expressive Policy Routing

The design of RON is intended to enable users to allow or deny certain types of traffic on arbitrary links (*e.g.*, "no commercial traffic between Utah and MIT") and to enable the RON application to make a decision about what type of traffic each flow contains (*e.g.*, tag Web browsing traffic as commercial, but email as non-commercial). Despite the need for policy routing and enforcement of network use policies (*e.g.*, the Internet2 policy permitting only non-commercial traffic) and other policies that affect the flow of traffic, today's approaches are primitive and cumbersome. For instance, BGP-4 is incapable of expressing fine-grained policies aimed at users or remote hosts. This lack of fine-grained policy control not only reduces the set of paths available in the case of a failure, but it also inhibits innovation in the use of carefully targeted policies, such as end-to-end per-user rate controls or enforcement of acceptable use policies (AUPs) based upon actual traffic content.

Figure 4-2 shows the AS-level network connectivity between four of our measurement hosts. In addition to showing the considerable underlying path redundancy available in the Internet—the reason RON has the potential to meet our goals—the figure shows situations where BGP's blunt policy expression inhibits fail-over. For example, if the Aros-UUNET connection failed, users at Aros would be unable to reach MIT even if they were authorized to use Utah's network resources to get there. This restriction of network service arises because it impossible to announce a BGP route only to particular users, so the Utah-MIT link is available only to hosts directly in those domains.

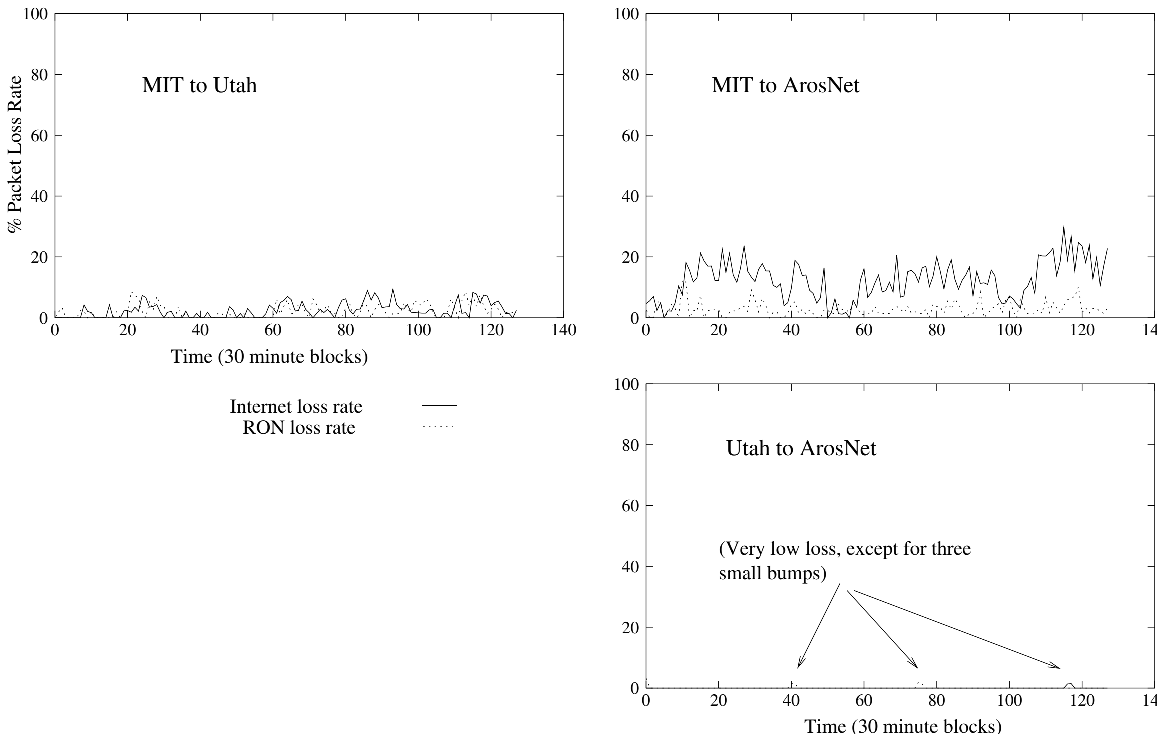


Figure 4-3: Results from the preliminary RON study of loss rates. The three graphs show the loss rates with and without indirect routing on the three paths between MIT, Utah, and ArosNet. Indirect routing successfully combines the low loss paths from ArosNet to Utah and from Utah to MIT to yield better performance than the direct path from ArosNet to MIT.

4.2 Exploratory Study Results

Before designing and implementing the full RON architecture, we conducted a preliminary study examining the latency and loss of packets sent directly and indirectly between the four nodes shown in Figure 4-2, which were connected via the University of Utah, MIT, an ISP in Salt Lake City, Utah named ArosNet, and a cable modem in Cambridge, Massachusetts. Each node measured the latency and loss to other nodes using `tcping`, a TCP-based ping utility that we created for this purpose. A periodic process sent back-to-back TCP bulk transfers between the nodes. One of the transfers went directly over the Internet, and the other went via the best path as indicated by previous `tcping` probes.

Figures 4-3 and 4-4 show the results of this preliminary study of 62 hours of TCP probes between the 4 hosts in our measurement study with and without indirect routing. The study was performed between 8 January and 11 January 2001. In 92% of the samples, the latency of the packets sent with indirection was better than the direct path latency. The average latency over the measurement period was reduced from 97ms to 68ms. The latency benefits arose somewhat from using the high speed Internet2 path between Utah and MIT, but came even more from avoiding a particularly slow connection between the cable modem ISP and the University of Utah's ISP, which was often routed through Seattle.

These preliminary results, along with earlier indications from the Detour study [140], gave us strong reasons to believe that RON would be effective in routing around Internet failures and paths that ex-

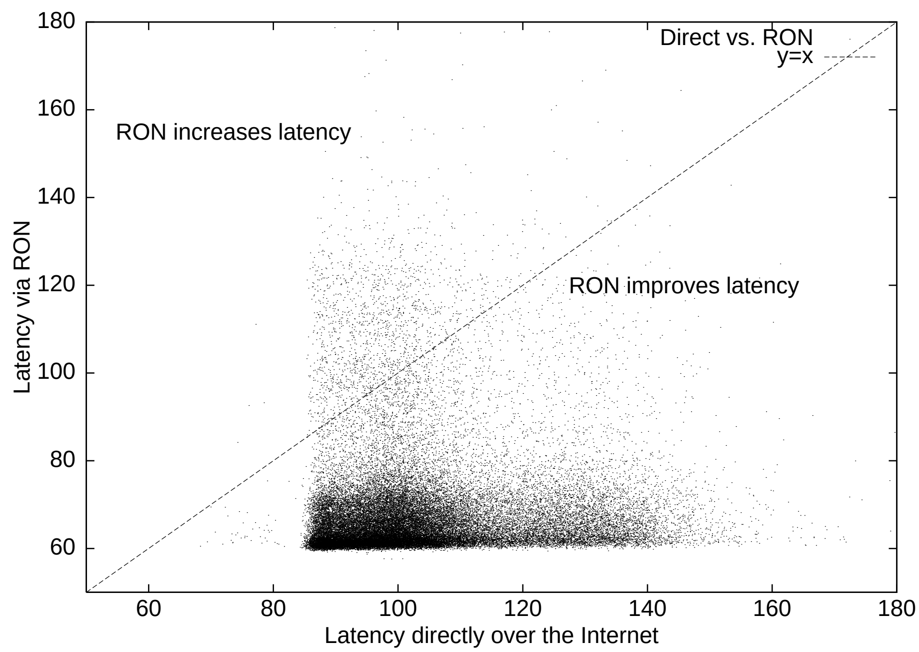


Figure 4-4: A scatterplot of the latency between the RON nodes in our preliminary study. In 92% of the samples, intelligently selecting an indirect path could reduce the latency between the nodes.

hibited degraded performance. As a result, we designed and implemented the software architecture described in this chapter. We present the results of a larger evaluation study of the resulting system in Chapter 5.

4.3 Software Architecture

The RON system architecture is shown in Figure 4-5. Data enters the system from one or more programs (*RON applications*) that communicate with the RON software on a node. The overlay network is defined by a single group of hosts that collaborate to provide a distributed service or application. This group of hosts can use application-specific routing metrics when deciding how to forward packets in the group. Our design accommodates a variety of RON applications, ranging from a generic IP packet forwarder that improves the reliability of IP packet delivery, to a multi-party conferencing application that incorporates application-specific metrics in its route selection.

Figure 4-6 shows RON's software architecture. The RON applications interact with RON via the *conduit* API, which permits them to send and receive packets. The probing, routing, and membership management components are all themselves RON applications that register to receive packets of particular types. A conduit can be interposed between other conduits to perform additional packet processing. For example, a debugging conduit can be placed between the application and the forwarder to log information about the traffic it observes, or a traffic shaping conduit can restrict the rate at which an application can send data.

On the data path, the first node to receive a packet classifies it to determine the type of path it should use (*e.g.*, low-latency, high-throughput, etc.). This *entry node* determines a path from its topology table, encapsulates the packet into a RON header, tags it with some information that simplifies forwarding by downstream RON nodes, and forwards it on. Each subsequent RON node simply

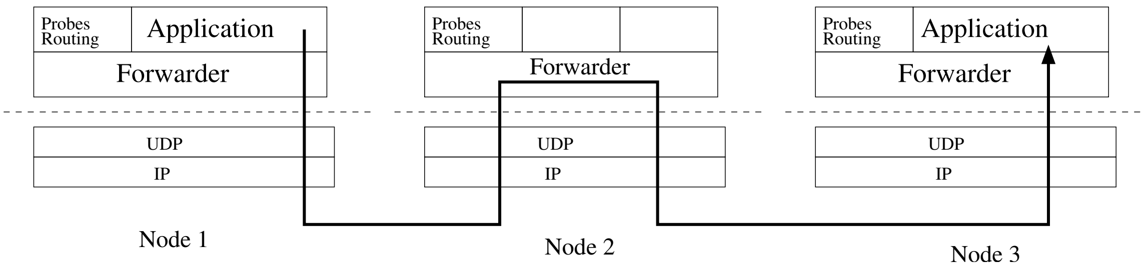


Figure 4-5: Data enters the RON through a RON application, which may be a full-fledged client program or a proxy that gathers external traffic and takes it into the RON. Path selection is performed at the entry node. Forwarding at the intermediate node does not require any interaction with the application; in this manner, an intermediate RON node can forward traffic on behalf of multiple applications. The communication between the probing and routing components is not shown in this figure.

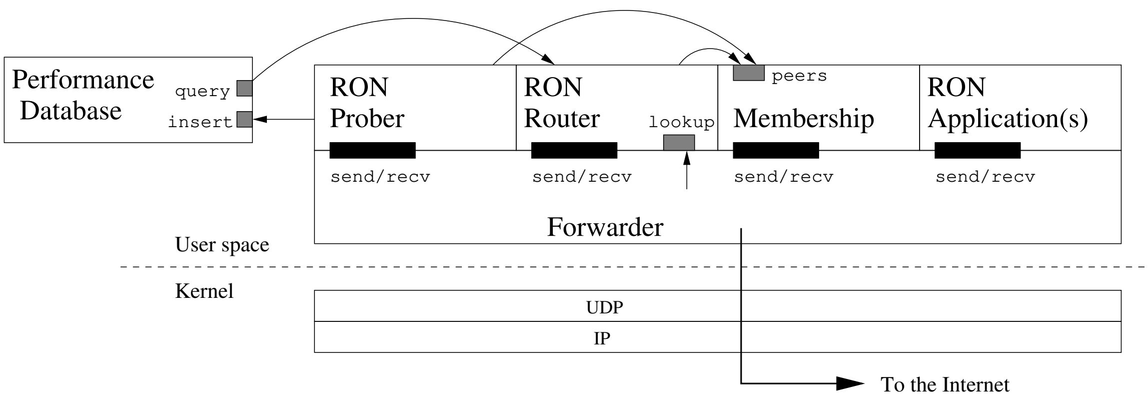


Figure 4-6: The RON software system architecture for a single host. The *forwarder* brokers all communication with the network. The probing, routing, membership management, and application data are all relayed through the forwarder. There are a small number of extra interfaces: the router provides a "lookup" function to the forwarder; the probing and routing systems interact with the performance database and the membership manager.

determines the next forwarding hop based on the destination address and the tag. The final RON node that delivers the packet to the RON application is called the *exit node*.

The conduit API consists of three functions:

1. `send(packet p, dest node d, boolean via_ron)` delivers a packet from the upstream conduit to the downstream conduit. The source may specify whether or not the packet should be sent using the direct Internet path, or whether the RON system may redirect it through an alternate path. RON's delivery, like UDP, is best-effort and unreliable. Section 4.6 explains the RON packet header.

2. `recv(packet p, boolean via_ron)` is a callback function that is called by the forwarder (or a subsequent conduit) when a packet arrives. This callback is invoked after the RON conduit matches the type of the packet in the RON header to the set of types pre-

registered by the client when it joins the RON. The RON packet type is a demultiplexing field for incoming packets.

3. `register_type(type)` requests that packets of a particular type be delivered to the caller. For example, the default RON routing manager registers to receive all packets of type `RON_ROUTING`. Each packet type is delivered to at most one recipient.

The RON `forwarder` ties together the RON functions and implements versions of the above functions that send and receive packets to and from the network. It also provides a timer registration and callback mechanism to perform periodic operations, together with a similar service for network socket data availability.

Each client must instantiate a forwarder and hand to it two modules: a *RON router* and a *RON membership manager*. The RON router implements a routing protocol. The RON membership manager implements a protocol to maintain the list of members of a RON. By default, RON provides a few different RON router and membership manager modules for clients to use. Most RON routers retrieve information about Internet path performance from a local *performance database* that tracks the results of path probes.

RON routers and membership managers exchange packets using RON as their forwarding service, rather than over direct IP paths. This feature is beneficial because it allows the control messages to be forwarded even when some underlying IP paths fail.

In our design, RON applications must run in the same address space as the RON forwarder. A particular RON application called a *shim* can broker access to the forwarder to applications running in separate address spaces (or even other machines) by exporting the same `send/recv` interface via the network. An example of a shim application is the divert mechanism created by Nakao *et al.* to permit a group of desktop nodes to tunnel their traffic through the same RON node [105].

4.4 Routing and Path Selection

Routing is the process of building up the forwarding tables that are used to choose paths for packets. In RON, the entry node has more control over subsequent path selection than in traditional datagram networks. Future RON routers could use the flow identifier to divide flows when performing multi-path routing and could use the flow ID to optimize packet forwarding lookups. Providing flow IDs also permits a RON system to provide rate control on a per-flow basis. Without some concept of a flow, multi-path routing can re-order packets in ways that lead to performance degradation. The entry node *tags* the packet's RON header with an identifier that identifies the flow to which the packet belongs. By tagging packets at the entry node, the application is given maximum control over what the network considers a "flow."

The small size of a RON relative to the Internet allows it to maintain information about multiple alternate routes and to select the path that best suits the RON client according to a client-specified routing metric. By default, it maintains information about three specific metrics for each virtual link: (i) latency; (ii) packet loss rate; and (iii) throughput. RON clients can override these defaults with their own metrics, and the RON library constructs the appropriate forwarding table to pick good paths. The router builds up forwarding tables for each combination of policy routing and chosen routing metric.

4.4.1 Link-State Dissemination

The default RON router uses a link-state routing protocol to disseminate topology information between routers, which is in turn used to build the forwarding tables. Each node in an N-node RON has $N-1$ virtual links. Each node's router periodically requests from its local performance database a summary of the virtual link quality metrics to its $N-1$ peers. The node then disseminates these values to the other nodes.

Link-state information is sent via the RON forwarding mesh itself, to ensure that routing information is propagated in the event of path outages and heavy loss periods. Thus, the RON routing protocol is itself a RON client, with a well-defined RON packet type. This leads to an attractive property: The only time a RON router has incomplete information about any other one is when *all* paths in the RON from the other RON nodes to it are unavailable.

4.4.2 Path Evaluation and Outage Detection

Every RON router implements *outage detection*, which it uses to determine whether the virtual link between it and another node is still working. The routers use an active probing mechanism for this task—if the last `OUTAGE_THRESH` probe packets were all lost, then an outage is flagged. Our implementation sets this outage threshold to four packets. Paths experiencing outages are rated on their packet loss rate history; a path having an outage will always lose to a path not experiencing an outage. The `OUTAGE_THRESH` and the frequency of probing (`PROBE_INTERVAL`) permit a trade-off between overhead time and the bandwidth consumed by the probing. With N nodes, each node sends $N/$`PROBE_INTERVAL` probe packets per second, and (on average) detects an outage in $\frac{1}{2}$`PROBE_INTERVAL`$+4\times$`PROBE_TIMEOUT` seconds. This trade-off is discussed further in Section 5.2.

RON chooses between working paths using a set of *metric evaluators* that provide a number quantifying how "good" a path is along some axis. These numbers are *relative*, and are only compared to other numbers from the same evaluator. The two important aspects of path evaluation are the mechanism by which the data for two links are combined into a single path, and the formula used to evaluate the path.

By default, every RON router implements three different routing metrics: the latency-minimizer; the loss-minimizer; and the throughput-optimizer. The latency-minimizer forwarding table is computed by computing an exponential weighted moving average (EWMA) of round-trip latency samples with parameter α. For any link l, its latency estimate lat_l is updated as:

$$lat_l \leftarrow \alpha \cdot lat_l + (1-\alpha) \cdot new\_sample_l \tag{4.1}$$

We use $\alpha = 0.9$, which means that 10% of the current latency estimate is based on the most recent sample. This number is similar to the values suggested for TCP's round-trip time estimator [123]. For a RON path, the overall latency is the sum of the individual virtual link latencies: $lat_{path} = \sum_{l \in path} lat_l$.

To estimate loss rates, RON uses the average of the last $k = 100$ probe samples as the current average. Like Floyd *et al.* [53], we found this to be a better estimator than EWMA, which retains some memory of samples obtained in the distant past as well. It might be possible to further improve our estimator by unequally weighting some of the k samples [53]. Unlike Floyd *et al.*, RON uses loss estimates only to compare links, not to dictate transmission rates, so the relative accuracy of comparing two estimates is more important than the absolute values.

Loss metrics are multiplicative on a path: if one assumes that losses are independent, the probability of success on the entire path is roughly equal to the probability of surviving all hops individually: $lossrate_{path} = 1 - \Pi_{l \in path}(1 - lossrate_l)$.

RON does not attempt to find optimal throughput paths, but strives to avoid paths of low throughput when good alternatives are available. Given the time-varying and somewhat unpredictable nature of available bandwidth on Internet paths [16, 116], we believe this goal is appropriate. From the standpoint of improving the reliability of path selection in the face of performance failures (Chapter 3), avoiding bad paths is more important than optimizing to eliminate small throughput differences between paths. While a characterization of the utility received by programs at different available bandwidths may help determine a good path selection threshold, we believe that more than a 50% bandwidth reduction is likely to reduce the utility of many programs. This threshold also falls outside the typical variation observed on a given path over time-scales of tens of minutes [16]. RON's throughput optimizer is therefore designed to avoid throughput penalties on this order of magnitude.

Throughput-intensive applications typically use TCP or TCP-like congestion control, so the throughput optimizer focuses on this type of traffic. The performance of a bulk TCP transfer is a function of the round-trip latency and the packet loss rate it observes. RON's throughput optimization combines the latency and loss metrics using a simplified version of the TCP throughput equation [109], which provides an upper-bound on TCP throughput. RON's granularity of loss rate detection is 1%, and the throughput equation is more sensitive at lower loss rates. The metric uses a minimum packet loss rate of 2% to prevent the router from estimating an artificially high throughput metric and to prevent large route oscillations from isolated packet losses. The formula used is:

$$score = \frac{\sqrt{1.5}}{rtt \cdot \sqrt{p}} \tag{4.2}$$

where p is the *one-way* packet loss probability and rtt is the end-to-end round-trip time estimated from the hop-by-hop samples as described above. The TCP throughput equation assumes that the loss and round-trip statistics are derived from a bulk TCP transfer that could completely fill the link. Therefore, the $p = 0$ and $p = 1$ cases cannot happen. RON's throughput estimator first performs outage avoidance, ensuring that the path is not used when $p = 1$. The minimum packet loss mentioned above avoids the case of $p = 0$.

The non-linear combination of loss and throughput makes this scoring difficult to optimize with standard shortest-path algorithms; for instance, the TCP throughput between nodes A and C via a node B is lower than either of the TCP throughputs between A and B and B and C[1]. While more complicated search algorithms can be employed to handle this, RON routing currently takes advantage of the resulting simplicity when only single-intermediate paths are considered to obtain throughput-optimized paths.

RON's probe samples provide the *two-way* packet loss probability, from which we estimate the one-way loss probability by (unrealistically, for some paths) assuming symmetric loss rates and solving the resulting quadratic equation. Assuming that losses are independent on the two paths $A \rightarrow B$ and $B \rightarrow A$, then the maximum absolute error in the one-way loss rate estimate occurs when all of the loss is in only one direction, resulting in an error equal to $\frac{loss_{two-way}}{2}$. Assuming a minimum loss rate of 2% ensures that, when choosing between two high-quality links, our loss rate estimate is within 50% of the true value. At large loss rates, highly asymmetric loss patterns cause this method

[1] TCP's throughput is inversely proportional to the round-trip time between the sender and receiver, and independent losses on the two links can reduce the throughput further.

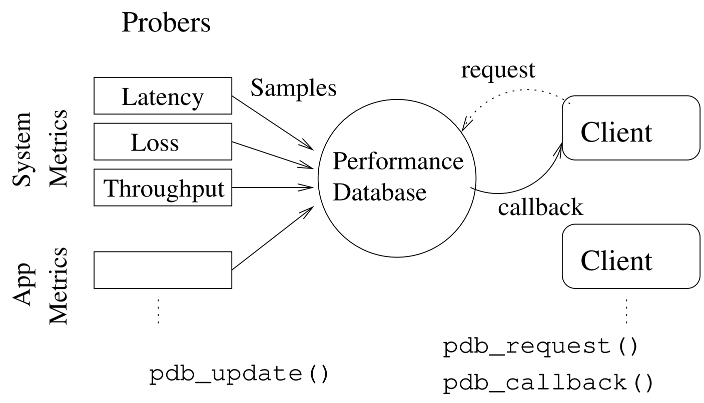


Figure 4-7: The RON performance database.

to disregard a potentially good path because the reverse direction of that path has a high loss rate. While the benefits of avoiding the high loss path rate typically outweigh the cost of missing one good link, a future version of the router should take asymmetric loss into account.

Throughput comparison using Equation 4.2 is not without weaknesses. For instance, a slow but unused path rarely loses the RON probe packets, causing the equation to predict an unreasonably high bandwidth estimate. Improved ways to tackle this problem include using a better model for predicted TCP throughput [61], and using path inference techniques to more accurately infer either the link capacities [79] or the available bandwidth [158, 73].

Oscillating rapidly between multiple paths is harmful to applications sensitive to packet reordering or delay jitter. While each of the evaluation metrics applies some smoothing, this is not enough to avoid "flapping" between two nearly equal routes: RON routers therefore employ smoothing hysteresis. Based on an analysis of 5,000 snapshots from a RON node's link-state table, we chose to apply a simple 5% hysteresis bonus to the "last good" route for the three metrics. This simple method appears to provide a reasonable trade-off between responsiveness to changes in the underlying paths and unnecessary route flapping, as we show in Section 5.4.2.

4.4.3 Performance Database

To make good routing decisions RON must have detailed performance information. It is impractical to send large performance histories to all participants in the RON. A reliable performance repository must handle participants that crash, reboot, or rejoin the RON. Measurement data is often noisy, and different clients may have different ways in which they will use that data; for instance, an outage detector may want to know if any packets were successfully sent in the last 15 seconds, but a throughput optimizer may be interested in a longer-term packet loss average. Therefore, the system needs a flexible *summarization* mechanism.

To support these requirements, each RON node or local group of nodes uses a separate *performance database* to store samples (Figure 4-7). The database is a generalization of SPAND [142], supporting different data types and summarization mechanisms. Sources of information and clients that use it agree on a *class name* for the data and the meaning of the values inserted into the database. The probes insert data into the database by calling `pdb_update(class, src, dst, type, value)`. The `src` and `dst` fields contain the IP addresses of the sampled data. The class defines the data type of the value; in practice, all performance database clients use a fixed-point data representation for all entries.

4.5 Policy Routing

RON allows users or administrators to define the types of traffic allowed on particular network links. In traditional Internet policy routing, "type" is typically defined only by the packet's source and destination addresses [28]; RON generalizes this notion to include other information about the packet. RON separates policy routing into two components: *classification* and *routing table formation*.

The application that inserts a packet into the RON classifies the packet and assigns to it a policy tag; this tag is used to perform destination-based lookups in the appropriate set of routing tables at each RON router. A separate set of routing tables is constructed for each policy by re-running the routing computation, removing the links disallowed by the corresponding policy. The routing computation computes the shortest path from the RON node to all other nodes, for any given metric. This construction applies to multi-hop or single-hop indirection; because a standard shortest-paths algorithm may not work for all metrics (specifically, TCP throughput), our implementation, described in Section 4.8.3, is specific to single-hop indirection. The result of running the table construction is the multi-level routing tables shown in Figure 4-8.

The construction of these routing tables and the production of packet tags is facilitated by the *policy classifier* component. The policy classifier produces the `permits` function that determines whether a given policy is allowed to use a particular virtual link. It also provides a conduit-specific data classifier module that helps the RON node decide which policy is appropriate for the incoming packet. For example, if packets from a commercial site are not allowed to go across the Internet2 educational backbone, and we received a packet from such a site at an MIT RON node, the data classifier module would determine which policy to use and would set the corresponding tag on the packet. Subsequent RON nodes examine the policy tag instead of reclassifying the packet.

We have designed two policy mechanisms: *exclusive cliques* and *general policies*. In an exclusive clique, only data originating from and destined to other members of the clique may traverse inter-clique links. This mimics, for instance, the "educational only" policy on the Internet2 backbone. This policy classifier uses as input a list of addresses (with subnet masks) that fall within the clique.

RON's general policy component is more powerful; it accepts a BPF (Berkeley Packet Filter)-like packet matcher [96], and a list of links that are denied by this policy. It returns the first policy that matches a packet's fields to the stored BPF-based information. Chapter 9 discusses possible abuse of AUPs and network transit policies.

4.6 Data Forwarding

The forwarder at each RON node examines every incoming packet to determine whether it is destined for a local client or a remote destination. If it requires further delivery, the forwarder passes the RON packet header to the routing table, as shown in Figure 4-8.

The RON packet header is shown in Figure 4-9. It is inspired by the design of IPv6 [114]. Because RON needs to support more than simply IP-in-IP encapsulation, RON uses its own header. RON does not fragment packets, but does inform applications if they exceed the Maximum Transmission Unit (MTU). As in IPv6, applications must perform end-to-end path MTU discovery. RON also

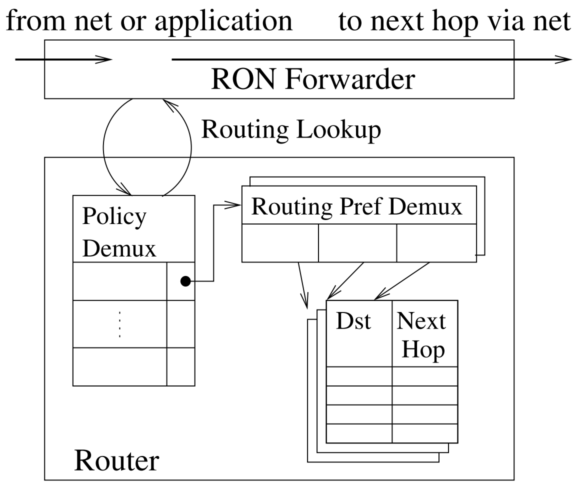


Figure 4-8: The forwarding control and data paths. The forwarder passes the packet header to the routing table, which performs three levels of lookups. The first level is based upon the policy type; the second is based upon the routing preference, and this lookup leads to a hash table of next hops indexed by destination.

provides a policy tag that is interpreted by the forwarders to decide which network routing policies apply to the packet. If the packet is destined for the local node, the forwarder uses the packet type field to demultiplex the packet to the RON client.

If the packet's flow ID has a valid flow cache entry, the forwarder short-cuts the routing process with this entry. Otherwise, the routing table lookup occurs in three stages. The first stage examines the policy tag, and locates the proper routing preference table. There is one routing preference table for each known policy tag. Policy adherence supersedes all other routing preferences. Next, the lookup procedure examines the routing preference flags to find a compatible route selection metric for the packet. The flags are examined from the right, and the first flag understood by the router directs the packet to the next table. All RON routers understand the basic system metrics of latency, loss, and throughput; this provides a way for a shared RON environment to support users who may also have

<table>
<tr><td>Version</td><td>Hop Limit</td><td colspan="2">Routing Flags</td></tr>
<tr><td colspan="4">RON Source Address</td></tr>
<tr><td colspan="4">RON Destination Address</td></tr>
<tr><td colspan="2">Source port</td><td colspan="2">Dest port</td></tr>
<tr><td colspan="4">Flow ID</td></tr>
<tr><td colspan="4">Policy Tag</td></tr>
<tr><td colspan="4">Packet Type</td></tr>
</table>

Figure 4-9: The RON packet header. The routing flags and flow ID are set by the RON application. The packet type is a demux key indicating the appropriate conduit or protocol at the receiver.

client-defined metrics, and still provide them with good default metrics. A lookup in the routing preference table leads to a hash table of next-hops based upon the destination RON node. The entry in the next-hop table is returned to the forwarder, which places the packet on the network destined for the next hop.

4.7 Bootstrap and Membership Management

In addition to allowing clients to define their own membership mechanisms, RON provides two system membership managers: a simple static membership mechanism that loads its peers from a file, and a dynamic announcement-based, soft-state membership protocol. To bootstrap the dynamic membership protocol, a new node needs to know the identity of at least one peer in the RON. The new node uses this neighbor to broadcast its existence using a *flooder*, which is a special RON client that implements a general-purpose, resilient flooding mechanism using a RON forwarder.

The main challenge in the dynamic membership protocol is to distinguish between a node departing the RON and a node experiencing a severe path outage. Each node builds up and periodically (every five minutes on average in our implementation) floods to all other nodes its list of peer RON nodes. If a node has not heard about a peer in 60 minutes, it assumes that the peer is no longer participating in the RON. Observe that this mechanism allows two nodes in the same RON to have a non-functioning direct Internet path for long periods of time: as long as there is some functioning indirect path between the two nodes, neither will think that the other has left.

The overhead of flooding memberships is small compared to the traffic caused by active probing and routing updates. The resulting robustness is high: each node receives up to $N-1$ copies of the peer list sent from a given other node. This redundancy causes a node to be deleted from another node's view of the RON only if the former node is genuinely partitioned for over an hour from *every other* node in the RON. A RON node will re-bootstrap from a well-known peer after a long partition.

To improve the chances of reconnecting to a RON after a period of disconnectivity, each node maintains a cache of the last full set of nodes in the RON. Upon restarting (*e.g.*, after a reboot or process restart), the node sends its join message to each of the last nodes, requesting that they rebroadcast the message. It does not, however, add these nodes into its list of peers until it hears back from them.

4.8 Implementation and the RON IP Tunnel

We implemented a RON prototype at user-level as a set of C++ libraries to which user-level RON applications link. Each application can pick and choose the components that best suits its needs. To provide a specific example of an application using the RON library, we describe the implementation of a resilient IP forwarder called the *RON IP tunnel*. The tunnel takes IP packets off of the wire, sends them via RON, and then transmits them from the exit RON node to their destination(s). This application improves IP packet delivery without any modification to the transport protocols and applications running at end-nodes. The architecture of the RON IP tunnel is shown in Figure 4-10.

The RON prototype uses UDP to forward data. TCP's reliable byte-stream is mismatched to many RON clients: running a TCP stream over another TCP stream causes problems between the interacting control loops. IP-based encapsulation would require that the RON libraries be able to send and receive raw IP packets, which would restrict RON's ability to be deployed as a part of a standard program. The prototype RON core services run without special kernel support or elevated privileges. The application classifies the packets it sends and labels the packets with information

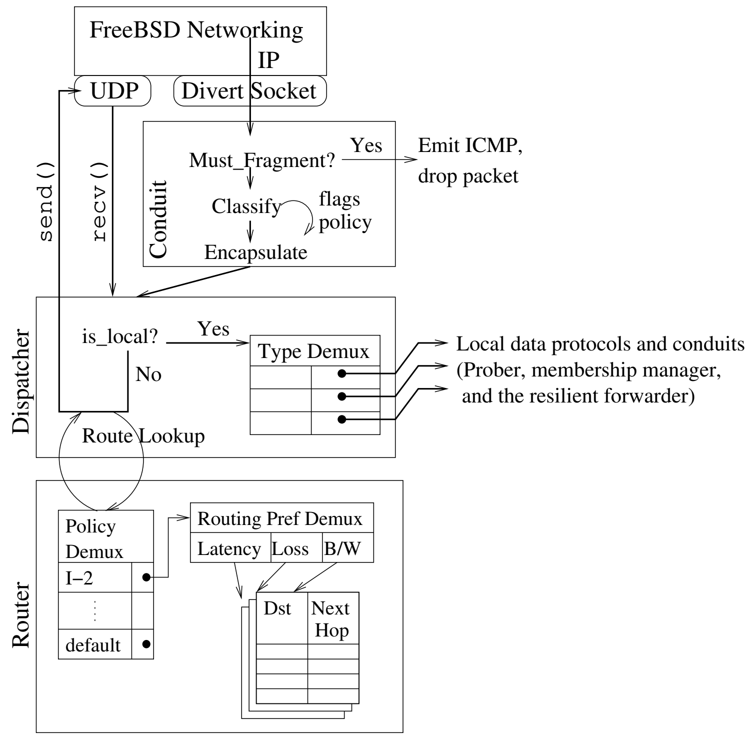


Figure 4-10: The RON IP tunnel. Packets enter the system through FreeBSD's divert sockets. They are handed to the RON forwarder, which looks up the next hop. When the packet arrives at its destination, the conduit writes the packet out a local raw socket, where it re-enters IP processing. The I-2 policy is the Internet2 policy described earlier.

that decides what routing metric will later be used to route the packet. As an example, the RON IP tunnel application might inspect the packets it receives and label DNS and HTTP traffic as "latency-sensitive" and FTP traffic as "throughput-intensive," which would cause the RON forwarders to use the appropriate routing metric for each packet type. Non-entry RON nodes route the packet based only on the attached label and destination of the packet.

This design ensures that all client- and application-specific routing functions occur only at the entry and exit conduits, and that forwarding inside the RON is independent of client-specific logic. In addition to reducing the per-packet overhead at intermediate nodes, it also localizes the client-specific computation required on each packet. This means, for example, that a RON node implementing an IP tunnel may also participate in an overlay with a conferencing application, and help improve the reliability of data delivery for the conference. The conference node conduits implement the application-specific methods that label packets to tell the non-conference nodes which routing metrics to use for conference packets.

4.8.1 Forwarder Interface

The forwarder defines several registration functions through which clients can indicate an interest in receiving particular events (packets of a particular type, notification that a socket is ready, or that a timer has expired). A client that registers an interest in an object must provide a corresponding callback function:

```
router *r = new latency_minimizing_router();
membership *m = new dynamic_membership_manager();

forwarder *f = new forwarder(r, m);
while (1) {
  f->dispatch();
}
```

Figure 4-11: The simplest RON program that forwards packets on behalf of other nodes but does no processing on its own.

Registration function	Callback function
`register_type(int packet_type)`	`recv(packet, boolean via_ron)`
`register_socket(int sockno, void *data)`	`socket_callback(sockno, data)`
`register_timer(int msec, void *data)`	`timer_callback(data)`

The forwarder provides a basic interface for sending data to other RON nodes. The via_ron parameter permits the caller to specify whether or not the packet should be forced to take the direct Internet route or whether the RON forwarder may re-route it.

```
send_packet(packet, dst, boolean via_ron)
```

And finally, to have time to process sockets, packets, and timers, the forwarder requires that the application frequently call its `dispatch()` function, where the actual work performed by the forwarder occurs.

The simplest RON program, one that forwards packets on behalf of other nodes but does no actual processing on its own, can then be constructed by instantiating a router, membership manager, and forwarder, as shown in Figure 4-11.

4.8.2 The RON IP Tunnel

We implemented the IP tunnel using FreeBSD's *divert sockets* [30] to automatically send IP traffic over the RON, and emit it at the other end. Divert sockets permit the application to receive raw IP packets that match particular firewall rules, modify the packets, and retransmit them. The RON IP tunnel provides classification, encapsulation, and decapsulation of IP packets through a special application called the `ip_conduit`. (Top of Figure 4-10).

The tunnel first configures the system's IP firewall to divert applicable packets to the tunnel application. For instance, a tunnel trying to speed up Web browsing within the RON would be interested in traffic to TCP port 80. Thus, the IP tunnel would first install the following two rules in the FreeBSD firewall table:[2]

Source	Destination	Protocol	Src Port	Dst Port	Action
me	any	`TCP`	80	*	`DIVERT to RON`
me	any	`TCP`	*	80	`DIVERT to RON`

[2]The keyword "me" matches any valid local address on the machine.

```
MAKEROUTINGTABLE(POLICIES, METRICS, PEERS)
  foreach P in POLICIES
    foreach M in METRICS
      foreach Dest in PEERS
        foreach Hop in PEERS
          if P.permits(me, Hop)
          AND P.permits(Hop, Dest) {
            sc = M.eval(me, Hop, Dest);
            if sc > best_score {
              best_score = sc;
              next_hop = Hop;
            }
          }
        table[P][M][Dest] = next_hop;
```

Figure 4-12: The algorithm that creates the RON routing table.

The tunnel could also install more selective rules that only divert data when the destination is another member of the RON, in order to reduce the amount of packet processing that would occur. The RON prototype has not yet required this optimization in the situations in which we have deployed it.

Once the firewall rules are installed, the tunnel opens a divert socket to receive the matched packets. Through this mechanism, the tunnel can run as a user-level program instead of as a kernel component, though it must run with root privileges to open the socket. The tunnel then waits on the divert socket using `select`, and receives incoming packets using `recvmsg`. The tunnel wraps the incoming IP packet with a RON packet header and passes control of it to the RON forwarder; from this point, it is handled by the RON routing until it reaches the destination tunnel application. The destination tunnel strips off the RON packet header and sends the IP packet using a raw socket. If the source tunnel cannot find a route to the destination of the packet, it re-emits the packet as a standard IP packet, so as to not interfere with normal Internet communications.

4.8.3 RON Routers

RON routers implement the `router` virtual interface, which has only a single function call, `lookup(pkt *mypkt)`. The RON library provides a trivial static router, and a dynamic router that routes based upon different metric optimizations. The dynamic router is extensible by linking it with a different set of `metric descriptions`. Each metric description provides an evaluation function that returns the "score" of a link, plus a list of metrics that the routing table needs to generate and propagate:

```
class metric_description {
  double evaluate(src, hop, dst)
  metric_list *metrics_needed()
}
```

The creation of the routing table in the RON prototype is specific to single-hop indirection. The RON router uses the algorithm shown in Figure 4-12 to fill in the multi-level routing table at the bottom of Figure 4-10.

4.8.4 Monitoring Virtual Links

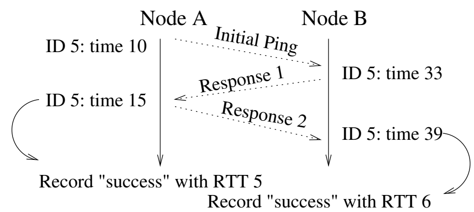


Figure 4-13: The active prober's probing mechanism. With three packets, both participants get an RTT sample, without requiring synchronized clocks.

Each RON node in an N-node RON monitors its $N - 1$ virtual links using randomized periodic probes. The active prober component maintains a copy of a `peers` table with a `next_probe_time` field per peer. When this field expires, the prober sends a small UDP probe packet to the remote peer. Each probe packet has a random 64-bit ID. The process used by the prober is shown in Figure 4-13. When a node receives an initial probe request from a peer, it sends *response 1* to that peer, and resets its probe timer for that peer. When the originating node sees *response 1*, it sends *response 2* back to the peer, so that both sides get reachability and RTT information from 3 packets.

The probing protocol is implemented as a RON client, registering to receive packets of type `RON_PROBES`. The prober communicates with the performance database (implemented as a standalone application running on the Berkeley DB3 back-end) using a simple UDP-based protocol.

The prober contacts each other host every PROBE_INTERVAL seconds plus a random additional offset of $\frac{1}{3}$PROBE_INTERVAL seconds. PROBE_INTERVAL defaults to 12 seconds, so each remote host is probed every $12 + random(\frac{12}{3}) = 14$ seconds on average. If the packet does not come back within the remote host's timeout value, the prober deems it lost and records a loss event. To speed outage detection, when a packet is deemed lost, the prober will immediately send another, up to a maximum of four of these fast probes.

The probe timeout value is computed using an "average RTT plus 3 linear deviations" formula similar to that used to compute TCP timeouts [120], with a one second minimum. This adaptive timeout helps RON's probing adapt to varying network conditions and paths with high or highly variable delay. For most well-connected Internet hosts, the value remains at one second.

With these parameters, the prober will detect an outage within

$$\frac{4}{3}\texttt{PROBE\_INTERVAL} + 4\ \texttt{PROBE\_TIMEOUT}$$

seconds, or about 18 seconds, with an average of 11 seconds.

4.8.5 Membership Management

Membership managers provide a `peers()` function that returns a list of nodes in the RON. The static membership manager, on initialization, reads its peers from a file and then simply returns this list upon request. The dynamic membership manager is implemented using the RON conduit API; it configures itself with the RON forwarder to receive membership packets and timer callbacks:

```
dynamic_membership::configure(forwarder *f) {
```

Method	**Function** [with optional parameter]
Earliest	first entry [Of the last N]
Latest	most recent entry [Of the last N]
Min	smallest entry [Of the last N]
Max	largest entry [Of the last N]
EWMA	exponential weighted moving average [with parameter p] of the entries.
AVG	unweighted average of the [last N] entries.

Table 4.1: The summarization methods supported by the performance database

```
    f->register_type(RON_MEMBERSHIP, this, NULL);
    f->register_timer(BROADCAST_TIME, this, NULL);
}
```

When the manager receives a timer callback, it sends out an announcement of its known peers. When it receives a `RON_MEMBERSHIP` data packet, it updates its list of known peers based upon the information in the packet.

4.8.6 Performance Database

The performance database exists in two forms. The first is a stand-alone application running on the Berkeley DB3 back-end. It communicates with clients using a simple UDP-based protocol. RON requires reasonably reliable communication with the performance database, but it can handle some packet loss. The second form of the performance database implements an identical interface to the first, but runs instead as a library linked to the RON application. In most scenarios, only a single client used the performance database. The external database process was an impediment to administration.

Probes insert data into the database by calling `pdb_update(class, src, dst, type, value)`. The `src` and `dst` fields contain the IP addresses of the sampled data. To avoid maintaining hard state in the database, the caller specifies a `type` field that tells the database what kinds of values will be inserted.

Clients that want information from the database make a request via `pdb_request(class, src, dst, summarizer, param)`. The `summarizer` and `param` fields allow the client to tell the database how to summarize the data. For instance, a client may specify `SUMMARIZE_AVG` with parameter 5 to request the average of the 5 most recent samples in the database. The database computes the answer, and supplies a callback to the client via `pdb_callback()`. The RON architecture defines metrics for loss, latency, and throughput, and it provides provers to measure them. Applications may define arbitrary metrics and request whatever summarization of the data that they want. The default summarization methods are listed in Table 4.1.

Summarizers take an optional parameter. The AVG method, for instance, accepts the number of samples to use. For example, AVG-4 provides the average of the four most recent samples.

4.9 Summary

This chapter presented the design and implementation of a Resilient Overlay Network. Chapter 5 shows that the described system improves the reliability of Internet packet delivery by detecting and recovering from outages and path failures more quickly than current inter-domain routing protocols. A RON works by deploying in different Internet routing domains nodes that cooperatively route packets for each other. Each RON is an application-layer overlay; nodes in a RON monitor the quality of the underlying Internet paths between themselves and use this information to route packets according to application-specified routing metrics either via a direct Internet path or via other RON nodes.

RON is designed as a flexible set of libraries from which programmers can pick and choose the components they want to customize to implement their overlay networks. To facilitate the use of RON, we designed and implemented a RON IP tunnel that encapsulates selected IP traffic flowing out of a computer and forwards it over the RON network.

We have deployed the RON system described in this chapter in our wide-area Internet testbed, and used it to evaluate RON's ability to overcome failures and performance outages. The next chapter presents a detailed evaluation of RON conducted with several weeks of traces from this deployment.

You never know what you can do till you try.

- William Cobbett

Chapter 5

RON Evaluation

The goal of RON is to overcome path outages and performance failures, without introducing excessive overhead or new failure modes worse than those it corrects. This section presents an evaluation of how well RON meets these goals. Experimental results from the RON testbed show that RONs are beneficial to applications (such as the RON IP tunnel) and that RONs can successfully route around path outages and performance failures.

This evaluation has three primary objectives: First, to study RON's ability to detect outages and recover quickly from them. Second, to investigate performance failures and RON's ability to improve the loss rate, latency, and throughput of badly performing paths. Finally, to investigate two important aspects of RON's routing, showing the effectiveness of its one-intermediate-hop strategy compared to more general alternatives and the stability of RON-generated routes. Table 5.1 summarizes our key findings.

5.1 Method

The results come from experiments with a wide-area RON deployed at several Internet sites. There are $N(N-1)$ different paths between the hosts in an N-site RON deployment. Table 5.2 shows the location of sites for the experiments in this chapter. The results are based on two distinct datasets—RON_1 from March 2001 with $N = 12$ nodes and 132 distinct paths, and RON_2 from May 2001

RON reduced outages in the first dataset by a factor of 5 to 10, and routed around nearly all complete outages and nearly all periods of sustained high loss rates (30% or more). RON avoided 46% of the sustained high loss periods in the second dataset.	§5.2
RON takes 18 seconds, on average, to route around a failure and can do so in the face of a flooding attack.	§5.2
RON successfully routed around bad throughput failures, but did not substantially impact the median throughput, despite improving both loss and latency.	§5.3
In 5% of sample periods, RON reduced the loss probability by 0.05 or more.	§5.3
Single-hop route indirection captured the majority of benefits in our RON deployment.	§5.4

Table 5.1: Major results from measurements of RON.

Name	Description
Aros	ISP in Salt Lake City, UT
CCI	.com in Salt Lake City, UT
Cisco-MA	.com in Waltham, MA
* CMU	Pittsburgh, PA
* Cornell	Ithaca, NY
Lulea	Lulea University, Sweden
MA-Cable	MediaOne Cable in Cambridge, MA
* MIT	Cambridge, MA
CA-T1	.com in Foster City, CA
* NYU	New York, NY
OR-DSL	DSL in Corvallis, OR
* Utah	Salt Lake City, UT
VU-NL	Vrije Univ., Amsterdam, Netherlands
——*Additional hosts used in dataset* RON_2——	
NC-Cable	MediaOne Cable in Durham, NC
PDI	.com in Palo Alto, CA
Mazu	.com in Boston, MA

Table 5.2: The hosts from the RON testbed used in this evaluation. Asterisks indicate U.S. Universities on the Internet2 backbone. The two European universities, `Lulea` and `VU-NL` are classified as non-Internet2.

with $N = 16$ nodes and 240 distinct paths. In RON_1, `traceroute` data shows that 36 different AS's were traversed, with at least 74 distinct inter-AS links; for RON_2, 50 AS's and 118 inter-AS links were traversed. The $O(N^2)$ scaling of path diversity suggests that even small numbers of confederating hosts expose a large number of Internet paths [115, 116].

We do not claim that our experiments and results are typical or that they represent anything other than our deployment, but they do suggest the kinds of gains one might realize with RON in practice. The results presented in this chapter are consistent, however, with those found in subsequent studies. Zhang *et al.* studied traffic from 9,232 ASs that entered 145 different PlanetLab nodes, finding that RON-style one-hop indirection could avoid 43% of outages [179]. Gummadi *et al.* found that one-hop routing could avoid 54% failures to random Internet destinations [64].

Several of the testbed host sites are Internet2-connected educational institutions, and we found that this high-speed experimental network presents opportunities for path improvement not available in the public Internet. To demonstrate that RON's policy routing module works, and to make our measurements closer to what Internet hosts in general would observe, all the measurements reported herein were taken with a policy that prohibited sending traffic to or from commercial sites over the Internet2. Therefore, for instance, a packet could travel from Utah to Cornell to NYU, but not from Aros to Utah to NYU. This policy is consistent with the Internet2 Acceptable Use Policy (AUP) that precludes commercial traffic. As a result, in our measurements, *all path improvements involving a commercial site came only from commercial links, never from Internet2 links.*

The raw measurement data consists of probe packets, throughput samples, and `traceroute` results. To probe, each RON node independently repeated the following steps:

1. Pick a random node, j;

2. Pick a probe-type from one of $\{direct, latency, loss\}$ using round-robin selection. Send a probe to node j;
3. Delay for a random interval.

This method characterizes all $N(N-1)$ paths. RON_1, contains 10.9 million packet departure and arrival times, collected over 64 contiguous hours between 21 Mar 2001 and 23 Mar 2001. These times produced 2.6 million individual RTT, one-way loss, and jitter data samples, for which we calculated time-averaged samples averaged over a 30-minute duration.[1] We also took 8,855 throughput samples from 1 MByte bulk transfers, recording the time at which each power-of-two's worth of data was sent, and the duration of the transfer. RON_2 contains 85 contiguous hours from 7 May 2001 and 11 May 2001, consisting of 13.8 million arrival and departure times.

Instead of recording probe successes and failures on-line, we post-processed all samples in a database, correcting for clock differences, computer crashes, and other errors. This post-analysis made it possible to retroactively extract one-way loss rates without using a complicated probing protocol. The measurement data is available at `http://nms.lcs.mit.edu/ron/`.

5.2 Overcoming Path Outages

Precisely measuring a path outage is more difficult than one might think. One possibility is to define an outage based on the length of time when no packets get through the path. As a yardstick, we could use ranges of time in which TCP implementations will time out and shut down a connection; these vary from about 120 seconds (*e.g.*, some Solaris versions) to 511 seconds (*e.g.*, most BSDs). This gives us a metric for when batch TCP applications will fail; from our own experience, we believe that most users have a far lower threshold for declaring their Internet connectivity "dead."

The problem with this small time-scale definition is that robustly measuring short outages and differentiating between a very high packet loss rate and true disconnections is hard. However, since distributed applications suffer in either case, it is operationally useful to define a path outage as the length of time over which the packet loss-rate is larger than some threshold:

$$outage(\tau, p) = \begin{cases} \text{true} & \text{if packet loss over the period } \tau > p \\ \text{false} & otherwise \end{cases}$$

For values of τ on the order of several minutes, measured values of p near or greater than 30% degrades TCP performance by orders of magnitude, by forcing TCP into frequent timeout-based retransmissions [109].

How often is $outage(\tau, p) =$ true, and how often is RON able to route around these situations? Figure 5-1 shows a scatterplot of the improvement in loss-rate, averaged over $\tau = 1800$ seconds achieved by RON. To identify outages consider the 32 points above the $p = 0.3$ line parallel to the horizontal axis, which signifies a condition bad enough to kill most applications. There are no points to the right of the $p = 0.3$ line parallel to the vertical axis. (The scatterplot conceals most of the data in the lower-left corner; see also the CDF in Figure 5-3.)

The number of times $outage(\tau, p)$ was true for $\tau = 1800$s is shown in Table 5.3. These are statistics obtained over the 64 and 85 hours of probes (13,650 and 34,000 half hour samples of loss rate

[1] Some hosts came and went during the study, so the number of samples in the averages is not always as large as one might expect if all hosts were continually present. More details are available in the documentation accompanying the data on the RON Web site, at `http://nms.lcs.mit.edu/ron/`

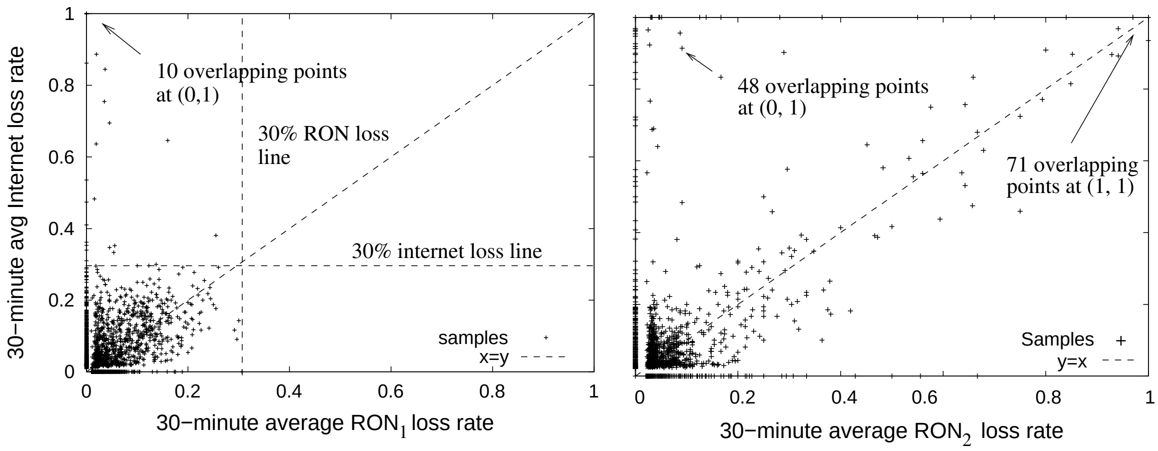


Figure 5-1: Packet loss rate averaged over 30-minute intervals for direct Internet paths *vs.* RON paths for the RON_1 and RON_2 datasets. The RON_1, plot includes 32 points above the $p = 0.3$ horizontal line, and 20 points above $p = 0.5$, including overlapping points; RON's loss optimizing router avoided all of these direct Internet failures, never experiencing 30-minute loss rates over 30%. The RON_2 dataset includes more monitoring noise, because several computers crashed and were not automatically removed for an hour, and also includes several periods where RON was not able to correct an outage.

averages) in RON_1 and RON_2, respectively. We count a "RON win" if the time-averaged loss rate on the Internet was $\geq p\%$ and the loss rate with RON was $< p\%$; "No Change" and "RON Loss" are analogously defined. There were 10 complete communication outages was 10 in this dataset, which means that there were 10 instances when the sampled paths had a 100% loss rate. TCP would perceive significant problems at loss rates $\geq 30\%$, and there were several occurrences of these in the data. The numbers in this table are not the number of link or routing failures observed in the Internet across the sampled paths; the number of such failures could have been lower (*e.g.*, multiple 30-minute averaged samples with a 100% loss rate may have been the result of the same problem) or higher (*e.g.*, a time-averaged loss rate of 50% could have resulted from multiple link outages of a few minutes each).

We emphasize that Internet2 paths were never used to improve a non-Internet2 connection's performance. In fact, the vast majority of the problems corrected by RON involved only commercial Internet connections, as is shown in brackets in table 5.3 by the number of outages when *all* Internet2 paths are removed from consideration.

These results show that RON offers substantial improvements during a large fraction of outages, but may pick sub-optimal paths at lower outage rates when p is between 0 and 20%. However, in the RON_1 dataset, RON's outage detection and re-routing machinery was in fact able to successfully route around *all* the outage situations! This finding is especially revealing because it suggests that none of the outages RON encountered occurred on "edge" links connecting the site to the Internet, but occurred where path diversity allowed RON to provide connectivity even when Internet routing could not. In RON_2, about 46% of the serious outage situations were overcome; our analysis showed that the outages that were not overcome were all cases where *no* other host in our RON had connectivity to the unreachable host.

One way to view these results is to observe that in RON_1 there are a total of 13,650/2 = 6,825 "path hours" represented. There were five "path-hours" of complete outage (100% loss rate); RON routed

	RON_1			RON_2		
Loss Rate	RON Win	No Change	RON Loss	RON Win	No Change	RON Loss
10%	526 [517]	58 [51]	47 [45]	557	165	113
20%	142 [140]	4 [3]	15 [15]	168	112	33
30%	32 [32]	0	0	131	184	18
40%	23 [23]	0	0	110	75	7
50%	20 [20]	0	0	106	69	7
60%	19 [19]	0	0	100	62	5
70%	15 [15]	0	0	93	57	1
80%	14 [14]	0	0	87	54	0
90%	12 [12]	0	0	85	48	2
100%	10 [10]	0	0	67	45	1

Table 5.3: Outage data. A "RON win" at p% means that the loss rate of the direct Internet path was $\geq p$% and the RON loss rate was $< p$%. Numbers in brackets show the contribution to the total outage number after eliminating all the (typically more reliable) Internet2 paths, which better reflects the public Internet.

around all these. Similarly, RON_2 represents 17,000 path-hours with 56 path-hours of complete outage. RON was able to route around 33.5 path-hours of outage. In the remaining outages, no other paths through the RON reached a partitioned site.

RON also encountered numerous outages of shorter duration (typically three or more minutes, consistent with BGP's detection and recovery time scales), and RON was able to mask them more quickly. As one example of outage recovery, see Figure 5-2, which shows a 10-minute interval when no packets made it between `Cisco-MA` and most of the Internet (the few "+" marks on the horizontal axis of the lower figure between the dashed vertical lines show periods of total failures). In fact, among the RON sites, the only site to which `Cisco-MA` had any connectivity at this time was `MIT`. RON was able to find this path and successfully route packets between `Cisco-MA` and the commercial RON sites[2].

5.2.1 Overhead and Outage Detection Time

The implementation of the RON IP tunnel adds about 220 microseconds of latency to packet delivery, and increases the memory bandwidth required to forward data. Figure 5-4 shows the ping latency between two machines on a 100 Mbits/s Ethernet. This overhead is primarily due to the use of divert sockets to obtain data. We do not envision our prototype being used on high-speed links, although it is capable of forwarding at up to 90 Mbits/s using a 733 Mhz Celeron-based machine (Table 5.4).

The frequency of routing and probing updates leads to a trade-off between network overhead and responsiveness to a path failure. Using the protocol shown in Figure 4-13 RON probes every other node every `PROBE_INTERVAL` seconds, plus a random jitter of up to $\frac{1}{3}$`PROBE_INTERVAL` extra seconds. Thus, the average time between two probes is $\frac{7}{6}$`PROBE_INTERVAL` seconds. If a probe is not returned after `PROBE_TIMEOUT` seconds, RON considers it lost. A RON node sends a routing update to every other RON node every `ROUTING_INTERVAL` seconds on average. The RON implementation uses the same values specified in Chapter 4:

[2]The RON Internet2 policy forbade permitting MIT to carry traffic from a commercial node.

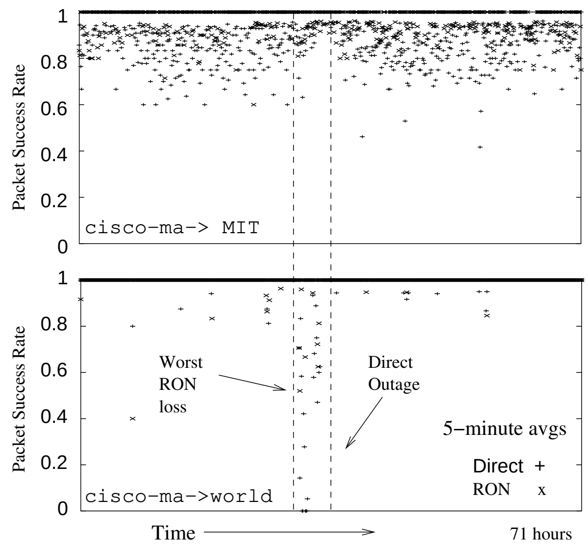


Figure 5-2: The lower figure shows a 10-minute outage from `Cisco-MA` to most places on the Internet (the few notches on the horizontal axis at the bottom). RON was able to route around the outage, because `Cisco-MA`'s packet loss rate to `MIT` was relatively unaffected during this outage.

Test	Direct	Opt RON	RON
100 Mbits/s	11563 KB/s	11562 KB/s	11170 KB/s
1 Gbits/s	37945 KB/s	11422.19 KB/s	5454.41 KB/s

Table 5.4: TCP throughput with and without RON on 100 Mbits/s and 1000 Mbits/s switched networks. The 100 Mbits/s test used a 600Mhz PIII-based computer with a 100Mhz bus. Its throughput reduction of 3.6% is almost exactly the amount of extra space added by the RON encapsulation (4%). A test using a slower Celeron/733-based computer with a 66Mhz bus and Gigabit Ethernet shows that the cost of memory copies to access the divert sockets dominated the overhead.

`PROBE_INTERVAL`	12 seconds
`PROBE_TIMEOUT`	3 seconds
`ROUTING_INTERVAL`	14 seconds

When a packet loss occurs, the next probe packet is sent immediately, up to a maximum of four "quick" probes. After four consecutive losses, RON considers the path down. As noted in Chapter 4, the RON implementation detects failures in 4–20 seconds, with an average of 11 seconds.[3] Announcing this failure via RON's routing takes an additional seven seconds on average.

Sending packets through another node introduces a potential new failure coupling between the communicating nodes and the indirect node that is not present in ordinary Internet communications. To avoid inducing outages through nodes crashing or going offline, RON must be responsive in detecting a failed peer. Recovering from a *remote* virtual link failure between the indirect host and the destination requires that the indirect host detect the virtual link failure and send a routing update. Assuming small transmission delays, a remote virtual link failure adds between 0 and

[3] Assuming the RTT between the hosts is under 1 second.

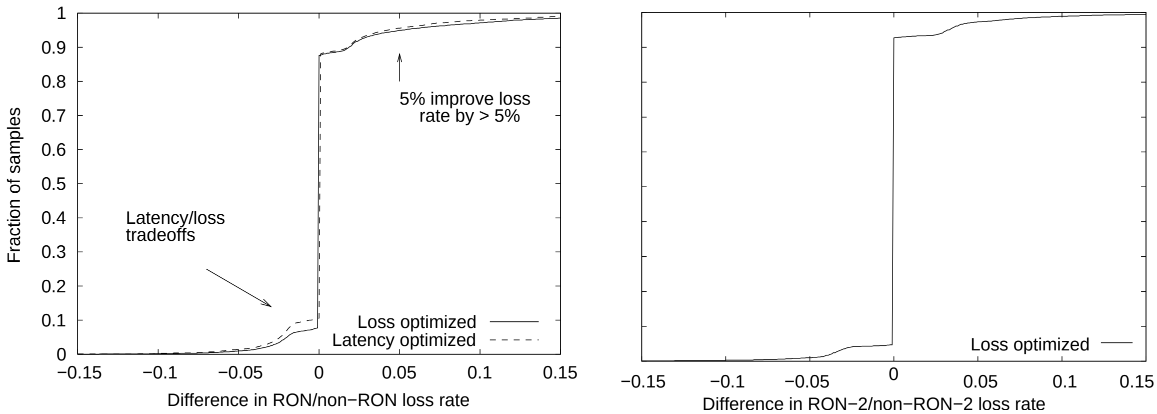


Figure 5-3: The cumulative distribution function (CDF) of the improvement in loss rate achieved by RON. The samples detect unidirectional loss, and are averaged over $\tau = 1800$s intervals.

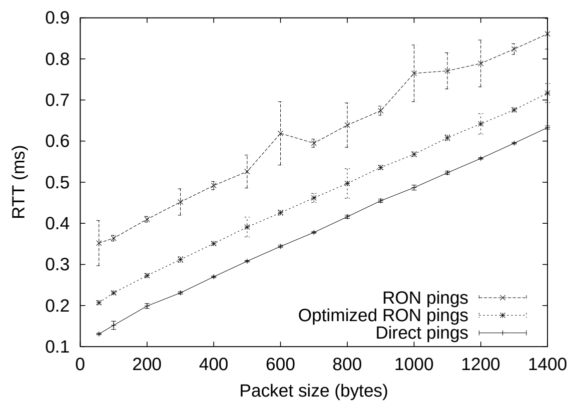


Figure 5-4: Latency increases from RON encapsulation on a 100 Mbits/s LAN. On average, RON adds about 220 microseconds from packet size increases and additional host processing. With optimized direct sending, RON adds only 75 microseconds.

ROUTING_INTERVAL seconds to the recovery times. RON detects a remote virtual link failure in about 18 seconds, during which time the intermediate node may or may not be able to re-route the packet. High packet loss can increase this duration by requiring extra transmissions of routing packets; in general, this takes notification time to $\frac{\texttt{ROUTING\_INTERVAL}}{2(1-(\texttt{loss rate})}$. On networks that experience frequent short failures, the underlying network may recover more rapidly than does RON. If, however, some of the paths through this network do not experience such failures, RON's loss adaptation mechanisms will have already routed the RON's traffic to those good paths.

The time required to detect a failed path suggests that passive monitoring of in-use links will improve the single-virtual-link failure recovery case considerably, since the traffic flowing down the virtual link can be treated as "probes." Although recovery from multiple virtual link failures will still take on the order of twenty seconds, it is still significantly faster than recovery with wide-area inter-domain Internet routing, which often requires several minutes (Chapters 1 and 2).

RON probe packets are $S = 69$ bytes long. The probe traffic (see Figure 4-13) received at each

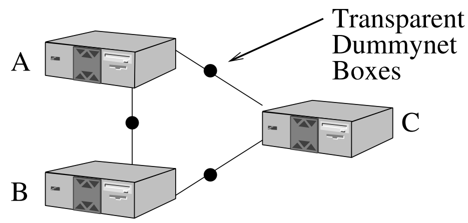


Figure 5-5: Testbed configuration.

node in an N-node RON is 2 probes ($2S$ bytes) from each of the $(N-1)$ remote hosts. Probes are sent every $\texttt{PROBE\_INTERVAL} + \texttt{rand}(\frac{1}{3}\texttt{PROBE\_INTERVAL})$ seconds, for total probe traffic of $\frac{2S\times(N-1)}{\frac{7}{6}\texttt{PROBE\_INTERVAL}}$ bytes/sec. RON routing traffic has $H = 60$ bytes of header information, plus $P = 20$ bytes of information to describe the path to each peer. Thus, each node sees $\frac{(N-1)\times(H+P(N-1))}{\texttt{ROUTING\_INTERVAL}}$ bytes/sec of routing traffic. The traffic for several sizes of RONs, with the default intervals, is shown below in kilobits per second. The figures represent the worst case in which RON must transmit routing updates each period because all link properties have changed significantly.

10 nodes	20 nodes	30 nodes	40 nodes	50 nodes
2.2 Kbits/s	6.6 Kbits/s	13.32 Kbits/s	22.25 Kbits/s	33 Kbits/s

For a RON of $N = 50$ nodes, about 30 Kbits/s of active probing overhead allows recovery between 12 and 25 seconds. Combining probing and routing traffic, delaying updates to consolidate routing announcements, and sending updates only when virtual link properties change past some threshhold could all be used to reduce the amount of overhead traffic. The total traffic growth, however, remains $O(N^2)$, and each node must still send $O(N)$ probing traffic to monitor its virtual links.

We believe that this overhead is reasonable for several classes of applications that require recovery from failures within several seconds. 30 Kbits/s is typically less than 10% of the bandwidth of today's "broadband" Internet links, and it is the cost of achieving the benefits of fault recovery using RON. This overhead is not excessive when compared to other sources of packets on the Internet. For example, many of the packets on today's Internet are TCP acknowledgments, typically sent for every other TCP data segment. These "overhead" packets are necessary for reliability and congestion control; similarly, RON's active probes may be viewed as "overhead" that help achieve rapid recovery from failures.

Several of the RON messages could be bundled together or sent opportunistically with user traffic. While such bundling could reduce the amount of traffic generated by RON's measurements and routing, it would not eliminate the need for $O(N^2)$ traffic. A promising alternative is to use measurements of ongoing traffic in place of the probing traffic to passively and rapidly detect failures. In addition to being an important aspect of future work, a variation of this technique has been useful in the MONET system, which selectively sends duplicate copies of packets to act both as probes and as guards against link failures (Chapter 6).

5.2.2 Avoiding Packet Flooding Attacks

To evaluate the effectiveness of RON in routing traffic around a link saturated by attack traffic, we conducted tests on the Netbed Testbed [174], using Intel PIII/600MHz machines on a quiescent

100 Mbits/s switched Ethernet with Intel Etherexpress Pro/100 interfaces. The network topology emulated three hosts connected in a triangle, with 256 Kbits/s, 30 ms latency links between each pair (Figure 5-5). Indirect routing was possible through the third node, but the latencies made it less preferable.

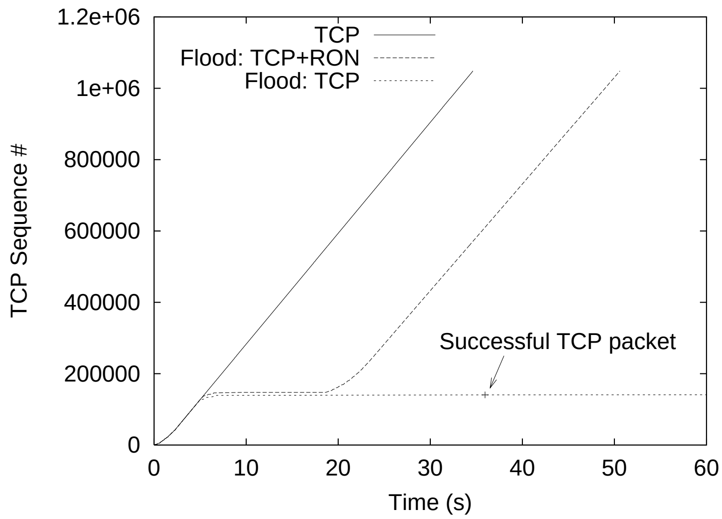


Figure 5-6: A TCP sequence trace at the receiver of a connection re-routed by RON during a packet flooding attack on its primary link. The 18-second recovery time is a combination of RON's link detection time and TCP's retransmission timers. Without RON, the connection is unable to get packets through at more than a crawl, though the link is still technically "working."

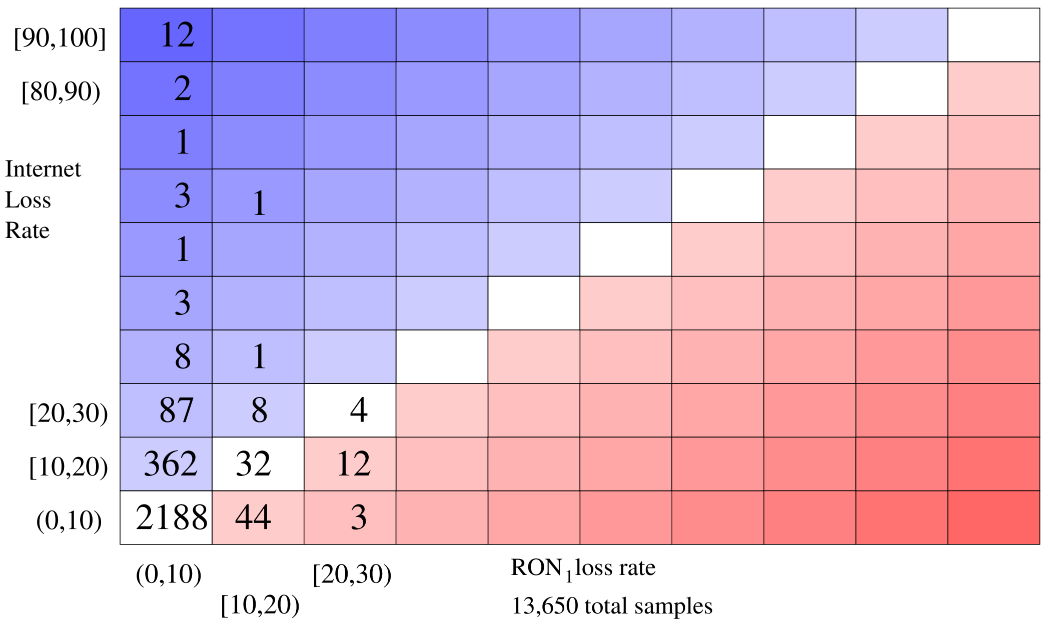
Figure 5-6 shows the TCP sequence traces of three bulk transfers taken at the receiver. The left-most trace is an uninterrupted TCP transfer, which finishes in about 35 seconds. The middle trace shows the transfer running over RON, with a packet flooding attack beginning at five seconds. RON recovers relatively quickly, taking about 20 seconds to re-route the connection through the third node after which the connection proceeds normally. The right-most trace (the horizontal dots) shows the non-RON TCP connection in the presence of the attack. Because TCP traffic was still getting through at a very slow rate, BGP would not have marked this link as down. Even had it done so, a route change would have simply carried the malicious traffic along the newly selected links, and the route change itself could have caused several minutes of impaired availability [143].

5.3 Overcoming Performance Failures

This section examines the improvements in loss-rate, latency, and throughput provided by RON. While the major goal of RON is to avoid outages, the system also helps avoid periods of extremely degraded performance, and can improveme the overall loss and latency on some paths.

5.3.1 Loss Rate and Latency

Figure 5-3 summarizes the observed loss rate results as a CDF: in RON_1, RON improved the loss probability by more than 0.05 a little more than 5% of the time. A 5% improvement in loss probability is substantial for applications that use TCP. Upon closer analysis, we found that the *outage detection* component of RON routing was instrumental in detecting bad situations promptly



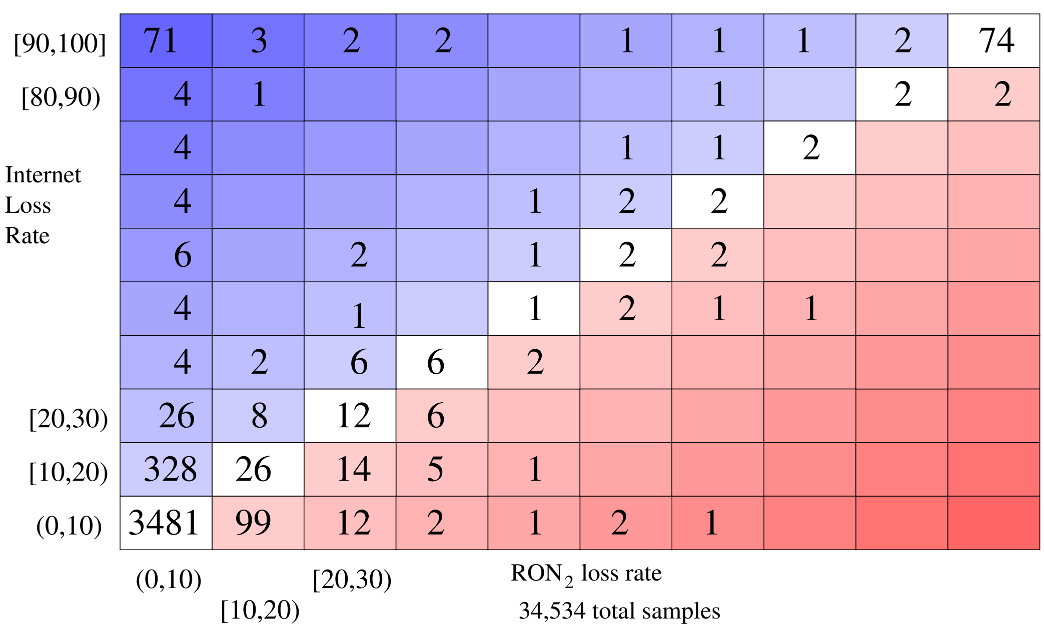


Figure 5-7: Binned scatterplot of loss rate changes under RON_1 and RON_2. Each count represents a 30 minute period on some path. For example, the upper left corner of the RON_1 plot shows that there were 12 30-minute periods in which RON reduced the loss rate from nearly 100% to near 0%. The RON_2 dataset experienced more times during which RON was unable to correct outages. The majority of points in both plots are at the (0,0) point and are not shown.

and in triggering a new path. RON's loss rate reductions could come either from making either insignificant changes (*e.g.*, 85% loss reduced to 80% loss) or more significant small changes (*e.g.*, 5% to 0%). Figure 5-7 presents a binned scatterplot of the loss rate changes with RON_1. From this figure, it appears that RON is often able to improve slightly impaired paths (0-20% loss), and make large improvements in the more rare cases when the paths have failed.

This figure also shows that RON routing does not always improve performance. There is a tiny, but noticeable, portion of the CDF to the left of the -0.05 region, showing that RON can make loss rates worse. There are two reasons for this: First, RON uses a longer-term average of packet loss rate to determine its low-loss routes, and it may mispredict for a period after link loss rates change. Second, and more importantly, the RON router uses *bi-directional* information to optimize uni-directional loss rates.

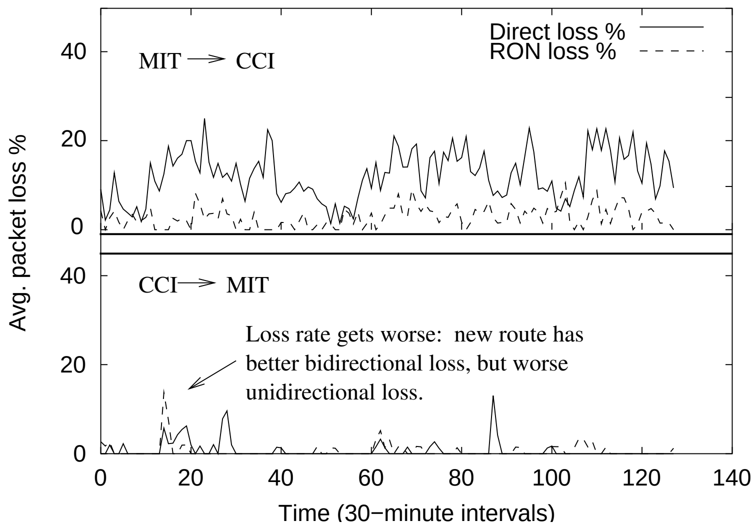


Figure 5-8: The benefits and limitations of bidirectional loss probes. The `MIT-CCI` link is heavily congested and plagued by outages. RON finds a path with much better *bi-directional* loss characteristics—but the direct path from `CCI` to `MIT` was sometimes better *in one direction* than the new RON path, even though the new RON path was superior overall. Note that the Y-axis goes only to about the 40% mark, not 100%.

This combination of substantial successes but observable failures of RON's loss optimizing router is shown in detail for one path in Figure 5-8. The figure examines three days of loss behavior between `MIT` and `CCI`, averaged over 30-minute intervals. The top curve shows the packet loss rate across the Internet and RON for the `MIT` sender, while the bottom curve shows the same for the `CCI` sender. RON is a tremendous improvement in the former case, but not in the latter. In fact, in the latter case, there are infrequent occurrences where RON makes the unidirectional loss rate substantially worse. This problem arises because the RON prober returns bi-directional loss estimates, which the optimizer uses to infer a uni-directional loss rate by assuming symmetric loss rates. The loss rate between `MIT` and `CCI` was highly asymmetric.

RON reduces communication latency in many cases, despite the additional hop and user-level encapsulation. Figure 5-9 shows the CDF of latency with and without RON.[4] It is difficult to determine

[4] During outages, RON may find paths that we cannot compare to the direct path; we eliminated 113 out of 39683 samples due to long outages.

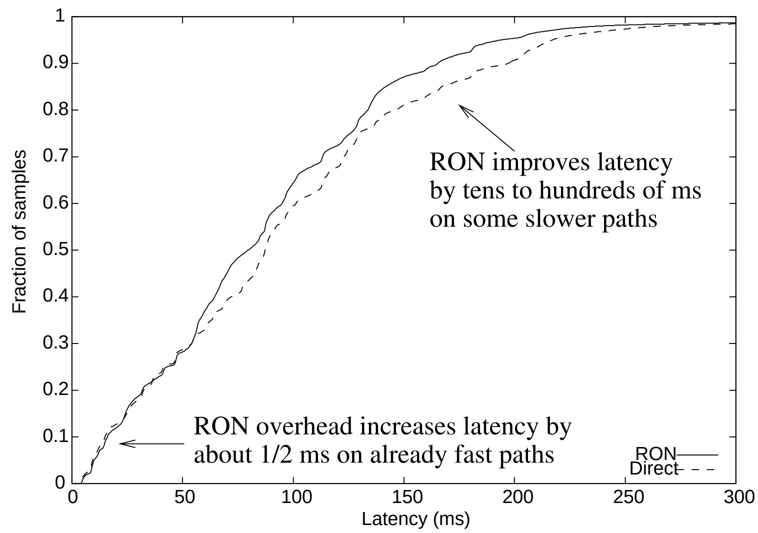


Figure 5-9: Five-minute average latencies over the direct Internet path and over RON, shown as a CDF.

from the CDF *where* the latency improvements come from. Figure 5-10 shows the same data as a scatterplot of the round-trip latencies achieved via RON against those found directly on the Internet, averaged in five-minute blocks. The points in the scatterplot appear in clustered bands, showing the improvements achieved on different node pairs at different times.

5.3.2 TCP Throughput

In many cases, RON improves the TCP throughput between communicating nodes. RON's throughput-optimizing router does not attempt to detect or change routes to obtain small changes in throughput, since the underlying Internet throughput is not particularly stable on most paths; rather, it re-routes traffic only when it is likely that the new path offers a 50% or greater improvement in throughput.

To compare the RON path used by throughput-optimizing RON routers to the direct Internet path, we took four sequential throughput samples—two with RON and two without—and compared the ratio of the average throughput achieved by RON to the average throughput achieved directly over the Internet. Throughput samples exhibit high variance over time. Because the samples on the different paths (RON and the direct Internet path) are taken at different times, the interleaved samples provide a way to estimate how much of the difference was due to variance on each path.

Figure 5-11 shows the distribution of the throughput ratio with and without RON. Out of 2,035 paired quartets of throughput samples in RON_1, only 1% received less than 50% of the throughput with RON, while 5% of the samples doubled their throughput. In fact, 2% of the samples increased their throughput by more than a factor of five, and nine samples improved by a factor of 10 during periods of intermittent Internet connectivity failures. The situation was slightly improved in the RON_2 dataset, because we corrected an error where RON mistakenly set the path's Maximum Transmission Unit (MTU) too small when sending packets directly over the Internet—an event most likely to occur between sites with fast default paths. In RON_2, the *median* transfer throughput was virtually unchanged, at just over 99% of its non-RON value. The *mean* throughput, however, was 123% of the non-RON throughput, showing that RON was able to make a significant contribution to some of the flows, some of the time. These numbers include times when RON was able to find

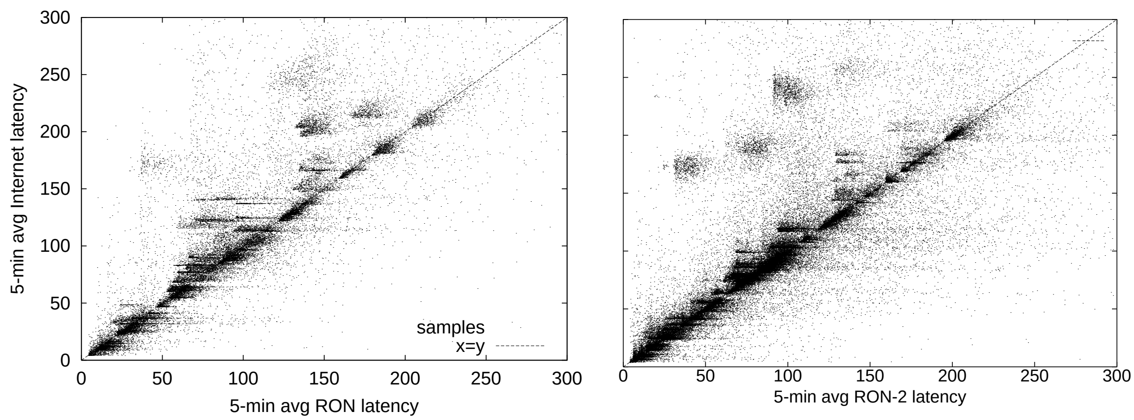


Figure 5-10: The same data as figure 5-9. Dots above the x=y line signify cases where the RON latency was lower than the direct Internet latency. The clustering and banding of the samples shows that the latency improvements shown in Figure 5-9 come from specific host-host pairs at different times.

a path via an indirect node, but no connection could be created via the Internet. These alternate hop connections may often be disadvantageous, and count against RON in terms of throughput. We have not filtered out the "Internet Outage" situations from the throughput analysis. Similarly, the deck is somewhat stacked against RON because of the no-Internet2 policy, which prevents RON from taking advantage of the fast inter-university links while forcing it to compete with them for bandwidth improvement. Given these constraints, we believe the 123% figure for RON_2 is very encouraging.

Not surprisingly, the distribution of the improvement in throughput was heavily biased towards slower sites. RON was never able to improve the connectivity between two Internet2-connected sites, but sites with network connections of a few Megabits/s or under experienced occasional improvements. Many instances in which RON reduced throughput occurred because of an Internet2 path with near-zero loss but unusually high latency because of a routing misconfiguration. RON was able to find an alternate, lower-latency path, but this indirect routing significantly reduced throughput.

5.4 RON Routing Behavior

We instrumented a RON node to output its link-state routing table roughly every 14 seconds, with a randomized jitter to avoid periodic effects. We analyzed a 16-hour log of RON's routing table updates. The log contained 5,616 individual table snapshots. The trace showed the routing information used to determine 876,096 different point-to-point routes over the period of the capture.

5.4.1 RON Path Lengths

The outage results show that RON's single-hop indirection worked well for avoiding problematic paths. If there is a problem with the direct Internet path between two nodes s and t, it is often because some link $l(s, t)$ between them is highly congested or is not working. As long as there is even one RON node, u, such that the Internet path between s and u, and the Internet path between u and t, do not go through $l(s, t)$, then RON's single-hop indirection will suffice. Of course, if

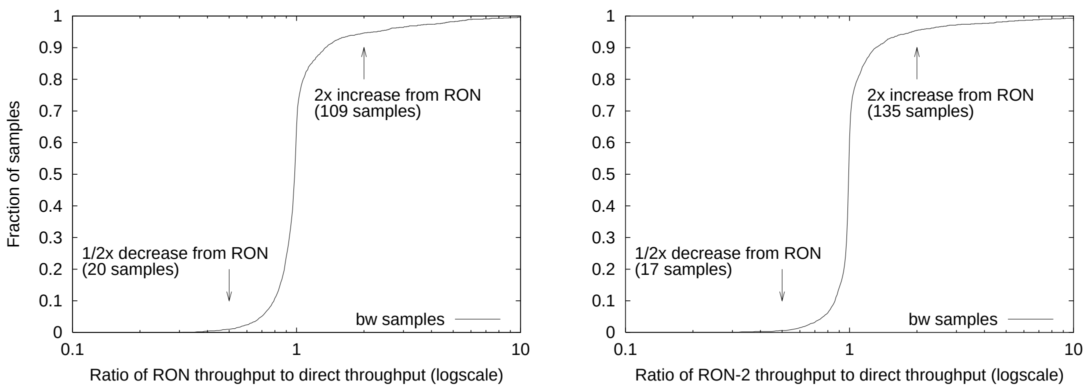


Figure 5-11: CDF of the ratio of throughput achieved via RON to that achieved directly via the Internet. RON_1 includes 2,035 ratios; RON_2 includes 3,029. RON markedly improved throughput in near-outage conditions, but did not change the median throughput.

all paths from s to the other RON nodes traverse $l(s,t)$, or if the paths from every RON node to t traverse $l(s,t)$, then the RON cannot overcome the outage of $l(s,t)$.

Single-hop indirection also suffices for a latency-optimizing RON. We found this by comparing single-hop indirection with a general shortest-paths algorithm on the link-state trace.[5] The direct Internet path provided the best average latency about 48.8% of the time. In the remaining 51.2% of the time when indirect routing reduced the latency, the shortest path involved only one intermediate node about 98% of the time.

The following simple (and idealized) model provides an explanation. Consider a source node s and a destination node t in a RON with R other nodes. Denote by p_s the probability that the lowest-latency path between node s and another node in the RON is the direct Internet path between them. Similarly, denote by p_i the probability that the lowest-latency path between node $i(\neq s)$ in the RON and t is the direct link connecting them. We show that even small values of these probabilities, under independence assumptions (which are often justifiable if RON nodes are in different AS's), lead to at most one intermediate hop providing lowest-latency paths most of the time.

The probability that a single-intermediate RON path is optimal (for latency) given that the direct path is not optimal is given by $1 - \Pi_{i=1}^{R}(1 - p_s p_i)$, since this can only happen if *none* of the other R nodes has a direct-link shortest path from s and to t. This implies that the probability that either the direct path, or a single-hop intermediate path is the optimal path between s and t is given by $1-(1-p_s)\Pi_{i=1}^{R}(1-p_s p_i)$. In our case, p_s and p_i are both around 0.5, and R is 10 or more, making this probability close to 1. In fact, even relatively small values of p_s and p_i cause this probability to approach 1; if $p_s \approx p_i = p \approx c/\sqrt{R}$, then the optimal path has either the direct path or has one intermediate RON hop with probability at least $1-(1-p^2)^{R+1} = 1-(1-c/R)^{R+1} \approx 1-e^{-c}$, which can be made arbitrarily close to 1 for suitable c.

5.4.2 RON Route Stability

As a dynamic, measurement-based routing system, RON creates the potential for instability or route flapping. We simulated RON's path selection algorithms on the link-state trace to investigate route

[5] This analysis does not consider the effects of the no-Internet2 policy, and only considered latency optimization *without* outage avoidance.

stability. The "Changes" column of Table 5.5 shows the number of path changes that occurred as a function of the hysteresis before triggering a path change. The other columns show the *persistence* of each RON route, obtained by calculating the number of consecutive samples over which the route remained unchanged. The average time between samples was 14 seconds.

Method	# Changes	Avg	Med	Max
No Hyst	26,205	19.3	3	4,607
Hyst-5%	21,253	24	5	3,438
Hyst-10%	9,436	49	10	4,607
Hyst-25%	4,557	94	17	5,136
Hyst-50%	2,446	138	25	4,703
Random process	260,000	2	1	<16

Table 5.5: Number of path changes and run-lengths of routing persistence for different hysteresis values.

The median run-length with the 5% hysteresis we implemented was five, about 70 seconds. The frequency of our table traces does not preclude undetected interim changes, especially with shorter run-lengths, but the longer run-lengths indicate reasonable stability with good probability. 5% and 10% hysteresis values appear to provide a good trade-off between stability and responsiveness. The "random process" row shows the expected number of route changes if the routes were flapping at random.

5.4.3 Application-specific Path Selection

There are situations where RON's latency-, loss-, and throughput-optimizing routers pick different paths. As an example, RON was able to find a lower-latency path between `CCI` and `MA-Cable` that had nearly three times the loss rate:

	`CCI` → `MA-Cable`	`MIT` → `Cisco-MA`
Direct	112 ms, 0.77% loss	38 ms, 12.1% loss
Lat-Opt	100 ms, 2.9% loss	43 ms, 10.5% loss
Loss-Opt	114 ms, 0.6% loss	189 ms, 3.62 loss%

In contrast, between `MIT` and `Cisco-MA`, RON's latency optimizer made the latency *worse* because the outage detector was triggered frequently. The loss-optimizing router reduced the loss rate significantly, but at the cost of a five-fold increase in latency. The existence of these trade-offs (although we do not know how frequently they occur in the global Internet), and the lack of a single, obvious, optimal path reinforces our belief that a flexible, application-informed routing system can benefit applications.

5.5 Security Discussion

RON succeeds in its goal of masking many Internet failures; does it also create new vulnerabilities? Because it uses a set of peer nodes to relay traffic, RON (and the MONET system described in the following two chapters) both extend trust to their peers, inviting the possibility that the peers could be compromised. There are several open security issues with these systems:

Lack of encryption. The RON implementation does not provide strong authentication guarantees for its traffic; it instead relies primarily upon IP-based security. Modifying RON's communication to use strong cryptography is a relatively straightforward design change that is similar to the functions performed by commodity virtual proviate network software. While such a change does involve careful implementation, adding strong authentication and encryption to RON should be considered mandatory were it to be more widely deployed.

A wider trust boundary. Compromising a highly specialized Internet router is more difficult than compromising a host that provides a larger set of services. By using overlay networks, RON extends trust to hosts outside the communicating pair, which increases the vulnerability of the communication to compromise. If overlay routing becomes important enough, the nodes providing such services will also become as specialized and hardened to compromise as are Internet routers. Until that time, special care must be taken to protect the nodes that provide overlay service. Augmenting RON with passive detection of the end-to-end success or failure of its routing choices (*e.g.*, by examining whether the TCP sequence numbers are increasing) could help RON withstand the compromise of a peer node that *only* performed forwarding, but the general issue of managing trust in overlay networks remains an open problem.

Targeted attacks. There are certain attacks that RON cannot yet withstand. For example, if an attacker can distinguish between RON's control messages and those containing data, the attacker could drop data packets without interrupting routing. While encryption would mask a direct reading of the packet types, it is still possible to infer the packet types by examining the packet size. Modifying RON to piggy-back routing and control information on top of data packets would increase RON's resilience to this attack, as would passively extracting more performance information from the flow of data traffic.

5.6 Summary

The combination of RON's outage detection and measurement-based routing can provide significant gains in practice. This chapter examined two datasets collected on the RON testbed. An analysis of this data showed that RON can reduce outages by 50-90%. The performance of many links can be improved using RON's latency, loss, and throughput optimizing routers. Finally, we have shown that RON adds a tolerable amount of overhead, and that a single RON deployed in the real Internet has reasonable routing stability.

We will either find a way, or make one.

- Hannibal

Chapter 6

Multi-homed Overlay Networks

The previous chapters examined several ways that overlay networks could improve communication between a small group of end-hosts. This chapter describes the design and implementation of *MONET*, a Multi-homed Overlay Network that improves a client's ability to reach Web sites anywhere on the Internet.

6.1 Introduction

The previous chapters of this dissertation presented overlay-network based fault masking systems that can circumvent failures that occur in the middle of the network. The remaining failures, however, often occur at the access link of one of the communicating nodes, where a single failure can disconnect a host from the entire network. MONET combines overlay failure masking with explicitly engineered access link redundancy to ensure that clients are not easily disconnected from the network when attempting to access Web sites. MONET also seeks out multiple servers when they are available in order to reduce the impact of server access link failures.

The goal of MONET is to reduce periods of downtime and exceptional delays that lead to a poor user experience—to save seconds, not milliseconds. MONET takes advantage of multiple redundant paths, whose failure modes are expected to be mostly independent, between a client and the Web site it seeks, and determines at any given time which of the many possible paths to use. Specifically, MONET addresses two questions:

1. How to obtain multiple paths from a client to a site?
2. Which path(s) to use, and at what times?

The rest of this chapter answers these questions by discussing the design of MONET. Section 6.2 begins with an overview of the MONET architecture. Section 6.3 describes the different ways in which MONET obtains multiple paths between clients and Internet services. These paths involve both distinct physical interconnections and an overlay network of nodes providing cooperative data transit. Section 6.4 describes the MONET protocol and algorithm for picking which paths to use at any given time for a client-server connection. Finally, Section 6.5 discusses optimizations to reduce the overhead on Internet servers introduced by the proposed techniques.

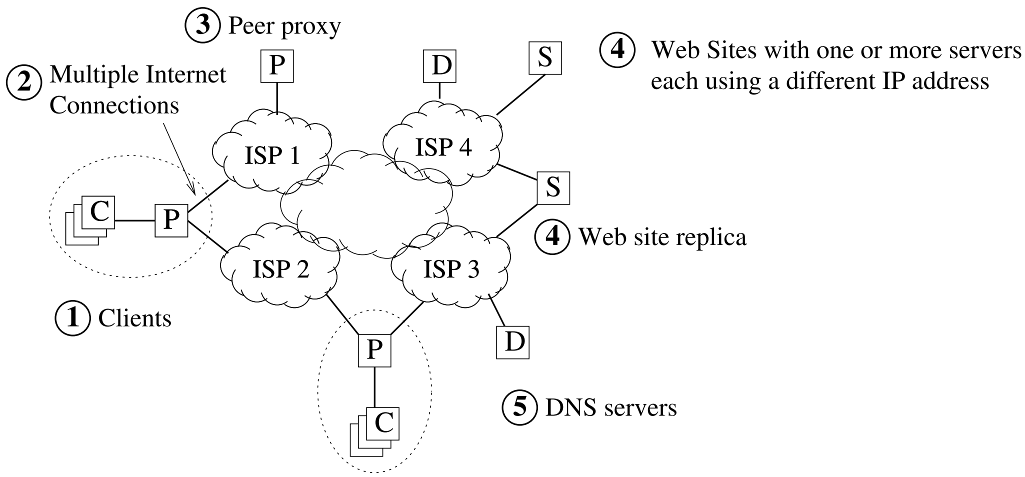


Figure 6-1: The MONET environment. Clients (1) contact Web sites via a local MONET proxy. That local proxy may be multi-homed with multiple local interfaces (2), and may also route requests through remote peer proxies (3). Clients want to communicate with Web sites (4), which may be themselves multi-homed or spread over multiple machines. Web sites must be located using DNS (5); DNS servers are typically replicated over multiple machines.

6.2 System Architecture

MONET consists of a set of Web proxies deployed across the Internet, which serve as conduits for client connections to Web sites. One site might have one or a small number of proxies, and the entire system may have from a handful to a few hundred proxies.

Each proxy has some of the following paths to a Web site at its disposal, as shown in Figure 6-1.

1. **Multi-homed local links**: A MONET proxy has a small number of links (ideally at least two; perhaps three or four) to different ISPs.
2. **Peer proxies:** Each MONET proxy may use one of the other proxies in the system as a conduit for client requests and server responses. These proxies form an overlay network for routing and forwarding requests and responses.
3. **Replicated sites:** Web sites are sometimes replicated on distinct hosts, or are multi-homed using different links to the Internet. The DNS name for a replicated site is often bound to multiple IP addresses in the DNS database. These addresses correspond to distinct client-server paths although portions of the paths may be shared.

For the purposes of the analysis presented in this chapter, we define a *path* to be a sequence of IP addresses that a Web request or response must traverse. For example, the sequence [local-ip, server-ip] defines a path to a server, the sequence [local-ip, proxy-ip, server-ip] defines a path to a server via a proxy, and the sequence [local-ip, proxy-ip] defines a path to a proxy.

When a site announces multiple IP addresses, we assume that each of those addresses represents a different path component: The different addresses either correspond to a different physical machine or to a different Internet connection for the site. Certain deliberate site configurations could violate this assumption, but we believe that such configurations are rare.

If a MONET proxy has ℓ local links and r other single-homed peer proxies it can use, and if the site has s IP addresses, then the total number of potential paths to the Web site at the proxy's disposal is

$\ell \cdot s$ direct paths plus $\ell \cdot r \cdot s$ indirect paths. If each of the peer proxies has ℓ local interfaces of its own, then the number of paths increases to $\ell \cdot s$ direct paths plus $\ell^3 \cdot r \cdot s$ indirect paths. For even moderate values of ℓ, r, and s, this number is considerable; *e.g.*, when $\ell = 3$, $r = 10$, and $s = 2$, the number of possible paths is 546. When the peer proxies are single-homed, this number is 66, still quite large.

Not all of these paths, of course, are truly independent of each other, and pairs of paths may actually share significant common portions. Each path, however, has something different from all the other paths in the set. MONET finds the right small subset of paths from this set to explore in greater depth for any given client request.

We define three requirements that guide MONET's approach:

R1 The network overhead introduced by MONET in terms of the number of extra packets and bytes must be low.

R2 The overhead imposed on Web servers in terms of TCP connections and data download requests must be low.

R3 When possible, MONET should improve user-perceived latency, by reducing the tail of the latency distribution and balancing load on multi-homed links.

The first two requirements preclude an approach that simply attempts concurrent connection requests along all paths between a proxy and Web site.

6.2.1 Approach

Each Web site is named with a DNS-based URL. When presented with a client's request for a URL, MONET follows the procedure shown in Figure 6-2. The MONET proxy first determines whether the requested object is in the proxy's Web cache. If not, then the proxy checks to see whether the site has successfully been contacted recently, and if so, uses an open TCP connection to it, if one already exists.

Otherwise, the proxy uses MONET's *waypoint selection* algorithm to obtain an ordered list of the available paths to the site. This list is in priority order, with each element optionally preceded by a delay. The proxy attempts to retrieve the data in the order suggested by this list, probing each path after the suggested delay. This path exploration could involve contacting one of the site's IP addresses directly from one of the local links, as well as attempting to obtain the data via one of the other MONET proxies.

If waypoint selection lists a peer proxy first, the request is issued immediately. MONET concurrently resolves the site's DNS name to its corresponding IP addresses to determine which paths are available for local interfaces. To mask DNS failures, the proxy attempts this resolution over all of its multi-homed interfaces.

The proxy uses the results of the waypoint selection algorithm and multi-path DNS resolution to probe the servers. To contact the server, the proxy simply sends one TCP connection initialization (SYN) packet on each selected interface. This process may be interspersed with requests via peer proxies, for which MONET extends the Internet Cache Protocol (ICP) [173]. The proxy retrieves data from the first probe (SYN or peer-proxy request) that responds. The results of the DNS queries and path probes update information about path quality maintained by the waypoint selection algorithm.

The MONET approach to masking failures operates on three different time-scales to balance the dual needs of adapting rapidly and of doing so with low overhead. The slowest adaptation (days

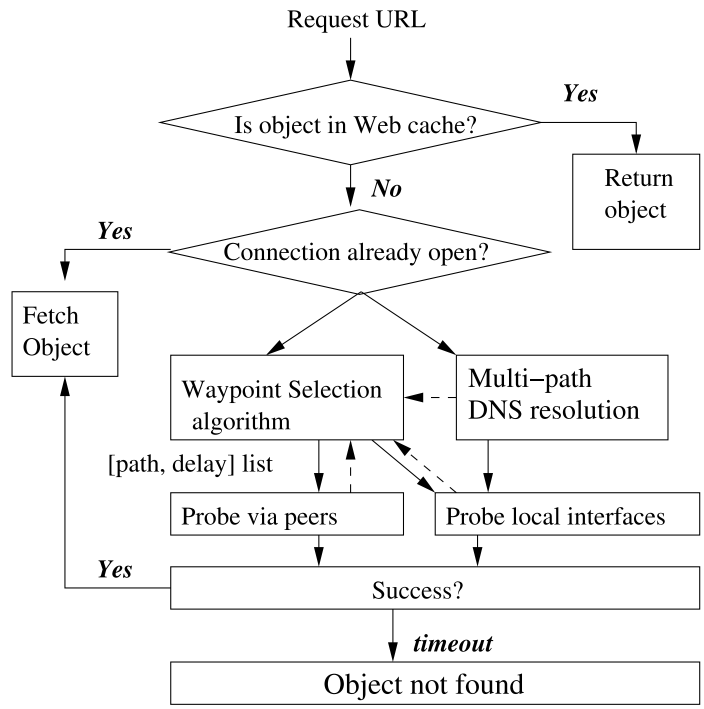


Figure 6-2: Request processing in MONET.

or weeks) involves the deployment of multi-homed local links and peer proxies in different routing domains. Currently, this configuration is updated manually; automating it is an important future task.

The intermediate time scale adaptation, waypoint selection, maintains a history of success rates on the different paths, allowing MONET to adapt the order of path exploration on a time-scale of several seconds.

The proxy generally attempts the first two paths returned by waypoint selection within a few hundred milliseconds, probing the rest of the paths within a few seconds. If this order is good, the chances of a successful download via one of the probed paths is high, since the probe includes setting up the connection to the destination site. This nearly concurrent exploration of paths allows MONET to adapt to failures within a few round-trip times of when they are observed.

Once the proxy has established the connection for a request, it uses the same path. MONET could mask mid-stream failures during large transfers by, for example, issuing an HTTP range request to fetch the remaining content, but the current implementation does not do so. Most typical Web workloads consist of many smaller objects, so failures are more likely to occur between downloads than during them.

6.2.2 The Client-MONET Interface

Clients specify a set of MONET nodes, preferably nodes that are close to them in the network, as their Web proxies (one of these is the primary and the rest are backup proxies). The proxy-based approach allows MONET to be easily and incrementally deployed within an organization, and has turned out to be a critical factor in our ability to attract users and to gather data using live user traffic. In addition to ease of deployment, we chose the proxy approach because it provides two other significant benefits:

1. **Path information**: Because a MONET proxy observes what *site* clients want to contact (such as `www.example.com`), instead of merely seeing a destination IP address, it has access to several more paths for the waypoint selection algorithm to consider when the site is replicated across multiple IP addresses. Moreover, by operating at the application layer and resolving the DNS name of a site to its IP addresses, MONET is able to mask DNS errors; such errors are a non-negligible source of client-perceived site outages and long delays [76, 32].
2. **Access control**: Many sites control access to content based upon the originating IP address, which is changed by using a different local link or by transiting through a remote proxy. Early users of MONET were occasionally unable to access material in licensed scientific journals, because the proxy had redirected their access through a non-licensed IP address. Knowing the site or object ID, instead of just an IP address, enables MONET to force access to licensed content through a local interface. The deployed MONET proxy is now configured to handle access to 180 licensed web sites in this manner. As in the CoDeeN proxies [110], this approach also ensures that clients cannot gain unauthorized access to licensed content via MONET.

There are several alternatives to using a proxy-based interface to MONET. While the proxy implementation provided significant benefits, the alternatives are not without merit.

1. **Application Integration**: Integrating MONET's techniques into an application is relatively straightforward when creating an application from scratch. Depending upon the structure of the application, the difficulty of performing this integration afterwards can range from easy to extremely challenging. A proxy-based system works with all client programs, including those to which we do not have source code, but a long-term architectural shift might best come from the applications themselves. While we chose the proxy solution, we believe that application integration (*e.g.*, into common Web browsers) is a promising long-term deployment path for a subset of MONET's techniques.
2. **Library Interposition**: In many operating systems, it is possible to redirect library calls such as `connect()` and `send()` into a user-supplied library, interposing on the original request stream. Such interposition is facilitated by systems like TESLA [137] that transparently handle the myriad details of correctly interposing on system calls. Using TESLA, we have created a MONET-like library that enables unmodified TCP-based applications (of any type, not just Web applications) to use multiple local interfaces.
   Interposition must draw inferences about path information and application intent (*e.g.*, if the interposing library wanted to contact multiple servers when the application requested a TCP connection, the library would have to infer the identity of those servers from a prior DNS query).
3. **Network Address Translation**: The last method we considered was to interpose on an application's packets as they flow across the network. These functions are commonly performed by a network address translator, or NAT. The interaction between the application and the NAT is similar to that in library interposition—the NAT can change a TCP connection to use multiple interfaces, etc., but can only infer application-level details. Unlike a proxy-based approach, a NAT solution can be deployed without even requiring configuration changes to the application, but such deployment in a large network is more difficult, because the NAT must interpose on all of the traffic in and out of the network.

6.3 Obtaining Multiple Paths

6.3.1 Multi-homed Local Interfaces

A MONET proxy that is directly connected to multiple ISPs can take advantage of local multi-homing when it performs DNS resolution and when it sends requests for Web objects. The MONET proxy is assigned one IP address from each upstream ISP, allowing it to direct requests through any chosen provider.

Obtaining one address per upstream ISP is particularly beneficial for a MONET proxy operating in a small to medium-sized organization because the proxy and its clients gain the benefits of multi-homing without requiring that the organization configure and manage BGP routing with its providers.[1] A multi-homed MONET proxy directs requests only to links that the proxy believes are working, helping to mask access link failures.

MONET uses TCP SYN packets to the server as both a probe and to establish a connection. This dual use reduces network overhead, and has been found to be an effective tactic for choosing between a set of replicated servers [43]. This approach, however, creates additional connection state on origin servers, because several of these connections will not be used to retrieve any data. Section 6.5 discusses a simple optimization to prevent this state from being created.

6.3.2 ICP+: Probing via Peer Proxies

Chapter 5 demonstrated that path redundancy accessed through overlay networks can help mask many end-to-end Internet path failures. Building upon this observation, we augmented MONET's use of multiple local interfaces with a mechanism to gain access to additional paths by way of a network of peer proxies. To probe these paths, we designed ICP+, a backward-compatible extension to the Inter-Cache Protocol (ICP) [172].

ICP was designed to check whether a requested object is in a peer's Web cache. ICP+ extends this check with an enhancement that requests the peer proxy to probe the origin server (using a TCP SYN, as described in Section 6.3.1) and return the round-trip connection establishment time.

An ICP+ query includes the URL of the object that the proxy wants to retrieve through a peer proxy. When a peer proxy receives an ICP+ query, it handles the query just like a request from its clients (Figure 6-2), but the proxy does not contact other peer proxies in turn (*i.e.*, the peer proxy ignores any peer-proxy paths returned by its waypoint selection algorithm). The proxies' ICP+ handling is as follows:

1. If the object is cached, then reply immediately.
2. If an open connection to the server exists, then reply with that connection's establishment RTT.
3. Else, resolve DNS, perform waypoint selection, ignore other peer paths;
   Open a connection to the server;
   Reply with RTT when TCP established.

Figure 6-3 depicts the operation of ICP+. In this figure, the peer proxy does not have the object cached, but does have the IP addresses from the DNS resolution cached. After receiving an ICP+ response, the client proxy fetches the data via the peer proxy. The peer proxy reuses the connection it initiated to the origin server during its probe.

[1] This "host-based" multi-homing aligns with multi-homing techniques suggested for IPv6 [91] and SCTP [31].

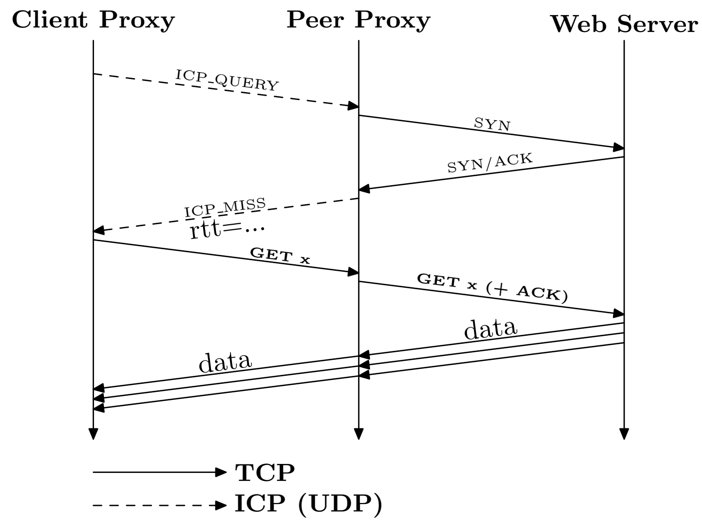


Figure 6-3: The ICP+ protocol retrieving uncached data through a peer.

Because it is used for probing network conditions, ICP+ uses unreliable UDP datagrams to communicate between peer proxies. Using UDP avoids mistaking temporary failures and packet loss for increased latency, as would happen if a reliable transport protocol like TCP were used for the probes. To treat local interfaces and peer-proxy paths consistently, MONET retransmits lost ICP+ messages with the same 3-second timer that TCP uses for its initial SYN packets. Once a peer has confirmed access to a Web site, the proxies use TCP to transmit objects between them.

A multi-homed MONET proxy can transmit multiple ICP+ probes to a peer proxy, one via each of its local interfaces. If the originating proxy knows the IP addresses of a multi-homed peer, then it can also send separate ICP+ probes to a subset of those addresses. The originating proxy tags these ICP+ probes with a unique ID tying the probes to a single client request, so that the peer proxy can aggregate identical requests, sending only a single connection initiation packet and subsequent Web request to the Web server. Doing so permits additional redundancy between proxies without additional server overhead.

The combined operation of a proxy with local interfaces and a peer proxy is illustrated in Figure 6-4. This diagram shows one additional benefit of performing the ICP+ queries in parallel with a connection to the origin server: it eliminates delays that the proxy would ordinarily experience waiting for ICP replies. If the ICP replies for a cached object are delayed, clients might fetch the object directly—the correct behavior if the origin server is closer than the peer proxy.

Caching Web objects improves availability because the cache can serve data when a site is unreachable, but using caching alone to improve availability requires aggressive prefetching [36], which the current MONET prototype does not provide. The benefits of caching, however, are complimentary to MONET: standard cache peering improves delivery of “hot” static objects and frees the servers of the burden of sharing these objects [25, 47, 169]. In contrast, MONET improves end-to-end client-site reachability, facilitating access to uncacheable or unpopular content as well. Such dynamically generated content and Web transactions are commonly found in commercial applications (*e.g.*, online purchases or account access).

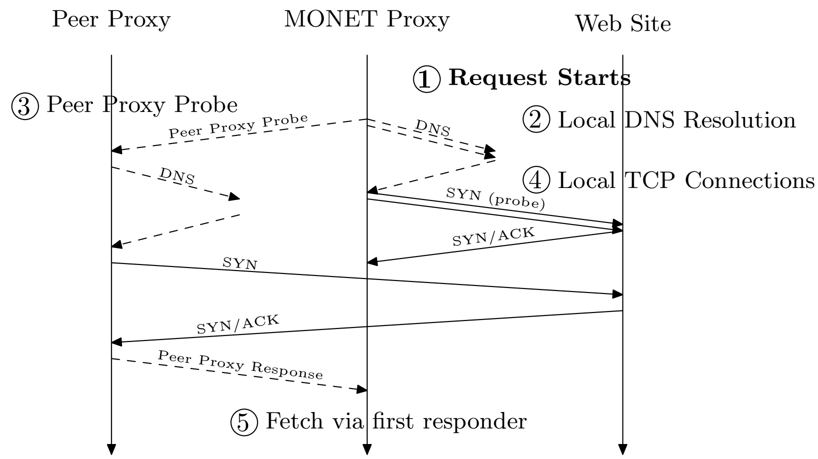


Figure 6-4: The MONET proxy performs several queries in parallel. When the request begins (1), it simultaneously begins DNS resolution (2), and contacts peer proxies for the object (3). After DNS resolution has completed, the MONET proxy attempts TCP connections (delayed by the output of the waypoint selection step) to the remote server via multiple local interfaces (4). The remote proxy performs the same operations and returns a reply to the client proxy. The MONET proxy retrieves the data using the first successful local or peer-proxy probe response (5).

6.3.3 Web Site Multi-homing

If a Web site advertises multiple IP addresses via DNS, a MONET proxy considers them as belonging to different paths during waypoint selection. This approach provides multiple client-service paths even for single-homed proxies. Using multiple IP addresses allows the MONET proxy to recover quickly from some path and server failures. Today's Web clients typically contact only a single address for a Web site, or they wait between 30 seconds and 13 minutes (BSD's default timeout) before "failing over" and contacting subsequent addresses. Because they cannot count on clients to quickly fail over, Web site administrators rely on one of two mechanisms to direct clients to a working server. Many sites use front-end load distributors to direct clients to a cluster of machines. Others answer DNS queries with responses that have very low TTL (time to live) values, forcing clients to frequently refresh the name-to-address mapping for the site. If a sever fails, an automated process removes the failed IP address from the DNS server. MONET does not eliminate the need for large, distributed services (*e.g.*, content distribution networks such as Akamai [2]) to perform geographic mapping to point clients to local servers, but MONET masks failures on shorter timescales without requiring Web sites to set low-TTL DNS records.

6.3.4 Multi-path DNS Resolution

A MONET proxy performs at least two concurrent DNS requests to mask DNS failures for two reasons. First, DNS servers are—or should be—widely replicated, so finding independent paths is easy. Second, sending multiple DNS requests does not cause high network overhead because DNS lookups are much less frequent than TCP connections: in our Web traces, 86% of the connections that the deployed MONET proxy established to remote servers used a cached DNS entry. This number is consistent with other studies of DNS and TCP workloads [76], which estimated that overall cache hit rates were between 70 and 80%.

Because some server-side content distribution services return DNS responses tailored to a client's location in the network, a MONET proxy performs DNS resolution using only its local interfaces. Each peer proxy performs its own DNS resolution. This localized resolution helps the MONET proxies fetch data from a replica near the proxy.

6.4 Waypoint Selection

Because the number of paths available to the proxy may be quite large, MONET uses waypoint selection to pick subsets of its paths to try at different times. The waypoint selection algorithm takes the available local interfaces and peer-proxy paths, as well as target Web site IP addresses, and produces an ordered list of these interfaces and paths. Each element of this list is preceded by an optional delay that specifies the time that should elapse before the corresponding path is attempted. The proxy attempts to connect to the server(s) in the specified order. The waypoint selection algorithm seeks to order paths according to their likelihood of success, but it must also occasionally attempt to use paths that are not the best to determine whether their quality has changed. MONET attempts these secondary paths in parallel with the first path returned by waypoint selection so that the measurement does not increase the client-perceived latency. If the measured path connects first, MONET uses it as it would any other connection.

We define the waypoint selection problem as follows: A client C (*e.g.*, a MONET proxy), wants to retrieve data from a set of k servers $S_1, ..., S_k$ (*e.g.*, multiple replicas of a Web site). The client has a set of local interfaces and peer proxy paths $I_1, ..., I_N$ that it can use to fetch content. The waypoint selection problem is to select an ordering of the I_x, S_y pairs to probe over time. The ordering may attempt to minimize the number of pairs that must be tested to find a functioning path from the client to any server, or to provide the best client-perceived latency, and so on.

Waypoint selection is superficially similar to classical server selection in which a client attempts to pick the best server according to some metric. As Figure 6-5 shows, however, under waypoint selection, a client can use its history of connections to a variety of servers along different paths to infer whether or not those paths are likely to be functioning, and what the path loss probabilities are. Then, when confronted with a request involving a new server, the client can decide which of its paths are best suited to retrieve data.

6.4.1 Which Paths to Probe

MONET uses a simple algorithm to rank the different paths by which the proxy can contact a Web site. The ranking tracks the success rate achieved using each local link and each local link-peer proxy pair. MONET maintains an exponential weighted moving average (EWMA) of the success rate along each of these paths. The algorithm updates the success rate for a local link, Success[path], within a short timeout of the proxy's sending a TCP SYN or DNS request using that link. The timeout is set to three seconds or to the average response time required for that link, whichever is shorter. MONET's computation of average response times is described in the next subsection. Three seconds represents the minimum TCP SYN retransmission interval, and thus represents a conservative choice for determining that a request has failed. The algorithm updates the Success[path] for a peer proxy after the proxy sends an ICP+ query to that peer:

$$\begin{aligned} \text{Success[path]} &\leftarrow q * \text{Success[path]} + (1-q) * \text{status} \\ 0.90 \leq q &\leq 0.99 \end{aligned}$$

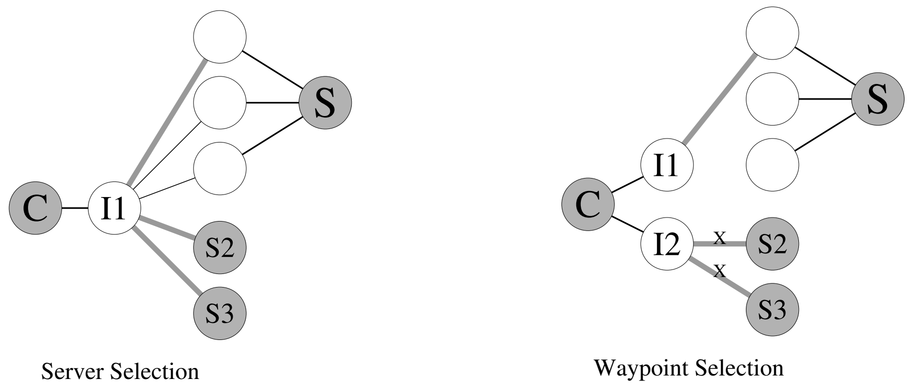


Figure 6-5: A server selection client has only a single outbound link to choose from when contacting a replicated site. Failures to other servers (S2 and S3) give the client no information about accesses to new servers. In contrast, the two failures shown in in the waypoint selection diagram permit the client to infer that the lower link is likely to be faulty, allowing the client to direct more of its future requests via the known good link.

When two paths in the EWMA predictor have the same success rate, the path with the lower average response time is ranked higher.

The proxy generally sends requests via the paths in order of best predicted success rate. Strictly following this order, however, risks never again sampling previously lossy paths, and the proxy would not know when those paths had improved. Hence, the proxy periodically samples paths independent of their rank. It sends all DNS requests both on the local link with the highest success rate and also via a randomly selected second local link (if one exists). For TCP and ICP+ requests, the proxy also attempts to contact the site using a random link between 1% and 10% of the time to ensure that infrequently used paths are still measured.

In designing MONET's waypoint selection algorithm, we considered only schemes that rank the local links and peer proxy paths, regardless of which servers were previously accessed along the various paths. It is possible that maintaining the success rates grouped by the remote site name or IP prefix could yield additional benefit.

6.4.2 When to Probe Paths

A MONET proxy should perform the next request attempt only when it is likely that each prior attempt has failed. The delay between requests on different paths must be long enough to ensure this behavior, but short enough to allow connections to connect and to retrieve data without undue delay.

A delay threshold estimator should adapt to the distribution of round-trip *connect times* to various servers observed along different paths. Measurements of these connect times from the operational MONET proxy at MIT show that this distribution is multi-peaked (the "knee" on the CDF, and the peaks in the histogram in Figure 6-6), suggesting that the best delay threshold is just after one of the peaks. The peak should correspond to a sufficiently low "tail" probability of a large connect time being observed. For example, in this figure, very few arrivals occur between 0.6 and 3.1 seconds; choosing a threshold over 0.6 seconds increases delay without significantly decreasing the overhead of a prematurely triggered connection attempt.

We explored two ways of estimating this delay threshold:

1. **k-means clustering.** This method identifies the peaks in the PDF by clustering the connect time samples into k clusters, and finding a percentile cutoff lying just outside one of the peaks (clusters). The centroids found by k-means with $k = 16$ are shown as horizontal lines in Figure 6-6. The clustering is relatively insensitive to the value of k.

This method is computationally expensive, particularly if the clustering is recomputed each time a connection attempt succeeds or fails. Even when the threshold is only recomputed periodically, the computational load and memory requirements may exceed what is acceptable for a busy proxy: the k-means clustering requires that the proxy maintain a large history of previous probes.

2. ***rttvar*-based scheme.** To avoid the cost of the k-means scheme, we considered an *rttvar* scheme inspired by TCP retransmission timers. Each delay sample, independent of the server contacted or path used, updates an EWMA estimate of the average delay (*rtt*) and another EWMA estimate of the average linear deviation of the delay (*rttvar*). The delay threshold between subsequent requests is set to *rtt* $+ 4\cdot$ *rttvar*:

$$\begin{aligned}
rtt &\leftarrow q * rtt + (1-q) * sample \\
err &\leftarrow |rtt - sample| \\
rttvar &\leftarrow q_2 * rttvar + (1-q_2) * err \\
thresh &\leftarrow rtt + 4 * rttvar
\end{aligned}$$

The *rttvar* scheme is substantially simpler to calculate than is k-means clustering, but it may pick a threshold that is in the middle of a "valley" between two peaks in the delay sample distribution. In practice, measurements from MONET (*e.g.*, the data illustrated in Figure 6-6) show that *rttvar* estimates an 800 ms delay threshold, while k-means estimates thresholds of 295 ms (2% false transmission probability), 750 ms (1.6%), and 3.2s (1%). A MONET using the 2% k-means estimator would decide that its first connection had failed after 300 ms instead of 800 ms, reducing the failover time for the failed connection. We do not believe that this modest latency improvement justifies the complexity and increased computational and storage requirements of the k-means estimation, and so we chose the rttvar scheme for MONET.

6.5 Reducing Server Overhead

Waypoint selection greatly reduces the number of wasteful connection attempts. MONET must also ensure that the few remaining connection attempts do not unnecessarily create server state. Because modern servers minimize processing of SYN packets (to thwart denial-of-service attacks) using techniques like SYN Cookies [19] and SYN caches [90], MONET can send multiple SYN packets without incurring serious overhead, as long as *exactly one TCP three-way handshake completes*, since a connection consumes significant server resources once the server receives the final ACK in the three-way TCP handshake. This section describes our proposed mechanism for ensuring that this condition is met.

To ensure that only one handshake completes, the MONET proxy acknowledges only one SYN ACK. It then either resets the remaining connections or keeps them in a pending connection cache for future use.[2]

[2] TCP does not use data-less packets for RTT estimation, so deferring the ACK does not affect its round-trip time calculations (see [155] pp. 299–305).

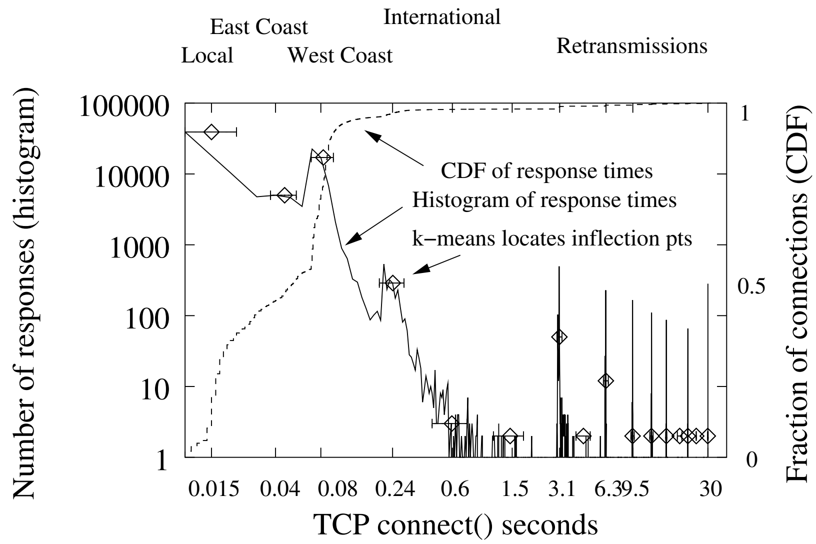


Figure 6-6: k-means clustering applied to TCP response times for 137,000 connections from one access link on the east coast. The CDF shows the cumulative fraction of requests amassed by the histogram.

To give the proxy control over the completion of the three-way handshake and SYN retransmissions, we propose—but have not yet implemented—a new TCP socket option, `TCP_CONTROL_DEFER`. This socket option lets applications inform the kernel when, if at all, it should reply to TCP control packets. MONET would benefit from `TCP_CONTROL_DEFER` in two ways:

1. The proxy can use TCP connections as probes without completely opening the connection.
2. The proxy can change servers if a SYN is lost, instead of retransmitting SYNs to the same server.[3] Doing so permits the application to select a new server as soon as TCP can retransmit the lost SYN packet (three seconds), instead of waiting until the TCP connection times out. Figure 6-7 shows a trace of a Web proxy contacting an unreachable server. Without `TCP_CONTROL_DEFER`, the proxy wastes nearly 45 seconds contacting an unresponsive address. `TCP_CONTROL_DEFER` reduces this wasted time without increasing the overhead.

Controlling the duration of TCP's connect attempts by explicitly aborting the connection can render far-away servers unreachable. For example, consider a client that implemented connection switching by closing connections after three seconds and attempting a new connection. Such a client would be unable to contact a server whose round-trip time was greater than three seconds—all of the client's connections would be closed before they succeeded. Using `TCP_CONTROL_DEFER`, applications can "pause" the socket that lost the SYN and open a new one, either of which can receive the SYN ACK if it is delayed.

The security implications of this new socket option are not severe. TCP deferral does not provide extra capabilities to a user with administrative privileges, but it could empower unprivileged users to perform "stealth" SYN scans or to attempt to SYN flood a remote machine. These capabilities, however, are already readily available, and limiting an unprivileged user's connection attempts is

[3] In the deployed MONET system, 99% of SYN packets to an eventually reachable server returned in well under 1 second (Chapter 7). Currently, TCP `connect()` may wait for as long as *13 minutes* before indicating failure to the client application, so this new socket option is quite useful.

Time	source	dest	Type
02:54:31	**client.3430**	> **server-A.80**	SYN
02:54:34	client.3430	> server-A.80	SYN
02:54:37	client.3430	> server-A.80	SYN
		...	
02:54:44	client.3430	> server-A.80	SYN
02:54:47	client.3430	> server-A.80	SYN
02:55:05	client.3430	> server-A.80	SYN
02:55:17	**client.3432**	> **server-B.80**	SYN

Figure 6-7: tcpdump of Squid proxy connecting to an unreachable server with two addresses. After 30 seconds and 7 retransmissions, the `connect` fails, and Squid attempts to contact the second server address.

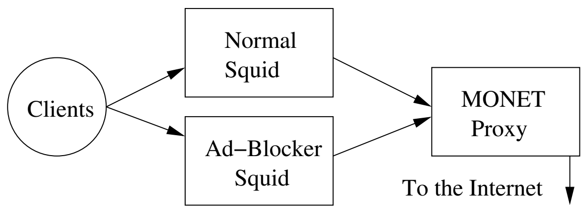


Figure 6-8: The squid configuration for the deployed MONET

relatively simple.

6.6 Implementation

The MONET proxy is implemented as a set of changes to the Squid Web proxy [152] and the `pdnsd` parallel DNS resolver [101], along with a set of host policy routing configurations to support explicit multi-homing. MONET runs under the FreeBSD and Linux operating systems.[4]

In the deployed system, Web client configurations are specified with a short Javascript program that arranges for a suitable backup proxy from the specified set to be used if the primary proxy fails. To further isolate clients from proxy crashes during development of the MONET proxy, all client accesses pass through an unmodified front-end proxy that forwards all requests to the MONET proxy unless it is down. As an extra incentive for users to use the MONET proxy, one front-end blocks common banner ads and pop-up advertisements. Figure 6-8 shows the Squid configuration.

Because we wanted to evaluate multiple waypoint selection algorithms, the deployed proxy probes *all* of its paths in parallel without performing waypoint selection. We then used subsets of this all-paths data to determine the performance of the waypoint selection algorithms. The currently deployed waypoint selection algorithm returns a static list of (path, delay) pairs that it chooses based upon the name of the destination Web site, to address the access control problems mentioned in Section 6.2.2.

[4]MONET should run on any POSIX-compliant operating system so long as that OS provides a way for an application to choose which upstream link to use for a particular TCP connection. MONET accomplishes this link selection in FreeBSD and Linux by binding TCP connections to a particular IP address and then using the host's policy routing framework to direct packets from that address to a particular link. This link selection could also be accomplished using policy routing on a router upstream from the host running the proxy.

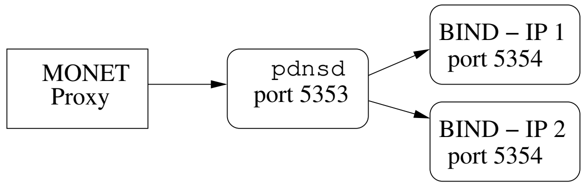


Figure 6-9: The DNS configuration. `pdnsd` sends queries in parallel to each BIND server, which resolves the query independently.

6.7 Explicit Multi-homing

The MONET proxy and DNS server explicitly bind to the IP address of each physical interface on the machine. MONET uses FreeBSD's `ipfw` firewall rules or Linux's policy routing to direct packets originating from a particular address through the correct upstream link for that interface.

The MONET proxy communicates with a front-end DNS server, `pdnsd`, running on a non-standard high port. `pdnsd` is a DNS server that does not recursively resolve requests on its own, but instead forwards client requests to several recursive DNS servers in parallel—in our case, to BIND, the Berkeley Internet Name Daemon [7]. An instance of BIND runs on each local interface, as shown in Figure 6-9. This configuration resolves each DNS query independently on each of the outbound interfaces, so that we can confirm during analysis whether the query would have succeeded or failed if sent on that interface alone.[5] Because each BIND resolves the query independently, our implementation does not explicitly force the queries to go to different servers. BIND does, however, rotate through the list of available name servers. Because most domains are configured with at least two name servers [44], MONET usually copes with the failure of one of its links or of a remote DNS server with no additional delay.

6.8 ICP+ with Connection Setup

As explained in Section 6.3.2, ICP+ is a backwards-compatible extension to ICP, the Inter-Cache Protocol, which is standardized in RFC 2187 [172]. ICP is a query/response protocol that one cache uses to determine whether a peer cache has a particular object. The requesting cache sends an ICP packet of type `ICP_QUERY` to its peer, which responds with either `ICP_HIT` or `ICP_MISS`.[6] If the peer indicates that it has the object, the querying cache will open a TCP connection to the peer cache to request delivery of the object.

ICP+ adds two new flags to the `ICP_QUERY` message: `ICP_FLAG_SETUP` and `ICP_FLAG_SETUP_PCONN`. A query with `ICP_FLAG_SETUP` requests that the remote proxy attempt a TCP connection to the origin server before returning an `ICP_MISS`. Peer caches that do not support ICP+—or that do not wish to provide ICP+ to that client—simply ignore the flag and reply with standard ICP semantics. Squid supports a mechanism for occasionally sending ICMP ping packets to origin servers, using ICP's option data field to return this ping time in response to

[5] Our first multiple link DNS implementation shared a response cache between the interfaces, which permitted "short-circuiting" a lost response, but made it impossible to analyze the success or failure of the links independently.

[6] A number of other ICP responses are used for error handling and other conditions.

Opcode	Version	Message Length
Request Number		
Options		
Option Data [hops + rtt]		
Sender Host Address		
Payload [URL] ...		

Figure 6-10: The ICP Packet Format. The overall format is the same as that used in ICP; bold type indicates fields overloaded or extended to support ICP+. Brackets show the contents of the fields for Web proxy communication.

an ICP query. ICP+ piggybacks upon this mechanism to return the measured RTT from connection initiation. Figure 6-3 in Section 6.3.2 showed a time-line of an ICP+ query.

MONET uses Squid's persistent connection cache to reduce connection setup overhead. If the originating proxy has a persistent connection open to a Web site, it bypasses peer selection and directly uses the persistent connection, on the assumption that in one of the previous selection attempts, its own connection seemed best. This approach gradually shifts subsequent connections to the direct interfaces, which reduces overlay overhead when the entire network is functioning well. When a remote proxy has a persistent connection to the origin server, it responds immediately to ICP queries, setting the `ICP_FLAG_SETUP_PCONN` flag, and supplying the RTT from when it initially opened the connection.

Figure 6-10 shows the ICP packet header with the MONET additions in bold. RFC 2187 notes that the sender host address is normally zero-filled. ICP+ uses this field and the request number to suppress duplicates. On startup, each MONET proxy picks a 32-bit number as its sender ID (*e.g.*, a random number or a local interface address), and uses the same ID when sending via any of its interfaces. The $(sender\ ID, request\ \#)$ tuple uniquely identifies each request and allows a peer proxy not to send multiple identical requests to a Web server.

ICP suffers from some known security flaws stemming from its use of connectionless UDP packets together with its lack of authentication. Although these flaws are already known in ICP, ICP+ makes it possible to use the MONET proxies as DoS reflectors. These flaws in ICP are corrected by the newer UDP-based HyperText Caching Protocol (HTCP) [168]. HTCP supports strong authentication of requests, and HTCP requests can specify additional request attributes (such as cookies) that may affect whether an object can be served from cache or not. We have extended HTCP in a similar fashion to our extension of ICP to create HTCP+.

Unlike ICP+, HTCP+ is not transparently compatible with non-HTCP+ proxies. HTCP does not provide a "sender ID" field; MONET uses the "Keyname" field for this purpose.[7] The HTCP+ implementation uses a new "Response" code to indicate the `SETUP` operation and returns RTTs as an additional Cache-Header field.

The MONET extensions to Squid update Squid's `http_access` access control list with a temporary "permit" when an authenticated HTCP packet arrives. Aside from these implementation

[7]This use is, perhaps, unfortunate; a future version of HTCP should make space for an ID.

differences, the HTCP-based MONET is functionally identical to the ICP-based version. The deployed system uses ICP+ because that implementation is more mature.

6.9 Summary

This chapter presented the design and implementation of MONET, a Web proxy system to improve the end-to-end client-perceived availability of accesses to Web sites. MONET masks several kinds of failures that prevent clients from connecting to Web sites, including access link failures, Internet routing failures, DNS failures, and a subset of server-side failures. MONET masks these failures by obtaining and exploring multiple paths between the proxy and Web sites. These paths go out via the proxy's multi-homed local links, through peer MONET proxies, and to multiple server IP addresses. MONET incorporates a waypoint selection algorithm that allows a proxy to explore these different paths with little overhead, while also achieving quick failure recovery, usually within a few round-trip times.

The next chapter evaluates the performance of MONET's failure masking algorithms by studying the behavior of an operational MONET that has been in daily use at several sites for 18 months.

I'll try anything once, twice if I like it, three times to make sure.

- Mae West

Chapter 7

MONET Evaluation

MONET is designed to improve the ability of clients to access Web sites. This chapter analyzes a dataset collected during the 18-month deployment of MONET, in order to examine how well its techniques fare against real-world failures.

The evaluation focuses on the number of "nines" of availability (*cf.* Chapter 1) achieved with and without MONET. It does so by addressing the following questions:

1. To what extent do subsystems such as DNS, access links, *etc.* contribute to failures incurred while attempting to access Web sites?
2. How well does MONET mask failures, what is its overhead, and how does it compare against a scheme that explores all available paths concurrently?
3. What aspects (physical multi-homing, peer proxies, *etc.*) of MONET's design contribute to MONET's demonstrated improvement in availability? Is MONET useful if BGP multi-homing is employed?
4. How much more of an availability improvement does MONET provide if the Web site is multi-homed?

7.1 MONET Testbed and Data Collection

We deployed the MONET proxy at six sites from the RON testbed, listed in Table 7.1. The analysis presented in this chapter examines requests sourced from two of these proxies, CSAIL and Mazu, both of which are physically multi-homed. The CSAIL proxy uses three different local links:

Site	**Connectivity**	**Times**
CSAIL	3: 2x100Mb, 1.5Mb DSL	6 Dec - 27 Jan 2004
Mazu	2: T1, 1.5Mb wireless	24 Jan - 4 Feb 2004
Utah	1 (US university - West)	proxy-only
Aros	1 (US local ISP - West)	proxy only
NYU	1 (US university - East)	proxy-only
Mediaone	2: DSL, Cable	22 Sep - 14 Oct 2004

Table 7.1: The sites at which the MONET proxy was deployed. Mazu's wireless connection uses a commercial wireless provider. The cable modem site has been operational for one year, but was monitored only briefly.

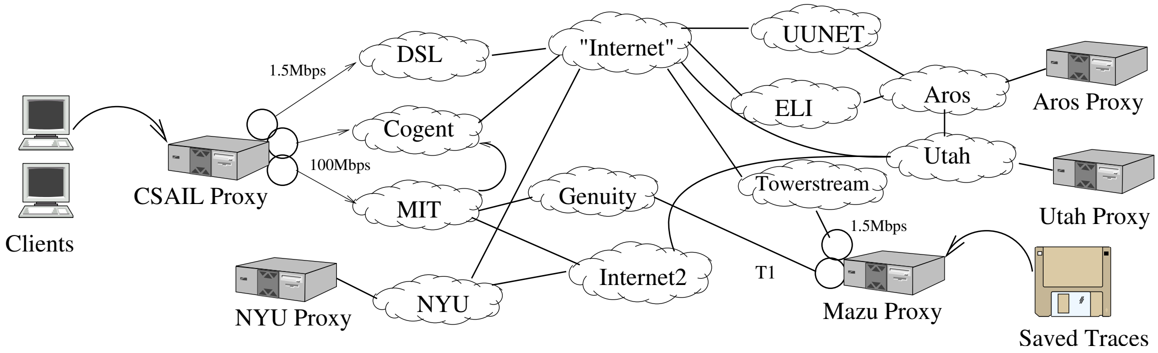


Figure 7-1: A partial AS-level view of the network connections between five of the deployed MONET proxies (the mediaone and CMU proxies are not shown). The CSAIL proxy peers with NYU, Utah, and Aros; the Mazu proxy peers with CSAIL, Aros, and NYU. The other sites are not directly multi-homed and do not have a significant number of local users; their traces are omitted from the analysis.

Request type	Count
Client objects fetched	2.1M
Cache misses	1.3M
Client bytes fetched	28.5 GByte
Cache bytes missed	27.5 GByte
TCP Connections	616,536
Web Sessions	137,341
DNS lookups	82,957

Table 7.2: CSAIL proxy traffic statistics.

1. **MIT:** A 100 Mbits/s link to MIT's network. MIT is itself served by three different upstream ISPs using BGP multi-homing.
2. **Cog:** A 100 Mbits/s link from Cogent.
3. **DSL:** A 1.5 Mbits/s (downstream), 384 Kbits/s (upstream) DSL link from Speakeasy.

The Mazu proxy uses two different physical access links: a 1.5 Mbits/s T1 link from Genuity, and a 1.5 Mbits/s wireless link from Towerstream. Figure 7-1 shows the AS-level view of topologies between our deployed proxies.

The CSAIL proxy has the largest client base, serving about fifty different IP addresses every day. It has been running since April 2003; the analysis here focuses on data collected during a six-week period from December 6, 2003 until January 27, 2004. The CSAIL proxy had three peers during this trace. Table 7.2 shows the traffic statistics for the CSAIL proxy.

The MONET proxies record the following events:

1. **Client request time:** The time at which the client (or peer) request arrived at the proxy, and, if the request was served, the time at which the HTTP response was sent to the requester. For uncached objects, the proxy also maintains records of the following three categories of events.
2. **DNS resolution duration:** The time at which the proxy made a request to `pdnsd`. For uncached DNS responses, the time at which DNS requests were sent on each local link, and

the times at which the corresponding responses were received (if at all).

3. **TCP connect duration:** The time at which TCP SYN packets were transmitted and the times at which either the TCP `connect()` call completed, or a TCP connection reset (RST) packet was received, for each local link.
4. **ICP+ duration:** The time at which the proxy sent an ICP+ message to a peer proxy, the time at which it was received by the peer proxy, and the time at which the ICP+ response returned.

When the proxy receives a request for an object from a Web site, it attempts to contact the Web site using *all* of its local interfaces and all of its peer proxies. The MONET proxy records the time at which the original request was received and the times at which the connection establishment steps occurred using each of the local interfaces and peer proxies. The analysis later in this chapter uses the record of those times to examine the performance of a proxy that had access only to a subset of the available interfaces. Because the proxy uses all of its interfaces at the same time, the analysis simulates the effects of different waypoint selection algorithms using the original timing record and introducing various delays before each component is used.

We make a few observations about the data collected from the MONET proxies:

Caching effects: The cache experienced a 37% hit rate for valid objects, saving about 3.5% of the requested bytes. As in previous studies, a few large transfers dominated the proxy's byte-count, while the majority of the sessions consisted of smaller requests. These cache hits reduce user-perceived delays, but do not mask many outages: numerous pages either required server revalidation, or included uncached objects.

Sessions: We primarily examine the success or failure of a *session*, defined as the first request to a particular server for a Web site after 60 seconds or more of inactivity.[1] Analyzing sessions avoids a significant bias: An unreachable server generates only a *single* failed request, but a successful connection generates a stream of subsequent requests. The proxy also uses persistent connections to fetch multiple objects from the same Web server, which reduces the total number of TCP connections. The proxy attempted 616,437 connections to external Web sites over 137,341 sessions.

Excluded objects: The following requests were excluded from analysis: Web sites within MIT, objects cached at remote proxies, accesses to unqualified domain names and domain names that returned "no such domain" (NXDOMAIN), access to subscription-based Web sites for which the proxy performs non-standard handling, and accesses to nine Web sites that consistently returned invalid DNS entries or consistently exhibited other anomalies.[2] Excluding NXDOMAIN requests ignores some classes of misconfiguration-based DNS failures. We also note that an internal network failure at the proxy site can prevent users' requests from reaching the proxy. We therefore missed network failures that coincided with client failures (*e.g.*, power failures).

We do not claim that the performance of these five Internet links at MIT and Mazu represents that of a "typical" Internet-connected site. In fact, MONET would likely be used in much worse situations than those we studied to group a set of affordable low-quality links into a highly reliable system. Our measurements do, however, represent an interesting range of link reliability, quality, and bandwidth, and suggest that MONET would likely benefit many common network configurations.

[1] The gaps between requests from a single user to one Web site are usually under 60 seconds [84], pp. 394.

[2] Several sites had persistent DNS lame delegations; another site always returned `0.0.0.0` as its IP address. One site returned DNS names that exceeded the length of the tcpdump capture used to obtain the DNS packets.

7.1.1 On-line Trace vs. Probes

An important question we faced in our system evaluation was whether to collect data on-line from active users, or to collect the data using traces and randomized probes. Because the goal of MONET is to mask failures that negatively affect users, we chose to evaluate our system primarily using on-line data. There are several consequences of this choice:

First, our evaluation uses more data from accesses to popular sites than from average or randomly chosen sites. Popular sites typically exhibit more stable routing behavior than does the average site [130]. Second, users tend to leave sites that exhibit poor response times, leading to a further trend towards more samples from sites with good performance. Finally, ISPs and some Web sites schedule maintenance and down-time during off hours and announce these times in advance.[3] The analysis counts failures during such maintenance windows if they have a negative effect on users, but presumably, if the outages were scheduled well, the number of failures due to these planned events is reduced.

7.2 Characterizing Failures

The failures observed by MONET fall into five categories, listed below. We were able to precisely determine the category for each of the 5,201 failures listed in Table 7.3 because MONET proxies are connected to a highly reliable ensemble of links—there were no times at which they all failed. The categories of observed failures are:

1. **DNS**: The DNS servers for the domain were unreachable or had failed. The originating proxy could contact multiple peer proxies, and no local links or peers could resolve the domain.
2. **Site RST**: The site was reachable because a proxy saw at least one TCP RST ("Reset") from a server for the site being contacted, but no connection succeeded on any local interface, and no peer proxy was able to retrieve the data. TCP RST packets indicate that the server was unable to accept the TCP connection.
3. **Site unreachable**: The site or its network was unreachable from multiple vantage points. The originating proxy contacted at least two peer proxies with at least two packets each, but none elicited a response from the site.
4. **Client Access**: One (or more) of the originating proxy's access links was unable to reach the rest of the Internet. The originating proxy could not use that link to perform DNS resolution (if the DNS response was not cached), to establish a TCP session to a server for the Web site, or to contact any of its peer proxies.
5. **Wide-area**: A link at the originating proxy was working, but the proxy could not use that link either to perform DNS resolution or to contact a server for the desired Web site. Other links and proxies *could* resolve and contact the site, suggesting that the failure was not at either the client access link or the server.

7.2.1 DNS Total Failures

It was rare that the MONET proxy was unable to eventually resolve the IP address(es) for a Web site; such failures occurred exactly once at each proxy. Our analysis of the MONET proxy data filtered out ten hosts with persistent DNS misconfigurations (*e.g.*, always returning "0.0.0.0"), but we were still surprised at how few failures were caused by unreachable DNS servers. In both of the DNS

[3] For example, the DSL provider for the CSAIL DSL link announced six multiple-hour scheduled maintenance windows during our measurements.

	CSAIL 137,612 sessions			Mazu 9,945 sessions	
Failure Type	**MIT**	**Cog**	**DSL**	**T1**	**Wi**
DNS	1	1	1	1	1
Site RST	50	50	50	2	2
Site Unreach	173	173	173	21	21
Client Access	152	14	2016	0	5
Wide-area	201	238	1828	14	13
Availability	99.6%	99.7%	97.0%	99.7%	99.6%

Table 7.3: Observed failures at two MONET proxy sites. The failures are further characterized by the physical link on which they manifested at the proxy, when possible. The DNS, Site RST, and Site Unreach rows each represent per-site characteristics, and are therefore the same for each link at a particular proxy. The client access and wide area failures occur on a per-link basis.

failures, all of the DNS servers for the domain were attached to the same last-hop router, contrary to the recommendations in RFC 2182 [44]. In the first failure, the Web server was co-located with the DNS servers, so a failure of the link to the DNS servers would also have caused the Web site to be unavailable. In the second case, however, the Web server was located at a different ISP from the failed name servers. The low incidence of DNS failures suggests that, in our data sets, the typical redundancy and geographic replication present in the DNS is sufficient to guard against nearly all total DNS failures when used by a client that contacts multiple DNS servers. The main win with MONET with respect to DNS is that MONET can reduce long delays.

7.2.2 Server Failures

With the exception of the DSL link, server failures contributed more to the measured unavailability in our data than did local access link failures.

We define a server as having failed when no connection attempts to it succeed, but the proxy is able to contact two or more peers with at least two packets being received by each peer. It is possible that the network was partitioned between all proxies and the destination server, but it is more likely that the problem lay with the destination server or its nearby links. Table 7.3 shows that there were 173 instances in which the CSAIL proxy was unable to reach a particular Web site, but was able to contact its peer proxies and other servers.

If the proxy receives TCP RSTs from a server for a Web site, we know that the server host or program was at fault, not the network. The proxies therefore recorded whether they received TCP RSTs from the failed server. A RST indicates that the server was reachable, so we separated this category from that of unreachable servers. Roughly 20% of the identified server failures sent RSTs to the CSAIL proxy, and about 10% sent RSTs to the Mazu proxy. In total, 223 session attempts from the CSAIL proxy failed to reach the server or received only TCP RSTs.

Because of peer proxy restarts and crashes, 8.2% of sessions at the CSAIL proxy never contacted a peer proxy. Our analysis thus underestimates the benefits from the overlay, and undercounts the number of server failures by a small margin. We would expect to miss about 8.2% (18) of the 223 server failures. In 6.3% (14) instances, MONET could not reach any peers or the server. In our later analysis, most of these instances are probably incorrectly identified as MONET failures instead of server failures. Supporting this conclusion, the proxies observed RSTs from three of the servers in these instances of "MONET failures," similar to the 20% RST rate with the identified server failures. We believe, therefore, there were no instances in which the proxies were unable to reach a functioning server—not surprising, given the number and quality of links involved.

In order to determine whether this analysis correctly identified failed servers, we re-checked these servers availability two weeks after the first data collection period. 40% of failed servers were *still* unreachable after two weeks. We conclude that the analysis was correct in identifying at least a portion of the servers as having failed. Many of the observed server failures were probably due to an attempt to contact a failed or nonexistent site.

The overall goal of this evaluation is to understand how well MONET could perform when contacting a highly reliable service; a Web site hosted on a highly failure-prone access link already has a clear bottleneck. To better understand how MONET could perform, and to eliminate the effects of "permanently" failed Web sites, the remainder of this analysis excludes positively identified server-side failures. To put these numbers in perspective, Section 7.2.2 examines the (server failure-included) performance of MONET to both all servers, and to a more reliable subset of multi-homed services.

7.2.3 Client access link failures

When *no* requests via a particular client link elicit a response—DNS, TCP, or ICP+ queries to peers—we consider it a client access link failure. Most links, other than the DSL line, displayed good availability—near 99.9%. Such high link availability is to be expected in the environments we measured; for example, MIT is connected to the Internet through BGP multi-homing to three upstream ISPs. The remaining periods of unavailability occurred despite the relatively high availability of the links themselves. BGP multi-homing greatly reduces complete client-side outages, but does not provide an end-to-end solution to failures or problems that occur in the middle of the network or close to the server.

We observed one ten-hour failure of the Mazu wireless link during two weeks of monitoring, but it occurred from 9:45pm until 7:45am when little traffic was being replayed through the proxy. The DSL link experienced one 14-hour failure and numerous smaller failures over several months.

7.2.4 Wide-area failures

Wide-area failures typically represent a communication failure between a specific client access link and a Web site that did not disrupt either the client's or the site's ability to communicate with other hosts. In about 30 of the wide-area failures at the CSAIL proxy and 2 of the failures at the Mazu proxy, the proxy received TCP RSTs from a server for the Web site, but the proxy was still able to successfully connect to the server over other links. Observing a successful connection *and* a TCP RST may occur because of the failure of one of several servers for the Web site, possibly hidden behind a load-balancing front-end. We classified these cases as wide-area failures because they did not impair all of the links (and thus, MONET was able to mask the failures), but they might also be classified as server failures.

Our data does not contain sufficient information to explain why the wide-area failures occurred, but the likely causes include: (1) Wide-area routing problems such as delayed BGP convergence; (2) Routing misconfiguration [92, 107]; or (3) persistent congestion or DoS traffic that affected only a subset of the paths between the proxy and the Web site.

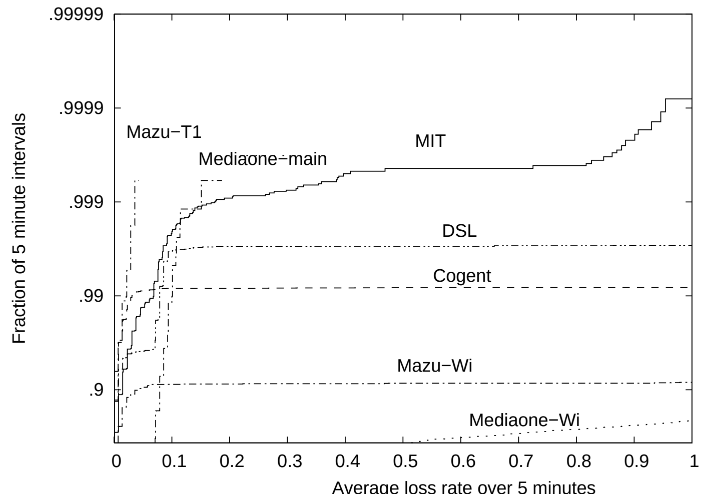


Figure 7-2: The CDF of success rates for DNS root server monitoring. The monitoring of the MIT and DSL links was considerably longer than the monitoring of the other links.

7.2.5 Estimating total outages

The evaluation of MONET examines failures experienced by users during an attempt to access a Web site. This section examines how such failures relate to the overall failure rate of the access links connecting the Web proxy to the Internet.

As noted in Section 7.1.1, observations of failures driven by client traces are biased by user behavior and temporal patterns. Planned network downtimes and maintenance are typically scheduled for times when user activity is at its lowest—for example, the Towerstream failure we observed lasted overnight, but was repaired at 7:54 a.m. As a counterpoint to examining failures that affect user traffic, we also monitored the "global" reachability of each link we studied.

A measurement tool sent a TCP ping[4] every two seconds to one of the 13 DNS root servers, a Google server, or a Yahoo! server in California.[5] We deemed it a total outage if we experienced 90% packet loss to *all* of these hosts for more than a minute. We used the 90% packet loss threshold instead of 100% so that the analysis identifies failures that are interrupted by extremely short recovery periods, and to identify failures when one root server very near MIT was reachable, but the rest were not. The 90% threshold requires that at least two of the 13 servers be reachable. This choice of threshold is somewhat arbitrary—we merely desired a threshold that would clearly identify periods during which the network was effectively useless.

The resulting data represents a "best case" reliability figure for each link. Being able to reach two root servers is an easy condition to meet: the root servers are highly reliable, are at diverse network locations, and many of the root "servers" are, in fact, confederations of many machines that all provide the same service. The paths to the root servers quickly diverge, making use of all of a

[4]The tool sent a TCP "ACK" packet to the server, causing the server to respond with a RST packet because the ACK was not associated with an existing connection. Such TCP probes are often able to reach servers that filter ICMP ping packets.

[5]We removed `l.root-servers.net` from the list because it experienced several times as many losses as any other root name server.

	Link		
failure duration	**MIT**	**Cogent**	**DSL**
30 s	36	389	58
1 min	0	29	0
2 min	0	4	0
5 min	0	0	1
10 min	0	0	0
30 min	0	0	3
1 hour	0	0	0
2 hours	0	0	0
5 hours	0	1	0

Table 7.4: The duration of failures observed during long-term root server monitoring. These data are from a longer monitoring period than the data in Figure 7-2, and so do not directly correspond to failures observed during the MONET data sets analyzed in this chapter. Note that the *durations* during which each link was monitored differ; only the *distribution* can be compared, not the absolute numbers.

site's upstream Internet links, so the resulting reliability figures are largely independent of transient failures to a single host and to route changes because of the failure of one of the upstream links.

During the six week observation period, MIT's Internet link experienced 10 complete outages that lasted longer than one minute, representing about 84 minutes of failures. Complete outage downtime was a relatively low 0.11% ($\sim$ 99.9% uptime), which corresponds closely with the 0.11% client side failure rate (152 failures) we observed for the CSAIL proxy. We observed HTTP traffic through the proxy during most of these outages, suggesting that they were not due to internal network disruption.

Figure 7-2 shows the overall availability packet loss metrics for the network links we directly observed. There were fewer samples for the Cogent, Mazu, and Mediaone links, so the resolution on their actual availability is lower than for the MIT link. The DSL link experienced several hours of total outages during the monitoring period. The Towerstream wireless link experienced a large outage during its short monitoring period. Table 7.4 shows the distribution of the failure durations observed *during a different and longer monitoring period*. These failures do not correspond to those that occurred during the MONET proxy monitoring, but do provide a hint at the duration of failures that MONET must mask. Both the Cogent and DSL links experienced several long-lasting access link failures, and all links experienced many short failures.

The main MIT link has high capacity (over a gigabit per second) and it is BGP multi-homed to three upstream providers. The overall availability of this link *is* much higher than that of the other links—but it exhibited several partial failures in which its packet loss rates were drastically elevated to 80-99%. In contrast, the Cogent link is nearly unused except by the MONET proxy; it is singly homed to one provider; and it is prone to occasional disconnection inside CSAIL. Its failures are nearly binary, exhibiting near-zero or near-total packet loss. The DSL line behaves similarly.

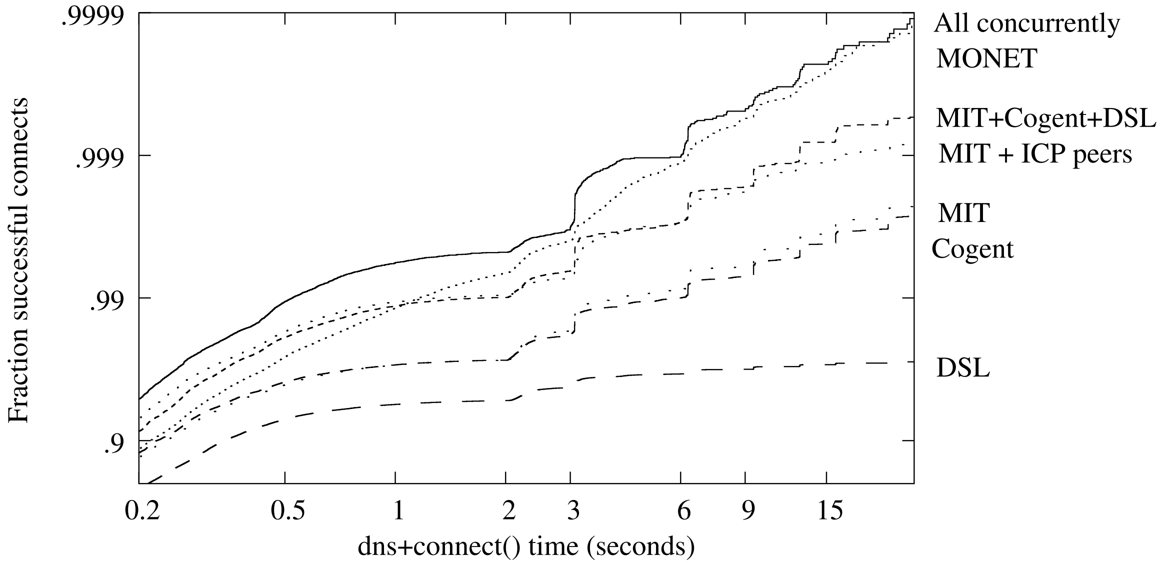


Figure 7-3: MONET performance at CSAIL. MONET with waypoint selection is nearly as effective as using all paths concurrently, but with only 10% overhead. The MIT+Cogent+DSL and MIT+ICP peers lines use the paths concurrently without waypoint selection delays.

7.3 How Well does MONET Work?

The CSAIL proxy has provided uninterrupted service through 20 major network outages.[6] One of our most notable results was the ability of a cheap DSL line to improve the availability of the MIT network connection by over an order of magnitude, which we discuss below.

Much of the following analysis concentrates on the effect that MONET has on long delays and failures. To see the overall effects of the proxy, we examine the cumulative distribution of requests whose DNS resolution and `SYN ACK` were received in a certain amount of time, omitting positively identified server failures. Figure 7-3 shows the availability CDFs for MONET and its constituent links at the CSAIL proxy. This graph and those that follow are in log-scale. The y axis for the graphs starts near the 90th percentile of connections. The top line, "All concurrently," shows the reliability achieved by using all paths concurrently, which the proxy performed to gather trace data. Our simulator picks the order in which MONET's waypoint selection algorithm uses these links, and examines the performance of various combinations of the constituent links and peer proxies. MONET's waypoint selection algorithm rapidly approaches the "All concurrently" line, and outperforms all of the individual links.

MONET has two effects on service availability. First, it reduces exceptional delays. For example, on the Cogent link in Figure 7-3, 2% of the HTTP sessions require more than three seconds to complete DNS resolution and a TCP `connect()`. Combining the MIT link with the Cogent link (which is already one of MIT's upstream ISPs) provides only a small improvement, because packets leaving MIT for many destinations *already* travel via Cogent. When these links are augmented with a DSL line, however, only 1% of sessions fail to connect within three seconds. The improvements in

[6]During the first outage it experienced, we realized that the proxy failed to perform redundant DNS lookups; fixing this shortcoming permitted uninterrupted service during all known outages thereafter. Many of these early outages occurred before our detailed measurement period.

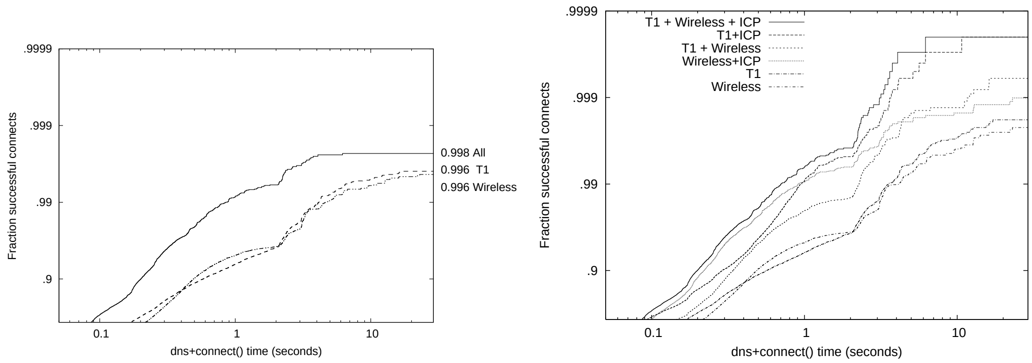


Figure 7-4: The slowest 10% of the cumulative distribution for HTTP sessions through the Mazu proxy, with (top) and without (bottom) server failures.

the one to three second range are primarily due to avoiding transient congestion and brief glitches.

The second effect MONET has is improving availability in the face of more persistent failures. Overall, MONET improves availability due to to non-server failures by at least an order of magnitude (*i.e.*, by at least one "nine"). The MIT+ICP line in Figure 7-3, shows that adding remote proxies to a high-uptime connection can create a robust system by allowing application-level path selection using existing path diversity. A proxy can realize similar availability benefits by augmenting its primary link with a slower and less reliable DSL line (Cog+MIT+DSL). If a site's primary link is already extremely good, the peer proxy solution increases availability without requiring additional network connectivity, and without periodically directing requests via a much slower DSL line. The benefits of using MONET without local link redundancy will, of course, be limited by the overall availability of the local link. For example, MIT+ICP achieves 99.92% availability, a figure similar to that measured by the root name server monitoring, both of which are nearly three times better than the MIT link alone.

Figure 7-4 shows the base link, combined link, and overlay performance for the Mazu proxy. Its performance is similar to that of the CSAIL proxy, though none of its constituent links performed as poorly as did the CSAIL DSL line.

7.3.1 DNS

50.4% of the sessions in the CSAIL proxy trace used a cached DNS response. Figure 7-5 shows the CDF of response times for the remaining 68,250 DNS queries. The fraction of total unavailability caused by DNS can be roughly estimated by multiplying the Y axis scale by two to account for the cached responses. The data shows a strong "hump" at 2 seconds when the DNS servers retransmit unanswered queries. Because the Cog+DSL line at 1.9 seconds is very near to the base Cog link after the 2 second retransmission, it is likely that the benefits from MONET in the 90-99% range arise primarily from avoiding packet loss.

Because DNS servers are frequently replicated, ordinary DNS resolvers are often able to fail-over to a second server and avoid both mid-network failures and DNS server failures. MONET's primary benefits with respect to DNS are that it helps to mask client access link failures and that it reduces the several-seconds delay before the resolver can find a functioning server.

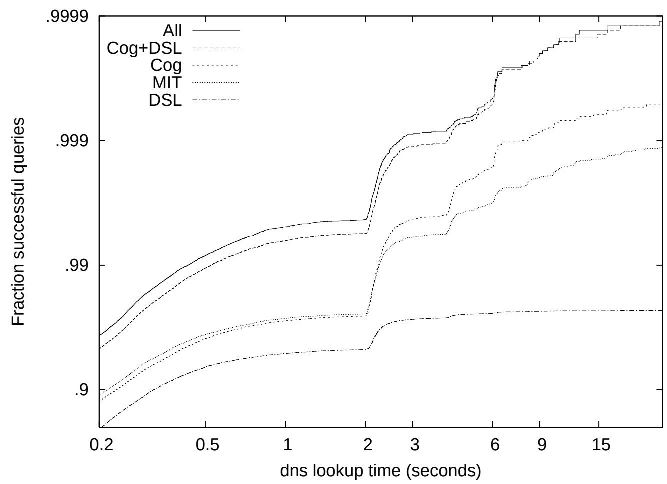


Figure 7-5: The performance of MONET's DNS resolution.

7.3.2 Overhead

MONET's waypoint selection algorithm nearly matches the performance of using all paths, but adds only 10% more SYNs and 5% more ICP+ packets than a client without MONET. The average Web request (retrieving a single object) handled by our proxy required about 18 packets, so this additional overhead comes to about 7% of the total packet load, and is a negligible addition to the byte count. The added packets are small—TCP SYN packets are 40 bytes, and the ICP+ query packets are around 100 bytes depending on the length of the requested URL. The mean Web object downloaded through the MIT proxy was 13 kilobytes. The extra SYN and ICP+ packets added by MONET therefore amount to an extra nine bytes per object.

The simulation of the waypoint selection algorithm chose a random link to use either 5% or 10% of the time. The benefit of the more frequent link measurements was at most a 100-200ms savings in the amount of time it took to find an alternate path when the first path MONET attempted to use had failed—and oftentimes, there was little benefit at all. These latency reductions do not appear to justify the corresponding increase in overhead. A better algorithm might find a better ordering of links for fail-over (*e.g.*, by discovering links whose behavior appears uncorrelated), but because failures are relatively unpredictable, we believe that avoiding transient failures will continue to require attempting several alternate links. Waypoint selection avoids links and peers that fail for longer than a few seconds, but does not improve latency in the shorter ranges.

Because of the relatively high incidence of remote proxy failures, random path selection performs poorly. We also simulated MONET using a static retransmit timer instead of using the `rttvar`-derived value. We found that the static value could be tuned to provide good performance with low overhead, but had to be carefully adjusted for each proxy—and the static value could not adapt to changing conditions over time.

MONET also introduces overhead from additional DNS lookups. As we noted in Section 6.3.4, we believe a MONET with multiple local Internet connections should always perform at least two DNS lookups. Because DNS answers are frequently cached, the overhead is not large—the MONET proxy performed 82,957 DNS lookups to serve 2.1 million objects. The mean packet size for the proxy's DNS queries was 334 bytes. Assuming that the average DNS request requires 1.3 packets in each direction [76], duplicating all DNS requests would have added 34 megabytes of traffic over 1.5 months, or 0.1% of the 27.5 gigabytes served by the proxy. Given that between 15 and 27% of queries to the root nameservers are junk queries [76], it is unlikely that the wide deployment of MONET-like techniques would have a negative impact on the DNS infrastructure, particularly since a shared MONET proxy helps aggregate individual lookups through caching.

Between the DNS, SYN, and ICP+ query/response overhead, MONET using waypoint selection increases the total bytes sent and received by the proxy by 0.15%, and increases the total packet volume by about 8%.

7.3.3 How Well *Could* MONET Do?

The top two lines in Figure 7-3 show the performance of all paths and waypoint selection, respectively. At timescales of one to two seconds, the scheme that uses all paths out-performs MONET, because a transient loss or delay forces MONET to wait a few round-trip times before attempting a second connection. Before this time, MONET approximates the performance of its best constituent link; by 1 second, MONET begins to match the performance of using two links concurrently.

At longer durations of two to three seconds, MONET comes very close to the "all paths" performance, though it does not track it perfectly. A part of the difference between these algorithms arises from mis-predictions by the waypoint algorithm, and a part probably arises from a conservative choice in our waypoint prediction simulator. The simulator takes the "all paths" data as input, knowing, for instance, that a particular connection attempt took three seconds to complete. The simulator conservatively assumes that the same connection attempt one second later would *also* take three seconds to complete, when. This is an accurate assumption if the connection delay were long-lasting, but this connection would likely take only two seconds (since one second had already passed) if the delay was transient.

7.4 Server Failures and Multi-homed Servers

MONET still improves availability, though less dramatically, in the face of server failures. MONET is more effective at improving availability to multi-homed sites than to single-homed sites. The leftmost graph in Figure 7-6 shows the performance of the "all paths" testing with server failures included. This graph includes requests to non-existent servers that could never succeed—the 40% of servers that were *still* unreachable after two weeks—and represents a lower bound on MONET's benefits.

Multi-homed Web sites, in contrast, generally represent a sample of more available, managed, and replicated sites. This category of sites is an imperfect approximation of highly available sites—at least one of the popular multi-homed sites in our trace exhibited recurring server failures—but as the data illustrated in Figure 7-6 showed, they do exhibit generally higher availability than the average site. The multi-homed services we measured typically used combinations of clustering, BGP multi-homing, and low-TTL DNS redirection to direct clients to functioning servers.[7]

23,092 (17%) of the sessions we observed went to Web sites that advertised multiple IP addresses. Web sessions to these multiple-address sites are dominated by Content Delivery Networks (CDNs) and large content providers. For example, Akamai, Speedera, the New York Times, and CNN account for 53% of the sessions (Table 7.4).

[7]This analysis assumes that being able to reach a replica denotes service availability, which may or may not be the case with some caching CDNs.

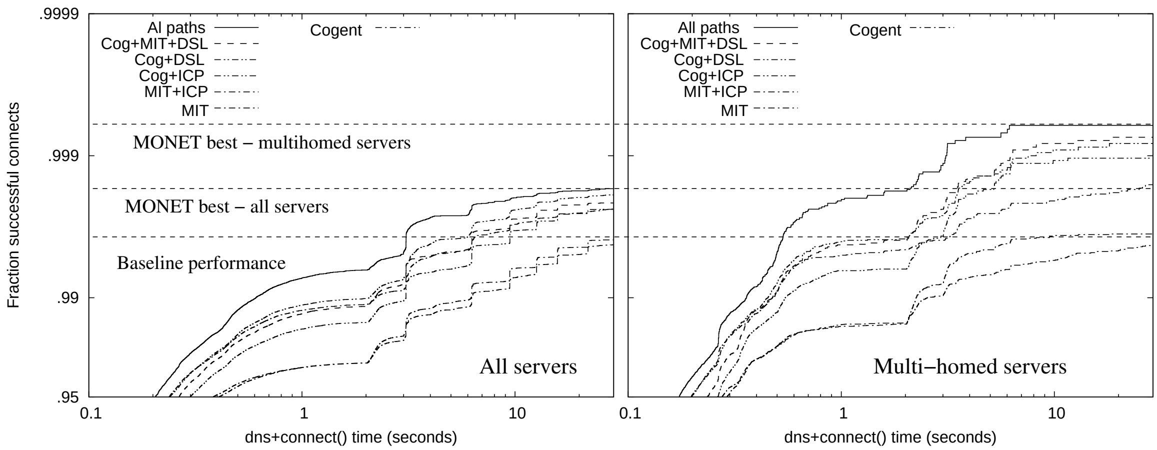


Figure 7-6: HTTP session success, including server failures, through the CSAIL proxy to all servers (left) and only multi-homed servers (right). The success rate of the base links is unchanged, but MONETs effectiveness is enhanced when contacting multi-homed Web sites. The lines appear in the same order as they are listed in the legend. The "baseline performance" line shows the performance of the single constituent links at MIT. The DSL line is not shown; it would appear below the bottom of the graph.

Intriguingly, a single link's access to the multiply announced subset of sites is *not* appreciably more reliable than accesses to all sites (Figure 7-6, right). The MIT connection achieved 99.4% reachability to all sites, and 99.5% to the multi-homed site subset. When augmented with peer proxies, the MIT connection achieved 99.8% availability. Using all local interfaces, MONET achieved 99.92%, and reached 99.93% after just *six seconds* when using both its local interfaces and peer proxies.

MONET's reduction of network failures is more apparent when communicating with multi-homed sites. The improved performance in accessing these sites shows that MONET's use of multiple server addresses is effective, and that MONET's techniques complement CDN-like replication to further improve availability.

The foregoing analysis counted as multi-homed *all* sites that advertised multiple IP addresses. We identified the major providers (Table 7.4 listed the ten most popular) by assigning sites to a content provider by a breadth-first traversal of the graph linking hostnames to IP addresses, creating clusters of hosted sites. We manually identified the content provider for the largest clusters and created regular expressions to match other likely hosts within the same provider. These heuristics identified 1649 distinct IP addresses belonging to 38 different providers. While the analytical method will not wrongly assign a request to a content provider, it is not guaranteed to find *all* of the requests sent to a particular provider.

7.5 Discussion and Limitations

MONET masked numerous major failures at the borders of its host networks and in the wide-area, but its benefits are subject to several limitations, some fundamental and some tied to the current implementation:

Site failures: Two power failures at the CSAIL proxy created failures that MONET could not

Site	# Sessions	Percent
akamai	6167	26
speedera	2582	11
nytimes	1958	8
cnn	1864	8
go	1354	5
google	1074	4
microsoft	813	3
aol	607	2
ebay	556	2
yahoo	523	2

Table 7.5: The 10 most frequently accessed multi-homed providers account for 75% of the accesses. Akamai and Speedera are CDNs; the rest are large content providers.

overcome.[8] Improvements provided by the proxy are bounded by the limitations of its environment, which may represent a more significant obstacle than the network in some deployments.

Probes do not always determine success: A failed Internet2 router near **MIT**'s border began dropping most packets larger than 400 bytes. Because MONET uses small ($\sim$ 60 byte) SYN packets to probe paths, the proxy was ineffective against this bizarre failure. While MONET's probes are more "end-to-end" than the checks provided by other systems, there is still a set of failures that could be specifically crafted to defeat a MONET-like system. A higher-level progress check that monitored whether or not data was still flowing on the HTTP connection could help detect additional failures, re-issuing the HTTP request if necessary. Such solutions, however, must take care to avoid undesirable side-effects such as re-issuing a credit card purchase. As mentioned in the previous chapter, MONET does not handle mid-stream failures; higher level progress checks apply also to mid-stream failures.

Software failures. Several Web sites could never be reached directly, but could always be contacted through a remote proxy. These sites sent invalid DNS responses that were accepted by the BIND 8 name server running on the remote proxies, but that were discarded by the BIND 9 name-server on the multi-homed proxies. While these anomalies were rare, affecting only two of the Web sites accessed through the MIT proxy, they show some benefits from having diversity in software implementations in addition to diversity in physical and network paths.

Download times. Initial connection latency is a critical factor for interactive Web sessions. Total download time, however, is more important for large transfers. Earlier studies suggest that connection latency is effective in server selection [43], but there is no guarantee that a successful connection indicates a low-loss path. We briefly tested whether this held true for the MONET proxies. A client on the CSAIL proxy fetched one of 12,181 URLs through two randomly chosen paths at a time to compare their download times, repeating this process 240,000 times over 36 days. The SYN response time correctly predicted the full HTTP transfer 83.5% of the time. The objects fetched were a random sample from the static objects downloaded by users of our proxy.

Trust. MONET proxies extend trust to their peer proxies, and users must trust their chosen MONET proxy. Many of the trust issues raised by MONET, and the potential solutions to those issues, are

[8] It is likely that most of the clients *also* had power failures, but clients accessing the proxy from other buildings may have been affected.

similar to those discussed for RON in Chapter 5. Because it already understands more application semantics than does RON, MONET is perhaps in a better position to solve these problems. For example, MONET could use the experimental secure HTTP [70] protocol, which supports caching proxies, the proxies and their clients could verify the content signatures and checksums on downloaded content. Most likely, however, administrators of MONET proxies would choose peers in the same way that administrators of Web caches choose proxies—by only peering with organizations they trust.

7.6 Summary

This chapter presented the evaluation of MONET, a Web proxy system to improve the end-to-end client-perceived availability of accesses to Web sites. A single-proxy multi-homed MONET system was first deployed in April 2003. The version of the system described in this dissertation using multiple proxies has been operational for over one year, and has been in daily use by a user community of at least fifty users.

Our experimental analysis of traces from a real-world MONET deployment show that MONET corrected nearly *all* observed failures where the server (or the server access network) itself had not failed. MONET's simple waypoint selection algorithm performs almost as well as an "omniscient" scheme that sends requests on all available interfaces. In practice, for a modest overhead of 0.1% more bytes and 8% more packets, we find that between 60% and 94% of all observed failures can be eliminated on the different measured physical links, and the "number of nines" of non-server-failed availability can be improved by one to two nines.

Our experience with MONET suggests that Web access availability can be improved by an order of magnitude or more using an inexpensive and relatively low speed link (*e.g.*, a DSL link), or using a few other peer proxies. With the techniques incorporated in MONET, the cost of high Web access availability is low, an encouraging possibility.

We believe that MONET's end-to-end approach addresses all the reasons for service unavailability and our experimental results show that these failures are maskable, except for server failures themselves. With MONET in place, the main remaining barrier to "five nines" or better availability is failure resilience at the server and the server's network.

They have computers, and they may have other weapons of mass destruction.

- Janet Reno

Chapter 8

Preventing Denial-of-Service Attacks using Overlay Networks

Previous chapters presented mechanisms that help mask accidental failures and those occurring because of misconfiguration or bugs, but they did little to protect a service from a final frequent cause of failures: a denial-of-service (DoS) attack specifically targeting the hosts in question. This chapter presents Mayday, an overlay-based distributed filtering system that proactively defends against flooding-based DoS attacks.

8.1 Introduction

Denial of service attacks are potentially devastating to the victim and require little technical sophistication or risk exposure on the part of the attacker. These attacks typically attempt to flood a target with traffic to waste network bandwidth or server resources. To obtain the network bandwidth necessary to attack well-connected Internet services, attackers often launch Distributed DoS (DDoS) attacks, in which tens to thousands of hosts concurrently direct traffic at a target (Figure 8-1). The frequency of these attacks is startling—one analysis of attack “backscatter” suggests that hundreds of these attacks take place each day [102]. DDoS attacks no longer require a high degree of sophistication. So-called “rootkits” are available in binary form for a variety of platforms, and can be deployed using the latest off-the-shelf exploits. Even worm programs have been used to launch DDoS attacks [21].

Chapter 2 identified four major approaches to preventing and detecting DoS attacks:

1. **Ingress filtering** can prevent attackers from forging the IP source address of the packets they send [52].
2. **Traceback** facilities attempt to permit administrators to identify the source of attacks [149, 38, 141].
3. **DoS detection** schemes attempt to identify DoS attacks and filter them locally [13, 15, 95].
4. **Pushback** schemes define methods to push packet filtering closer to the attack source [112, 72] to further reduce the harm done by the attack.

Most of these measures require widespread adoption to be successful. Unfortunately, even the simplest of these techniques, ingress filtering, is not globally deployed despite years of advocacy. While there is some interim benefit from the incremental deployment of earlier measures, they either

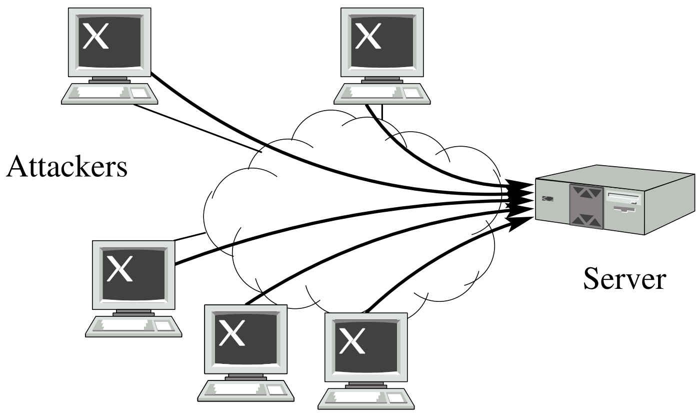


Figure 8-1: A Distributed Denial-of-Service (DDoS) attack. Hundreds to thousands of hosts concurrently direct traffic at a target, overwhelming its network and/or servers.

lack the deployment incentive of a solution that provides immediate relief to the deployer, or tend to be deployed inside the customer network, in which case a DoS attack can still clog the customer's Internet access link.

An ideal DDoS prevention system stops attacks as close to their source as possible. Unfortunately, the targets of attacks have the most incentive to deploy solutions, and deployment is easiest inside the target's own network. Intrusive systems that perform rate-limiting or that require router modifications hold promise. While the customer that is a victim of DoS attacks has the incentive to deploy such solutions, most Internet Service Providers (ISPs) are unwilling or unable to deploy these solutions in the places where they would be most effective—in their core or at their borders with other ISPs.

This chapter presents a set of solutions that are resource-intensive to deploy, because they require overlay nodes, but that are easily understood and implemented by ISPs using conventional routers. Mayday generalizes the Secure Overlay Services (SOS) approach [81]. Mayday uses a distributed set of *overlay nodes* that are trusted (or semi-trusted) to distinguish legitimate traffic from attack traffic. To protect a server from DDoS traffic, Mayday prevents Internet hosts from communicating directly with the server by imposing a router-based, network-layer *filter ring* around the server; only the overlay nodes are allowed to communicate directly with the server. Clients instead communicate with the overlay nodes, which verify that the client is permitted to use the service. These overlay nodes then use an easily implemented *lightweight authenticator*, such as sending the traffic to the correct TCP port on the server, that permits their packets to to pass through the filter ring. Within this framework, the earlier SOS system represents a particular choice of authenticator and overlay routing, using distributed hash table lookups to route between overlay nodes, and using the source address of the overlay node as the authenticator. This chapter explores how different organizations of the overlay nodes operate under various threat models, and present several lightweight authenticators that provide improved levels of defense over source address authentication.

Finally, we define several threat models with which we evaluate proactive DDoS protection. Within these threat models, we present several attacks, including a novel scanning method we call next-hop scanning, that are effective against SOS, certain variants of Mayday, and against conventional router-based filtering of DDoS attacks.

8.2 Design

The design of Mayday evolved from one question: using existing IP router functionality, what kind of system can protect a server from DDoS attacks while ensuring that legitimate clients can still access the service? We restricted our answer to solutions that use only routers with commonly available packet filtering abilities, and augment these routers with additional hosts that are not on the packet forwarding path. Mayday aims to protect against realistic attackers who control tens or thousands of machines, not malicious network operators or owners (*e.g.*, governments). Before exploring the design of Mayday, we define these attackers and the capabilities they possess.

8.2.1 Attacker Capabilities

We focus exclusively on *flooding* attacks, and not on attacks that could crash servers with incorrect data. (Using the overlay nodes to verify that clients are sending appropriately constructed data could prevent additional attacks, but that remains a project for future work.) The simplest flooding attacks, which may be effective if launched from a well-connected site, require only a single command such as `ping`. Many more sophisticated attacks come pre-packaged with installation scripts and detailed instructions, and can often be used by people who may not even know how to program. The greatest threat to many Internet services comes from these relatively simple attacks because of their ease of use and ready accessibility; hence, we focus our attention on them.

We assume that attackers can send a large amount of data in arbitrary formats from forged source addresses of their choice. Ingress filtering may reduce the number of hosts with this capability, but is unlikely to eliminate all of them. Certain attackers may have more resources available, may be able to sniff traffic at points in the network, and may even be able to compromise overlay nodes. We consider the following classes of attackers:

- The **client eavesdropper** can view the traffic going to and from one or more clients, but cannot see traffic reaching arbitrary, distant hosts.
- The **legitimate client attacker** is authorized to use the service, or is in control of an authorized client. In some cases, such as protecting a large, popular Internet service, most attackers will be considered legitimate clients.
- The **random eavesdropper** can monitor the traffic going to one or more Internet hosts, but cannot monitor arbitrary hosts.
- The **targeted eavesdropper** can view the traffic going to and from any particular host, but cannot monitor all (or a large set of) nodes at once. Changing the monitored node requires non-negligible time.
- The **random compromise attacker** can compromise one or more randomly chosen hosts; in particular, this class can compromise one or more overlay nodes, because our solution relies on an overlay network.
- The **targeted compromise attacker** can select a particular overlay node, or a series of them, and obtain full control of the node.

We ignore certain attackers. For instance, an attacker capable of watching all traffic in the network or compromising all nodes concurrently is too powerful for our approach to resist. The difference

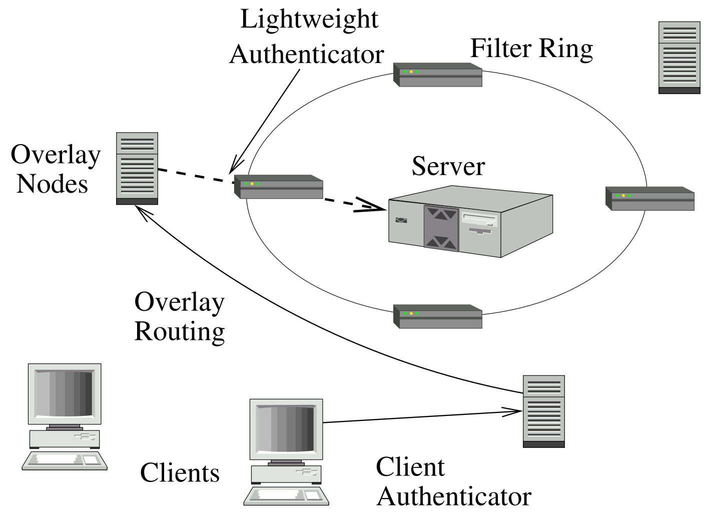


Figure 8-2: The Mayday architecture. Clients communicate with overlay nodes using an application-defined client authenticator. Overlay nodes authenticate the clients and perform protocol verification, and then relay requests through the filter ring using a lightweight authenticator. The server handles requests once they pass through the network-layer filter ring.

between these global attackers and the targeted eavesdropper or compromiser is one of time and effort. Given sufficient time, the targeted compromise attacker may be able to control all nodes, but during this time the service provider may counteract the offense by reloading or patching compromised nodes.

8.2.2 Mayday Architecture

For purposes of this discussion, the *server* is a centralized resource that provides some service. *Clients* are authorized to use the service, but are not trusted to communicate directly with the server because clients are more numerous and more prone to compromise. An example of a server would be a Web site or email provider. Mayday is an application-specific solution—it depends upon being able to authenticate clients in some way—but it is not restricted to a *particular* application. The application could be as generic as "a service that accepts TCP connections," provided there was some out of band mechanism to authenticate clients.

Figure 8-2 shows the Mayday architecture. Clients communicate directly with an overlay node, the *ingress node*, not with the server. Some of the overlay nodes, the *egress nodes*, can send packets to the server through a *filter ring* of routers that prevent most nodes on the Internet from directly reaching the server. If the ingress node is not also an egress node, the overlay nodes must forward the request from the ingress node to the egress node, possibly through a series of intermediate overlay nodes.

The Mayday architecture assumes that some entity—perhaps the server's ISP—has routers around the server that provide its Internet connectivity, and is willing to perform some filtering at those routers on behalf of the server. Such a filter ring implemented only at the router closest to the server would provide little attack protection, because the traffic would consume the limited bandwidth close to the server. Pushing the filter ring too far from the server increases the chance that nodes within the filter ring can be compromised. Instead, the ring is best implemented near the "core-edge" boundary, where all traffic to the server passes through at least one filtering router, but before the network bottlenecks become vulnerable to attack. To provide effective protection against large attacks, this filtering must be lightweight enough to be implemented in high-speed Internet routers.

High speed routers can perform forwarding lookups and a limited number of packet classification and filtering operations at line speed. Clients, however, may be numerous, and the set of clients may change dynamically. Verifying the legitimacy of individual clients or users may involve database lookups or other heavyweight mechanisms. Access lists in core routers are updated via router configuration changes, so network operators are not likely to favor a solution that requires frequent updates. Creating an access list of authorized clients is unlikely to be practical because of the sheer size and rapidly changing nature of such a list, and the lack of an unambiguous notion of "identity" within the Internet architecture. As a result, a filtering solution requires keys that change less often. These filter keys are analogous to *capabilities*, but because of their lack of cryptographic strength, we term them *lightweight authenticators.*

The requirement for a fast router implementation rules out certain design choices. One obvious mechanism would be for clients to use IPSec [161] to authenticate themselves to a router in the filter ring, at which point the router would pass the client's traffic through. Such a mechanism could shield a server from some DoS attacks, but it requires that routers participate in client authentication.

Using this architecture, a designer can make several choices to trade off security, performance, and ease of deployment. First, the designer can first pick one of several overlay routing methods: more secure overlay routing techniques reduce the impact of compromised overlay nodes, but increase request latency. Second, the designer can pick one of several lightweight authenticators, such as source address or UDP/TCP port number. The choice of authenticator affects both security and the overlay routing techniques that can be used. The security and performance of the resulting system depend on the *combination* of authenticator and overlay routing. We discuss the properties of these combinations in Section 8.2.6 after describing the individual mechanisms.

8.2.3 Client Authentication

Clients must authenticate themselves to the overlay before they are allowed to access the server. The nature of the client authentication depends on the service being protected; the system designer must customize this component of a Mayday-based system, since the authentication is inextricably linked to the specific application being protected. If Mayday is used to protect a small, private service, clients could be authenticated using strong cryptographic verification. In contrast, if Mayday is protecting a large, public service that provides service to millions of weakly-authenticated users, client authentication may be only a database verification of the user's password—or even nothing at all, in which case Mayday would merely enforce per-client rate-limiting, or prevent "known bad" computers from contacting the service.

8.2.4 Lightweight Authenticators

Mayday uses lightweight authentication tokens to validate communication between the overlay node(s) and the server. Such tokens must be supported with low overhead by commodity routers. Modern routers can filter on a variety of elements in the packet header, such as source and destination address, TCP or UDP port number, and so on. Mayday thus uses the IP header fields in packets between the egress node and the server as authenticators. Each of these fields has its own strengths and weaknesses:

- **Egress node source address**: Source filtering is well understood by network operators, and gains effectiveness when other providers deploy IP spoofing prevention. If attackers can spoof IP addresses, then only a few overlay nodes can be permitted to directly contact the server because the IP address of the authorized nodes must remain secret.

- **Server destination port**: The TCP or UDP destination port is easily used, and provides 16 bits of authentication key-space (2^{16} "keys"). Using the destination port allows multiple authorized sources to communicate with the server.

- **Egress node source port**: The source port can be used in the same fashion as the destination port, but doing so with an unmodified server limits the total number of concurrent connections to the server: TCP identifies a connection by the (source address, source port, destination address, destination port) tuple. Ordinarily, multiple connections from one node to another use different source ports. If the system design permits changing the protocol between overlay nodes and the server (perhaps by encapsulating the real packets in a stream of TCP-like packets), the source port can be easily used to provide an extra 16 bits of authenticator.

- **Server destination address**: If the server has a variety of addresses that it can use, the destination address can be used as an authentication token. For example, if a server is allocated the netblock 192.168.0.0/24, its ISP would announce this entire block via BGP to other routers in the Internet. Internally, the server would only announce a single IP address to its ISP, and send a null route for the remaining addresses. Thus, a packet to the correct IP would go to the server, but packets to the other IP addresses would be dropped at the border routers. The advantage of this mechanism is that it requires no active support from the ISP to change filters, and uses the fast routing mechanisms in routers, instead of possibly slower filtering mechanisms.[1] Because it uses standard routing mechanisms, updates could be pushed out much more rapidly than router reconfigurations for filter changes. We term this effect *agility*, and discuss later how it can provide freshness to authenticators. The disadvantage is that it wastes address space (a problem solved by IPv6, though IPv6 has its own deployment delays). Destination address filtering is unique in that it can be changed relatively quickly by routing updates, even in a large network.

- **Other header fields**: Some routers can filter on attributes like protocol, packet size, and fragment offset. Each of these fields can be manipulated by the egress node to act as a lightweight authenticator, but they require IP-layer changes to set. While they could be useful for providing additional bits of key space, they are less usable than port or address filtering, except to provide some security through obscurity.

- **User-defined fields**: Firewalls can filter on user-defined fields inside packets. This approach provides a large key-space and source address flexibility, but few high-speed routers support this feature.

Authentication tokens can be combined. Using both source address and port verification provides a stronger authenticator than source address alone, making it more difficult to successfully execute some of the attacks we discuss in Section 8.3.

[1] An interesting manual use of destination filtering occurred during the Code Red worm in 2001. The worm was designed to flood the IP address of `www.whitehouse.gov`, but used a hard-coded target IP address instead of performing a DNS lookup. The site's administrators changed the IP address and filtered the old address to successfully protect themselves from the attack.

8.2.5 Overlay Routing

The choice of overlay routing can reduce the number of overlay nodes that have direct access to the server, thus providing increased security. These choices can range from direct routing, in which every overlay node can directly access the server (*i.e.*, be an egress node), to Mixnet-style routing ("onion routing"), in which the other overlay nodes do not know the identity of the egress node [57].

The choice of lightweight authenticator affects which overlay routing techniques can be used. For instance, using source address authentication with a system that sends traffic directly from the ingress node to the server is extremely weak, because an attacker already knows the IP addresses of the overlay nodes, and any of those addresses can pass the filter.

- **Direct ingress**: By sending traffic directly from the ingress node to the server, the system imposes low overhead. In fact, if the ingress node is chosen to maximize the performance between the client and server, direct ingress routing might actually improve performance (*e.g.*, in the same way that MONET and RON can do) instead of being an impediment. This form of routing requires that that all overlay nodes possess the lightweight authenticator.

- **Singly-indirect routing**: The ingress node passes the message directly to the egress node, which sends the message to the server. The number of egress nodes is much smaller than the number of overlay nodes. All overlay nodes know the identity of the egress node or nodes.

- **Doubly-indirect routing**: Ingress nodes send all requests to one or more overlay nodes, which then pass the traffic to the egress node. Only a subset of overlay nodes knows the identity of the egress node. SOS uses this scheme.

- **Random routing**: The message is propagated randomly through the overlay until it reaches a node that knows the lightweight authenticator. If there are E egress nodes, this routing scheme adds $O(\frac{N}{E})$ additional overlay hops, but provides better compromise containment because no overlay node actually knows the identity of the egress node. This routing is largely inferior to mix routing, discussed next.

- **Mix Routing**: Based on Mixnets [26] and the Tarzan [57] anonymous overlay system. A small set of egress nodes configure encrypted forwarding tunnels through the other overlay nodes in a manner such that each node knows only the next hop to which it should forward packets, not the ultimate destination of the traffic. At the extreme end of this style, *cover traffic*—additional, fake traffic between overlay nodes—can be added to make it difficult to determine where traffic is actually originating and going. This difficulty provides protection against even the targeted eavesdropper and compromise attacker, but it requires many overlay hops and potentially expensive cover traffic.

8.2.6 Choice of Authenticator and Routing

A major design choice for a Mayday-based system is which combination of overlay routing and authenticator to implement. An obvious first concern is practicality: If an ISP is only able to provide a certain type of filtering in the filter ring, the designer's choices are limited. There are three axes on which to evaluate the remaining choices: security, performance, and agility. Many combinations of authenticator and routing fall into a few "best" equivalence classes that trade off security or performance; the remaining choices provide less security with the same performance, or vice versa. The choices are listed in order of decreasing performance and increasing security; system designers can choose the option that provides the best security against the classes of attacks the system is likely to observe and still that falls within the range of acceptable overhead.

High performance: Proximity routing provides the best performance, but a random eavesdropper can immediately identify the lightweight authenticator by observing the traffic from any overlay node. Proximity routing works with any authenticator *except* source address, since the address of the overlay nodes is known. Blind DoS attacks against the system are difficult, since all nodes can act as ingress and egress nodes. Singly-indirect routing with source address provides equivalent protection with inferior performance.

Eavesdropping Resistance, Moderate Performance: Singly-indirect routing, when used with any authenticator other than source address, provides resistance to the random eavesdropper and random compromise attack, because only a small number of nodes possess the authentication key.

SOS: The SOS method uses doubly-indirect routing with source address authentication. In the SOS framework, packets enter via an "access node," are routed via a Chord overlay [156] to a "beacon" node, and are sent from the "beacon" node to the "servlet" node. The servlet passes packets to the server. This method provides equivalent security to the singly-indirect scheme above, but imposes at least one additional overlay hop.

Mayday-Mixnet: By using Mix-style routing with cover traffic, a service provider can provide some resistance against the targeted compromise attacker. For example, using three hops within the Tarzan anonymizing overlay routing system [57], an attacker must compromise 24 nodes to reach the egress node. By using agile destination-address based authentication, the service provider gains resistance to adaptive attacks. By combining the agile authenticator with port number authentication, the system increases its key space, while retaining the ability to recover from egress node failures. Alternately, source address authentication would slow this recovery, but it practically reduces the number of attack nodes that can successfully be used since many Internet hosts are filtered.

8.2.7 Agility

A Mayday system that uses destination address authentication (perhaps in combination with other authenticators) provides an agile method for authenticator changes. Because routing updates, not manual configuration changes, are used to change the lightweight authenticator, it is feasible to update the authentication token often. This agility can be used to resist adaptive attacks by changing the authentication token before the attack has sufficiently narrowed in on the token. Destination address authentication can provide this benefit in concert with other authenticators (such as port number) to provide an agile scheme with a large number of authenticators.

8.2.8 Switchable Protection

By using destination address-based filtering, we can provide *switchable* DoS protection: When no attack is present, clients may directly access the service. When an attack commences, the system can quickly and automatically switch to a more secure mode, assuming that some channel exists to notify the nodes and routers of the change. Such switching allows us to use Mayday as both a reactive and a proactive solution.

The protected service is given two or more IP addresses. IP address A is the "normal" mode address, and the other addresses are the "secure" mode addresses. When an attack commences, the service sends a routing update (in the same manner as destination-address based authentication) to change to one of the secure addresses. The service simultaneously informs the egress nodes of its new address.

The limiting step in reactive DoS protection is configuring access lists at many routers concurrently. To speed this step, the ISP configures two sets of access lists *in advance*. The first list permits access from all clients (or all overlay nodes, for proximity routing) to the normal mode address A. The second list restricts access to those packets containing a lightweight authenticator, and applies to the secure mode addresses. The server can quickly switch modes by sending a routing update.

When used to switch between a mode in which clients directly access the server and a more protected mode, such an update may require that clients contact a different IP address, which could be handled by a DNS update with an accompanying delay before all clients moved to the new address. This change is not a problem when the normal mode relays packets packets directly from the ingress node to the server, and the secure involves more stringent routing and filtering. In this case, the addresses to which clients connect do not change, and client connections need not be interrupted at the commencement of a DDoS attack.

8.2.9 Changing Authenticators or Overlay Nodes

Changing the authentication key or the overlay nodes through which traffic passes could interrupt client traffic to the server, particularly if that traffic is carried via a TCP connection from a standard application. Fortunately, the communication between the ingress node and the server is under the control of the system designer. When a connection between the ingress node and server is interrupted by address or port changes, the designer can use mechanisms such as TCP Migrate [148] or other end-to-end mobility solutions to transparently reconnect the session. Using these mechanisms between ingress node and server would not require client changes.

8.3 Attacks and Defenses

The ability of an overlay-based architecture to resist simple flooding attacks was explored in the SOS study [81]. For various simple attack models, a sufficiently large number of overlay nodes can resist surprisingly strong attacks targeted against the server or against individual overlay nodes. In this section, we examine a more sophisticated set of attacks than the simple flooding explored in earlier work.

We view these attacks within the environment shown in Figure 8-3. We first present several probing attacks that can quickly determine a valid lightweight authenticator to defeat the DDoS protection. We then examine more sophisticated flooding attacks, and examine the effects of eavesdropping and compromise attacks. We assume that attackers may discover the ISP network topology by various means [151, 11, 60].

8.3.1 Probing

Several lightweight authenticators, such as destination port or destination address, allow arbitrary hosts to communicate directly with the target. While this provides flexibility and higher performance, it can be vulnerable to simple port-scanning attacks (Figure 8-4), particularly when the key-space provided by the lightweight authenticator is small. If the target machine replies to any packet with the correct lightweight authenticator, it is a trivial matter to scan, say, the 2^{16} possible destination ports, or the 2^8 addresses in a /24 netblock. On a 100 Mbits/s Ethernet, a full port scan takes about 11 seconds. To prevent these attacks from succeeding, a *secondary key*, drawn from

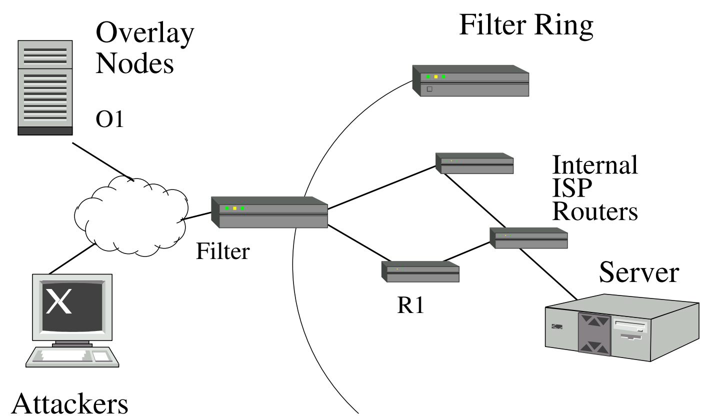


Figure 8-3: The framework for considering attacks. Overlay nodes and clients are both outside the filter ring. Inside the filter ring may be more ISP routers, which eventually connect to the server.

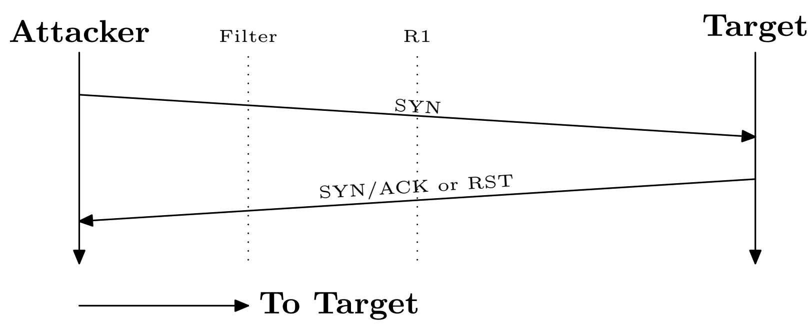


Figure 8-4: A simple port-scan. The attacker sends packets directly to the target to determine which ports are open. If the communication between the attacker and the target is not filtered, the SYN packet will elicit a SYN/ACK response if a service is listening to the port, or a RST packet if the port is closed. If the server is filtered, there will be either no response, or an ICMP "administratively prohibited" message from the filtering device.

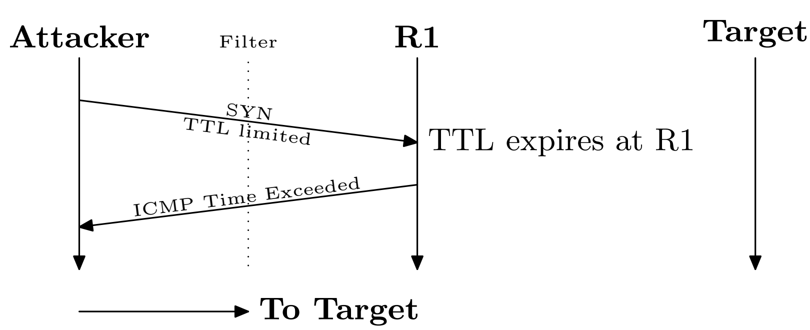


Figure 8-5: Firewalking. The attacker uses a traceroute-like mechanism to probe the ports that are allowed through the filter ring, without needing replies from the actual target.

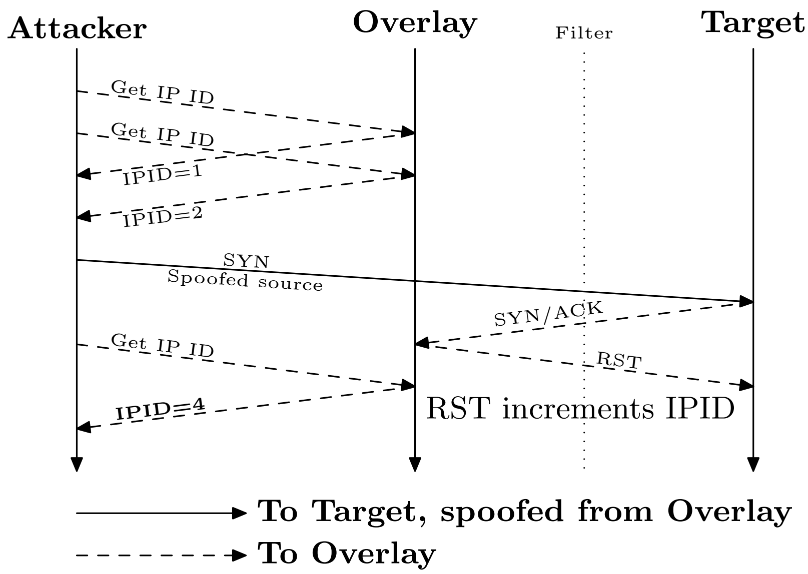


Figure 8-6: Idlescan indirect probing. The attacker spoofs a TCP SYN packet to the target. If the packet gets through the filter, the target replies with a TCP ACK to the overlay node. The overlay generates a RST because the connection does not exist. The attacker notices the IP ID increment at the overlay node when it sends the RST to determine whether the packet got through the filter ring.

a large key-space, is needed. While packets with a valid lightweight authenticator will go through the firewall, the server will respond to only packets with the proper secondary key. This approach requires considerable care to prevent the server from responding to *any* queries that do not possess the correct secondary key—sending ICMP port unreachable messages or TCP RST packets could all betray the lightweight authenticator. The secondary key could be the addresses of the valid correspondent hosts, a key inside the packets, or heavier weight mechanisms such as IPSec.

The application of the secondary key is complicated by techniques such as *firewalking* [59] that use time-to-live (TTL) tricks to determine what ports a firewall allows through, without requiring that the target host reply to such messages. Figure 8-5 shows an example of firewalking in which the attacker sends a packet whose TTL reaches zero before it reaches the target. An intermediate router may respond with an ICMP "time exceeded" message, revealing to the attacker that their packet made it through the filter ring. Firewalking can be defeated by blocking ICMP TTL exceeded messages at the filter ring, but this breaks utilities like `traceroute` that rely on these messages.

If the filter ring authenticates based on the egress node's source address, attackers can use indirect probing mechanisms to determine the set of nodes that can reach the server. Tools such as Nmap [166] and Hping [139] can use IP ID increment scanning (or *idlescanning*) [138] to scan a host indirectly via a third party. Figure 8-6 shows an idlescan in which the attacker watches to see whether the overlay node has received TCP ACK packets from the target. If the overlay node receives a TCP ACK for a connection it did not initiate, it will reply with a TCP RST packet. Transmitting this RST causes the overlay node to increment its IP ID. The attacker monitors the IP ID by sending a constant stream of packets to the overlay node, and thus can determine when the attack traffic has passed the filter ring. This technique is limited to TCP, and can be deterred by modifying the overlay nodes to carefully randomize the contents of the IP ID field in packets they emit. This technique also depends on low or predictable traffic volumes on the overlay nodes.

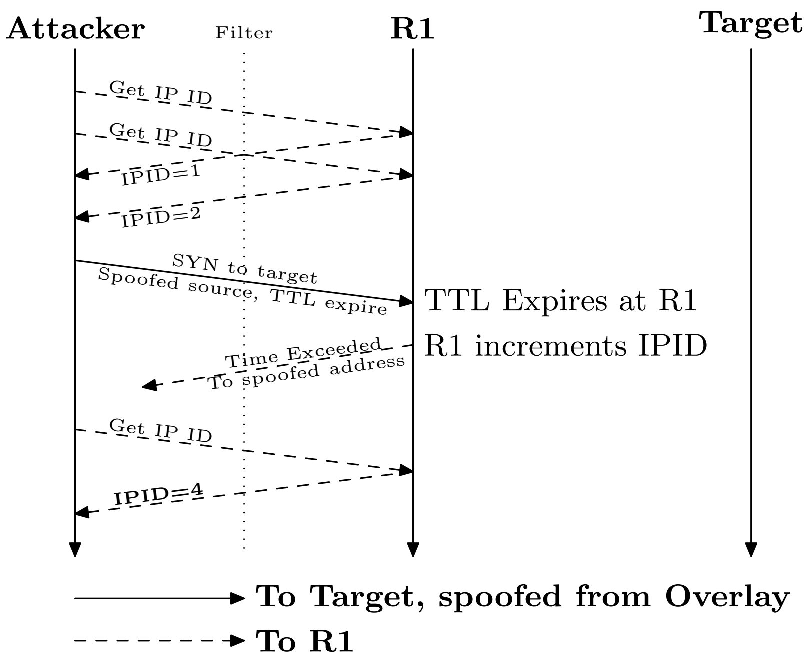


Figure 8-7: Next-hop scan. This attack combines the idlescan and firewalking to determine from an interior router whether packets got through the firewall.

A variant on idlescanning that we call *next-hop* scanning can use other routers behind the filter ring to determine whether spoofed packets are getting through. Figure 8-7 shows next-hop scanning. Like firewalking, next-hop scanning sends a TTL-limited probe towards the target, which expires at some router $R1$ inside the filter ring. $R1$ generates an ICMP time exceeded message. Instead of directly receiving this message (if it's filtered or the source address was spoofed), the attacker indirectly observes the generation of the ICMP reply by the IP ID increment at $R1$.

Figure 8-8 shows a next-hop scan in action. Between sequence 93 and 94, the attacker machine sent 40 TTL-limited probes at the target, causing a large jump in $R1$'s IP ID. This trace was taken using `hping` against a production Cisco router inside a large ISP; the IP addresses have been obscured.

If the overlay nodes run vulnerable software, they may be open to other attacks that permit an

Source	Seq #	**IP ID change**	rtt
192.168.3.1	seq=91	id=+19	76.5 ms
192.168.3.1	seq=92	id=+16	233.4 ms
192.168.3.1	seq=93	id=+14	259.6 ms
192.168.3.1	seq=94	**id=+61**	76.2 ms
192.168.3.1	seq=95	id=+12	76.6 ms
192.168.3.1	seq=96	id=+10	75.5 ms

Figure 8-8: Next-hop scan showing IP ID increase at the router after the filter. After packet 93, the attacker sent a burst of packets that went through the filter. This scan method can be used to determine whether spoofed packets are permitted to go towards a target.

attacker to easily scan the lightweight authenticator key space. Unlike application or specific host vulnerabilities, the IP ID scans are applicable to a wide range of host and router operating systems. They are difficult to defeat with an overlay, because they require either upgrades to the interior routers to prevent next-hop scanning from working, or require extensive firewalling techniques that may not be practical on high-speed routers that serve numerous customers.

8.3.2 Timing Attacks

In an N-indirect Mayday network in which only certain overlay nodes are allowed to pass traffic to the server, a malicious client may be able to determine the identity of these nodes by timing analysis. Requests sent to an egress overlay node will often process more quickly than requests that must bounce through an extended series of intermediate nodes; for example, overlay traversal in SOS adds up to a factor of 10 increase in latency. This attack could allow an attacker to determine the identity of the egress node even in a randomly routed overlay. This attack can be mitigated by using multiple egress nodes and always relaying requests to a different egress node.

8.3.3 Adaptive flooding

This attack is one step up from blindly flooding the target with spoofed IPs. If the attacker can measure the response time of the target by collusion with a legitimate client or passively monitoring clients, he can launch a more effective attack than pure flooding. The success of a DoS attack is not binary—intermediate levels of attack may simply slow down or otherwise impair the service provided.

Consider a lightweight authenticator whose key-space has N possible values (*e.g.*, the 2^{16} destination TCP ports, or the number of possible egress overlay nodes). One value allows traffic to reach the target and consume limited resources. The target has a certain unused capacity, its *reserve*, R. The attacker can generate a certain amount of traffic, T. If $T > R$, the attacker uses up the target's resources, and the target's service degrades.

In most DDoS attacks, $T >> R$: the attacker's force is overwhelmingly large. In this case, the attacker can attack with *multiple* authenticators concurrently to guess which one is being used. If the attacker uses $\frac{N}{2}$ different authenticators, then 50% of the time, one of those authenticators is valid, and $\frac{2T}{N}$ traffic will reach the target. If the service slows down, the attacker knows that the authenticator was in the tested half of the key-space. By recursively eliminating half of the remaining nodes in a binary-search like progression, the attacker can identify the authenticator in $O(\log N)$ attack rounds. After this, the full ferocity of the attack will penetrate the filter ring. Even intermediate rounds are likely to damage the target.

Mayday slows this attack using a large key-space and by updating its keys if the attack makes too much progress. When the attack power is sufficiently diluted (*i.e.*, $\frac{2T}{N} < R$), the attack must first linearly probe small batches of the key-space before identifying a range into which the binary search can proceed. Because this attack takes multiple rounds, key agility is effective at reducing the threat by allowing the system to change the key, ensuring that it remains fresh in the face of an attack.

8.3.4 Request Flood Attacks

Without careful attention to design, the overlay itself could be used by a malicious client to attack the target. As an example, consider an overlay system designed to protect a Web site. A naive implementation of such an overlay might use a standard Web proxy at the overlay nodes. A legitimate client attacker could turn this system to his advantage by simultaneously requesting content from all of the overlay nodes and reading the responses as slowly as possible. A naive Web proxy on the nodes would request the content as quickly as possible from the server, turning the overlay system itself into an attack tool.

These attacks are fairly easy to trace because they involve legitimate packets and can be traced to particular users, and they apply more to large, open systems than to closed systems with trusted clients. However, they point out the need for caution when designing a system to improve performance or security, to ensure that the resulting nodes cannot themselves be used to launder or magnify a DoS attack. It may be possible to use a "reverse Turing test" (one that is easy for a human to solve, but difficult for a computer) to staunch automated attacks that use legitimate accounts [78].

8.3.5 Compromised Overlay Nodes

Controlling an overlay node allows an attacker not only the ability to see source/destination addresses, but to see the actual contents of the information flowing across the network. An attacker knows anything a compromised node knows.

Furthermore, the attacker can now launch internal attacks against the overlay itself. For example, the SOS system uses Chord [156] to perform routing lookups. The Chord system, and similar distributed hash tables, are themselves subject to a variety of attacks [145]. Any other component of the lookup system is similarly a potential source of cascaded compromise when an overlay node is compromised. This observation argues for keeping the overlay routing as simple as possible, unless the complexity results in significant security gains.

Proximity routing and singly-indirect routing can immediately be subverted when nodes are compromised. Doubly-indirect routing provides a degree of resilience to an attacker who compromises a node in a non-repeatable fashion (physical access, local misconfiguration, etc.). Random routing and mix routing can provide increased protection against compromise, but even these techniques will only delay an attacker who exploits a common flaw on the overlay nodes.

8.3.6 Identifying Attackers

It is possible to reverse the adaptive flooding attack to locate a single compromised node, if the lightweight authenticator can be changed easily. The search operates in an analogous fashion to the adaptive flooding attack: The server distributes key A to half of the nodes, and key B to the other half. When an attack is initiated with key A, the server knows that the attacker has compromised a machine in that half of the nodes. The search can then continue to narrow down the possibly compromised nodes until corrective action can be taken. This response is likely to require the agility of destination address authentication.

8.4 Analysis

Analysis of "backscatter" traffic in 2001 suggests that more than 30% of observed DDoS SYN-flood or direct ICMP attacks involved 1000 packets per second (pps) or more, and that about 5% of them involved more than 10,000 pps [102]. This study did not observe indirect attacks that can take advantage of traffic amplifiers, and which can achieve even larger attack rates. Fortunately, these indirect attacks can often be stopped using source address authentication: we know of no attacks that can indirectly generate spoofed traffic.

How powerful are these attacks relative to the sites they attack? A T1 line (~ 1.54 Mbits/s) is likely the smallest access link that would be used by a "critical" service today. With full-size packets (typically 1500 bytes), a T1 line can handle just 128 packets per second. The 30th percentile of DoS attacks is nearly an order of magnitude larger than this. A server in a co-location center with a 10 Mbits/s Ethernet connection can handle about 830 pps, and a 100 Mbits/s connected server could not withstand the upper 5% of DoS attacks at 10,000 pps.

For a victim on a T1 line, the top 5% of attacks could mount an adaptive flooding attack against a 100-node overlay with source authentication in under eight rounds: Dividing the 10,000 pps by 50 nodes gives 200 packets per spoofed node per second, more than the T1 can handle. Thus, an attacker can immediately conduct a binary search of the space of egress node IP addresses, taking about $\log_2(100)$ rounds.

Many of the IP ID attacks take about 10 packets per attempted key. At 1000 pps, an attacker could discover a destination-port key in about five minutes. In a doubly-indirect overlay using source address authentication (SOS), the attacker could expect to locate the egress node's IP address in about 50 seconds. Using both of these keys, however, would force the attacker to spend nearly four days scanning at extremely high packet rates, and using both source and destination port filtering would provide a 32-bit key-space that would require months of probing.

Resource consumption attacks, such as SYN floods, can be destructive even at lower packet rates; One study noted that a Linux Web server could handle only up to 500 pps of SYN packets before experiencing performance degradation [37]. SYN packets are also smaller, and are thus easier for an attacker to generate in large volume. By attacking multiple ingress nodes, an attacker could attempt to degrade the availability of the overlay. The top 5% of the attacks, over 10,000 pps, could disable about $\frac{10{,}000}{500} = 20$ overlay nodes. Modern TCP stacks with SYN cookies or compressed TCP state can handle higher packet rates than older systems, but SYN floods will still consume more server resources than pure flooding attacks.

Simple filtering schemes that use only the overlay node source address for authentication are vulnerable to probing by attackers with extremely modest attack capacity, and to adaptive flooding attacks by more powerful attackers. Using a 16 bit authentication key space such as destination port filtering is sufficient to guard against adaptive flooding attacks by all but the largest attackers, but may still be vulnerable to probing attacks. While care must be taken to guard against host vulnerabilities, a Mayday-like system can, if sufficient bits are used for filtering, effectively guard against even the highest rate DDoS attacks.

8.5 Practical Deployment Issues

We believe it is feasible to deploy a Mayday-like system. CDN providers like Akamai [2] have existing overlay networks that number in the thousands of nodes. Routers are increasingly capable of performing hardware-based packet filtering at line speed on high-speed links [77]. ISPs have historically been willing to implement filtering to mitigate large DoS attacks; this willingness was tempered by the inability of their routers to do line-speed filtering. With the deployment of ASIC-assisted filters, ISPs should be able to deploy a few access list entries, especially on behalf of major customer networks.

Mayday is primarily useful for protecting centralized services. Services may use a central server to ease their design, or it may not be economically feasible for a single service to purchase many under-utilized nodes to protect itself from attacks. In these cases, it may be particularly useful to take a service-provider approach, in which multiple clients contract with a single Mayday provider, who provides a shared overlay infrastructure. The service-provider approach helps amortize the cost of the overlay across multiple clients, and provides shared excess capacity to deal with transient load spikes. Protecting clients in this manner allows the deployment of a larger overlay network, and reduces the number of entities that ISPs must deal with for creating router access lists.

Finally, DDoS protection is only the first line of defense for servers. The objective of Mayday is to prevent flooding attacks from overwhelming servers. In the real world, servers have a host of additional security problems that they must contend with, and interior lines of defense must still be maintained.

8.6 Summary

Mayday is a general architecture for using efficient router filtering with semi-trusted overlay nodes to provide denial of service resistance to servers. By generalizing from earlier work, we presented several novel mechanisms that can provide improved performance with equivalent security. Designers implementing the Mayday architecture gain the flexibility to trade security for performance to better create a system that matches their needs.

To illustrate how overlay-based DoS protection would work in the real world, we presented several attacks that are effective against many router-based DoS prevention schemes. By providing options for more precise filtering and more agile rule updates, the Mayday architecture can successfully reduce the impact of these attacks.

While the Mayday architecture can provide a practical and effective proactive defense against DoS attacks, much work remains to be done. Current router architectures are vulnerable to probes like our next-hop scan, and correcting these vulnerabilities will take time. Not all services can afford to protect themselves with Mayday, but they still require some protection. There have been many proposals for detecting and preventing DoS attacks at the network layer and up, and sorting through the options remains a formidable task.

It's not about the bike...

- Lance Armstrong

Chapter 9

Conclusion

This dissertation addressed several major impediments to availability on the Internet. By exploring an "end-to-end" view of availability, the systems we presented are able to mask failures that are not masked by the existing Internet routing substrate. This chapter concludes by briefly summarizing our contributions, and then discussing open issues and future work.

9.1 Summary

This dissertation began by identifying several challenges to end-to-end availability. In today's networks, user-visible failures are caused not only by simple link failures, but also by protocol errors, human errors, or unexpected hardware and software errors. Frequent causes of failures include:

- **Routing delays and policy problems**: When links fail, inter-domain routing protocols can take several minutes to reach a new, stable topology. While they do eventually converge, connectivity is impaired during this period of delayed convergence. Routing policies can also prevent the routing protocols from using good links as backup links, creating unnecessary disconnections.

- **Unexpected failure modes**: Operator error, hardware and software bugs, and congestion can all interrupt end-to-end reachability without the failure being reflected at the Internet routing layer. From the network layer's perspective, all is well, but application packets are not delivered.

- **A complex system of fallible components**: The proper functioning of Internet services depends on working links, routers, interior and exterior routing protocols, DNS servers, client and server software, and so on. Each of these systems is a relatively complex distributed system on its own, and changes made to correct problems are nearly as likely to cause new problems. The failure of a single one of these systems can prevent clients from making effective use of Internet services.

- **Malice**: Malicious traffic is responsible for an increasing number of Internet outages. The relatively open and unauthenticated design of the Internet makes it difficult to detect and prevent these attacks, and their severity and frequency is increasing.

In the face of these failures, merely routing around link-level failures does not guarantee the success of end-to-end communication. Instead, we proposed an end-to-end view of availability, with systems that perform their own path selection based upon end-to-end criteria. This dissertation explored these ideas through three overlay-network based systems. The first two systems, RON and MONET, were designed to mask Internet failures. The third, Mayday, helps shield servers from malicious traffic:

- **Resilient Overlay Networks (RON)**: RON creates an overlay network between a small group of cooperating nodes. RON nodes probe the path conditions between each other, and then use an overlay routing protocol to determine whether to send packets directly over the Internet, or indirectly via another node. RON's key contribution is demonstrating that overlay-based alternate path routing is a practical and effective technique for quickly avoiding Internet path failures—on the order of a few seconds. The analysis in Chapter 5 showed that single-hop indirect routing is sufficient to achieve the majority of RON's benefits.

- **Multi-homed Overlay Networks (MONET)**: MONET combines an overlay network with physical multi-homing to overcome both failures within the Internet and access-link failures. Unlike RON, which enhances communication *within* a group, MONET provides increased availability for access to unmodified external Web sites. In addition to providing about an order of magnitude improvement in availability, our measurements of a deployed MONET allowed us to accurately categorize the sources of Internet failures observed by the MONET proxies. MONET's *waypoint selection* algorithm ensured that the proxies achieved improved availability with little overhead.

- **Mayday**: While RON and MONET help guard against accidental failures and can increase resilience against some attacks, an attacker can still craft specific, effective attacks. Mayday addresses this last source of failures by shielding Internet servers from distributed denial-of-service attacks. It does so by surrounding the servers with a ring of packet filters and by using a group of overlay nodes to provide trusted clients with access through the packet filters. Mayday provides a range of implementation options that permit system designers to trade overhead for attack resistance.

The extensive real-world evaluations of the production RON and MONET implementations presented in Chapters 5 and 7 demonstrated several important points:

1. Overlay routing can *improve* upon the underlying network.
2. End-to-end measurements effectively guide path selection.
3. It is possible to provide higher availability between *actively* communicating hosts than between *potentially* communicating hosts.
4. Waypoint selection can quickly locate working paths without requiring excessive probing traffic.

9.1.1 Software and Data

The software implementations for RON and MONET are available on-line. RON has been tested under FreeBSD versions 4.7 through 4.10, and on an older version of Linux 2.2, but does not work on more recent versions of Linux that do not provide divert sockets. MONET has been tested under FreeBSD 4, FreeBSD 5, and Linux 2.6.

For privacy reasons, the datasets used to evaluate MONET are not publicly available. The RON_1 and RON_2 datasets, and a number of other datasets collected during this research using the RON testbed, are available, along with the software implementations, from:

`http://nms.lcs.mit.edu/ron/`

The MONET proxy deployment is described in further detail at:

`http://nms.lcs.mit.edu/ron/ronweb/`

9.2 Open Issues and Future Work

The work presented in this dissertation raises a number of questions about both the techniques presented herein and network architecture in general.

9.2.1 Scalability and Stability

RON and MONET both scale by adding more independent confederations of nodes, instead of by increasing the size of a single confederation. Several related research efforts have studied the network effects of wide-scale deployment of overlay routing schemes [132, 124]. These studies used simplified overlay networks and simplified network models to make the analysis tractable; to our knowledge, there are no studies that examine the large-scale scaling and stability behavior of overlay networks in a real setting.

Roughgarden *et al.* present a game-theoretic analysis of the impact of "selfish routing" in which nodes are permitted choose the path their packets take through the network [132]. The conclusions of this analysis vary depending on the network model used: If network "cost" is linear in the amount of traffic, then the selfish solution has at most $\frac{4}{3}$ the cost of an optimal solution. If, however, the network follows an M/M/1 queuing model, then the selfish solution *could* have an arbitrarily higher cost. Because this study considered a highly abstract model of the network, it is difficult to directly adapt these results to the behavior of RON and MONET on the real Internet.

Qiu *et al.* studied the question asked by Roughgarden *et al.* through a simulation of an internal ISP-like topology using OSPF-like routing [124]. In this somewhat more realistic model, the authors found that selfish routing often *outperforms* the solution found by the conventional routing protocol, though it does lead to increased link utilization on certain popular links. These results suggest two things: First, that the worst-case scenarios evaluated by Roughgarden *et al.* may not occur frequently in real topologies; and second, that existing routing protocols do not find optimal solutions.

These two studies examined the steady-state performance of the network after the network and overlay routing protocols stabilized, but *will* they stabilize? Keralapura *et al.* find in a simple simulation that RON-like overlay routing may lead to oscillatory conditions. The RON system includes considerable randomization, which was not accounted for in the simulation study, precisely to avoid such oscillations. Further study is, however, needed: we observed some of this oscillatory behavior in practice when routing high-bandwidth flows across multiple low-capacity links. The problem of routing a single large flow is particularly challenging to any routing system. It seems likely that some form of concurrent multi-path routing is needed both to take advantage of the capacity of multiple links and to avoid potential oscillations.

9.2.2 Policy issues

Overlay networks potentially conflict with network policies. In the cases of RON and MONET, traffic can only be routed through a collaborating node; in an important sense, all of the network capacity used "belongs to" one of the participants in the overlay. From this perspective, overlay routing increases the amount of traffic that an ISPs customer sends and receives, which should increase the ISP's revenue. This viewpoint, however, ignores the frequent over-subscription in ISP pricing as of the time of writing. It may be that overlay routing can, in fact, take advantage of a "loophole" in many ISP's pricing models, which depend on clients not actually using much of the capacity of their links. If overlay routing becomes widespread, such loopholes are likely to close.

A more fundamental question arises, however, about the general case of clients being allowed to choose the paths their packets take through the network. Several projects have proposed *mechanisms* for permitting users to select which of several ISPs carries their traffic [176, 125]; others have examined particular router extensions that let overlays push indirect routing changes deeper into the network, avoiding the need for traffic to transit the users access link [74]. Both these proposals and the work in this dissertation leave open the question of *what* policies ISPs and other entities would want to use to govern the indirect flow of traffic through their networks.

9.2.3 Future Directions

Object-level or service-level availability

RON focused on providing connectivity between two host computers. MONET stepped up a level to provide connectivity between a host and a Web site, which could be represented by multiple server computers. By making this shift, MONET realized additional availability by masking the failure of a single server. MONET still primarily functions at the connection level. Moving to an architecture that attempted to provide higher availability at *object* or *service* granularity would enable a failure masking system to survive additional classes of failures, such as failures in the middle of a transfer, data corruption, and so on.

Architectural support for path selection

The research described in this dissertation derived a portion of its benefits from the application-specific nature of overlay networks, and a portion of its benefits merely from using overlays to gain access to multiple paths through the network. The latter ability could be incorporated into the basic Internet architecture, by providing "best effort" route choice: depending on the value of a path selector in the packet header, Internet routers would attempt to ensure that different values resulted in the the packet's taking a different network path.

Such a change would not provide all of the benefits of a RON or MONET-like solution, such as the ability to multi-home without requiring routing support, or application-level caching. Architectural support for path choice would, however, with arbitrary Internet applications and it could be more efficient than the overlay based solution. An architectural approach would be ripe for optimization—while the basic path selection model would remain best effort, the algorithms used to determine those paths could evolve considerably.

Bibliography

[1] Active network backbone (ABone). `http://www.isi.edu/abone/`.

[2] Akamai. http://www.akamai.com, 1999.

[3] Aditya Akella, Bruce Maggs, Srinivasan Seshan, Anees Shaikh, and Ramesh Sitaraman. A measurement-based analysis of multihoming. In *Proc. ACM SIGCOMM*, Karlsruhe, Germany, August 2003.

[4] Aditya Akella, Jeff Pang, Bruce Maggs, Srinivasan Seshan, and Anees Shaikh. A comparison of overlay routing and multihoming route control. In *Proc. ACM SIGCOMM*, Portland, OR, August 2004.

[5] Aditya Akella, Srinivasan Seshan, and Anees Shaikh. Multihoming performance benefits: An experimental evaluation of practical enterprise strategies. In *Proc. USENIX Annual Technical Conference*, Boston, MA, June 2004.

[6] Cengiz Alaettinoglu, Van Jacobsen, and Haobo Yu. *Towards Mili-Second IGP Convergence*. Internet Engineering Task Force, November 2000. Internet-Draft draft-alattinoglu-isis-convergence-00 `http://www.packetdesign.com/news/industry-publications/drafts/convergen%ce.pdf`. Work in progress, expires May 2001.

[7] P. Albitz and C. Ciu. *DNS and BIND in a Nutshell*. O'Reilly and Associates, 1992.

[8] Mark Allman, Ethan Blanron, and Wesley M. Eddy. A scalable system for sharing Internet measurement. In *Passive & Active Measurement (PAM)*, Fort Collins, CO, March 2004.

[9] David G. Andersen, Hari Balakrishnan, M. Frans Kaashoek, and Robert Morris. Resilient Overlay Networks. In *Proc. 18th ACM Symposium on Operating Systems Principles (SOSP)*, pages 131–145, Banff, Canada, October 2001.

[10] David G. Andersen, Hari Balakrishnan, M. Frans Kaashoek, and Robert Morris. Experience with an Evolving Overlay Network Testbed. *ACM Computer Communications Review*, 33(3):13–19, July 2003.

[11] David G. Andersen, Nick Feamster, Steve Bauer, and Hari Balakrishnan. Topology inference from BGP routing dynamics. In *Proc. ACM SIGCOMM Internet Measurement Workshop*, Marseille, France, November 2002.

[12] David G. Andersen, Alex C. Snoeren, and Hari Balakrishnan. Best-path vs. multi-path overlay routing. In *Proc. ACM SIGCOMM Internet Measurement Conference*, Miami, FL, October 2003.

[13] Arbor Networks. Peakflow for enterprises datasheet. `http://arbornetworks.com/up_media/up_files/Pflow_Enter_datasheet2.1.pdf%`, 2002.

[14] Mohit Aron, Darren Sanders, Peter Druschel, and Willy Zwaenepoel. Scalable content-aware request distribution in cluster-based network servers. In *Proc. USENIX Annual Technical Conference*, San Diego, CA, June 2000.

[15] Asta Networks. Convergence of security and network performance (vantage system overview). `http://www.astanetworks.com/products/data_sheets/asta_data_sheet.pdf`, 2002.

[16] H. Balakrishnan, S. Seshan, M. Stemm, and R.H. Katz. Analyzing Stability in Wide-Area Network Performance. In *Proc. ACM SIGMETRICS*, pages 2–12, Seattle, WA, June 1997.

[17] Bellcore. SR-NWT-001756, automatic protection switching for SONET, issue 1. Technical report, October 1990.

[18] Steve Bellovin. *ICMP Traceback Messages, Internet-Draft, draft-bellovin-itrace-00.txt, Work in Progress*, March 2000.

[19] SYN Cookies. http://cr.yp.to/syncookies.html, 1996.

[20] CAIDA's Skitter project, 2002. `http://www.caida.org/tools/measurement/skitter/`.

[21] CAIDA Analysis of Code-Red. `http://www.caida.org/analysis/security/code-red/`, 2002.

[22] CAIRN Home Page. http://www.isi.edu/div7/cairn/, 1996.

[23] Don Caldwell, Anna Gilbert, Joel Gottlieb, Albert Greenberg, Gisli Hjalmtysson, and Jennifer Rexcord. The cutting EDGE of IP router configuration. In *Proc. 2nd ACM Workshop on Hot Topics in Networks (Hotnets-II)*, Cambridge, MA, November 2003.

[24] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems*, 20(4):398–461, November 2002.

[25] A. Chankhunthod, P. Danzig, C. Neerdaels, M.F. Schwartz, and K.J. Worrell. A Hierarchical Internet Object Cache. In *Proc. USENIX Annual Technical Conference*, San Diego, CA, January 1996.

[26] D. Chaum. Untraceable electronic mail, return addresses and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, 1981.

[27] Cisco Security Advisory: Code Red Worm - Customer Impact. `http://www.cisco.com/warp/public/707/cisco-code-red-worm-pub.shtml`, 2001.

[28] David Clark. *Policy Routing in Internet Protocols*. Internet Engineering Task Force, May 1989. RFC 1102.

[29] I. Clarke, O. Sandbert, B. Wiley, and T. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Proc. the Workshop on Design Issues in Anonymity and Unobservability*, Berkeley, CA, July 2000.

[30] Archie Cobbs. DIVERT(4) manual page for FreeBSD, September 1996.

[31] L. Coene. Multihoming issues in the Stream Control Transmission Protocol. http://www.ietf.org/internet-drafts/draft-coene-sctp-multihome-04.txt, June 2003.

[32] Edith Cohen and Haim Kaplan. Prefetching the means for document transfer: A new approach for reducing Web latency. In *Proc. IEEE INFOCOM*, volume 2, pages 854–863, Tel-Aviv, Israel, March 2000.

[33] Andy Collins. The Detour Framework for Packet Rerouting. Master's thesis, University of Washington, October 1998.

[34] Mark Crovella and Robert Carter. Dynamic server selection in the Internet. In *Proc. Third IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems (HPCS)*, 1995.

[35] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with CFS. In *Proc. 18th ACM Symposium on Operating Systems Principles (SOSP)*, Banff, Canada, October 2001.

[36] Mike Dahlin, Bharat Chandra, Lei Gao, and Amol Nayate. End-to-end WAN Service Availability. *IEEE/ACM Transactions on Networking*, 11(2), April 2003.

[37] Tina Darmohray and Ross Oliver. Hot spares for DoS attacks. *;Login: The Magazine of USENIX and SAGE*, July 2000.

[38] Drew Dean, Matt Franklin, and Adam Stubblefield. An algebraic approach to IP traceback. *Information and System Security*, 5(2):119–137, 2002.

[39] Sven Dietrich, Neil Long, and David Dittrich. Analyzing distributed denial of service tools: The shaft case. In *Proceedings of LISA*, pages 329–339, 2000.

[40] R. Dingledine, M. Freedman, and D. Molnar. The Free Haven Project: Distributed anonymous storage service. In *Proc. Workshop on Design Issues in Anonymity and Unobservability*, Berkeley, CA, July 2000.

[41] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proc. 13th USENIX Security Symposium*, San Diego, CA, August 2004.

[42] Sean Donelan. Update: CSX train derailment. `http://www.merit.edu/mail.archives/nanog/2001-07/msg00351.html`.

[43] Sandra G. Dykes, Kay A. Robbins, and Clinton L. Jeffery. An empirical evaluation of client-side server selection algorithms. In *Proc. IEEE INFOCOM*, volume 3, pages 1361–1370, Tel-Aviv, Israel, March 2000.

[44] R. Elz, R. Bush, S. Bradner, and M. Patton. *Selection and Operation of Secondary DNS Servers*. Internet Engineering Task Force, July 1997. RFC 2182 / BCP 16.

[45] Patricia Enriquez, Aaron Brown, and David A. Patterson. Lessons from the PSTN for dependable computing. In *Workshop on Self-Healing, Adaptive and Self-Managed Systems*, 2002.

[46] H. Eriksson. Mbone: The Multicast Backbone. *Communications of the ACM*, 37(8):54–60, 1994.

[47] Li Fan, Pei Cao, Jussara Almeida, and Andrei Z. Broder. Summary cache: A scalable wide-area Web cache sharing protocol. In *Proc. ACM SIGCOMM*, pages 254–265, Vancouver, British Columbia, Canada, September 1998.

[48] Rik Farrow. Routing instability on the Internet. *Network Magazine*, March 4, 2002. `http://www.networkmagazine.com/article/NMG20020304S0007/2`.

[49] FatPipe. FatPipe WARP FAQ. `http://www.fatpipeinc.com/warp/warpfaq.htm`, 2003.

[50] Nick Feamster. Practical verification techniques for wide-area routing. In *Proc. 2nd ACM Workshop on Hot Topics in Networks (Hotnets-II)*, Cambridge, MA, November 2003.

[51] Nick Feamster, David Andersen, Hari Balakrishnan, and M. Frans Kaashoek. Measuring the effects of Internet path faults on reactive routing. In *Proc. ACM SIGMETRICS*, San Diego, CA, June 2003.

[52] Paul Ferguson and Daniel Senie. *Network Ingress Filtering*. Internet Engineering Task Force, May 2000. Best Current Practice 38, RFC 2827.

[53] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-Based Congestion Control for Unicast Applications. In *Proc. ACM SIGCOMM*, pages 43–54, Stockholm, Sweden, September 2000.

[54] Paul Francis. Yoid: Extending the Internet multicast architecture. `http://www.aciri.org/yoid/docs/yoidArch.ps`, April 2000.

[55] Michael J. Freedman, Eric Freudenthal, and David Mazières. Democratizing content publication with Coral. In *Proc. First Symposium on Networked Systems Design and Implementation (NSDI)*, San Francisco, CA, March 2004.

[56] Michael J. Freedman and David Mazières. Sloppy hashing and self-organizing clusters. In *Proc. 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, Berkeley, CA, February 2003.

[57] Michael J. Freedman and Robert Morris. Tarzan: A peer-to-peer anonymizing network layer. In *Proc. 9th ACM Conference on Computer and Communications Security*, Washington, D.C., November 2002.

[58] David K. Goldenberg, Lili Qiu, Haiyong Xie, Yang Richard Yang, and Yin Zhang. Optimizing cost and performance for multihoming. In *Proc. ACM SIGCOMM*, pages 79–92, Portland, OR, August 2004.

[59] David Goldsmith and Michael Schiffman. Firewalking: A traceroute-like analysis of IP packet responses to determine gateway access control lists. `http://www.packetfactory.net/firewalk/`, 1998.

[60] Ramesh Govindan and Hongsuda Tangmunarunkit. Heuristics for Internet map discovery. In *Proc. IEEE INFOCOM*, pages 1371–1380, Tel-Aviv, Israel, March 2000. IEEE.

[61] Mukul Goyal, Roch Guerin, and Raju Rajan. Predicting TCP Throughput From Non-invasive Sampling. In *Proc. IEEE INFOCOM*, New York, NY, June 2002.

[62] Jim Gray. Why do computers stop and what can be done about it? Technical Report 85.7, Tandem Computers, June 1985.

[63] David Greenfield. RadWare Linkproof. *Network Magazine*, December 1999. `http://www.networkmagazine.com/article/NMG20000427S0005`.

[64] Krishna P. Gummadi, Harsha V. Madhyastha, Steven D. Gribble, Henry M. Levy, and David Wetherall. Improving the reliability of Internet paths with one-hop source routing. In *Proc. 6th USENIX OSDI*, San Francisco, CA, December 2004.

[65] Fanglu Guo, Jiawu Chen, Wei Li, and Tzi cker Chiueh. Experiences in building a multihoming load balancing system. In *Proc. IEEE INFOCOM*, Hong Kong, March 2004.

[66] Robert M. Hinden. IP Next Generation overview. *Communications of the ACM*, 39(6):61–71, 1996.

[67] Ningning Hu, Li (Erran) Li, and Zhuoqing Morley Mao. Locating Internet bottlenecks: Algorithms, measurements, and implications. In *Proc. ACM SIGCOMM*, pages 41–54, Portland, OR, August 2004.

[68] Yang hua Chu, Sanjay G. Rao, and Hui Zhang. A case for end system multicast. In *Proc. ACM SIGMETRICS*, pages 1–12, Santa Clara, CA, June 2000.

[69] Gianluca Iannaccone, Chen nee Chuah, Richard Mortier, Supratik Bhattacharyya, and Christophe Diot. Analysis of link failures in an IP backbone. In *Proc. ACM SIGCOMM Internet Measurement Workshop*, Marseille, France, November 2002.

[70] Internet Engineering Task Force. *The Secure HyperText Transfer Protocol*, August 1999. RFC 2660.

[71] Internet2. `http://www.internet2.edu/`.

[72] John Ioannidis and Steven M. Bellovin. Implementing Pushback: Router-Based Defense Against DDoS Attacks. In *Proc. Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, February 2002.

[73] Manish Jain and Constantinos Dovrolis. End-to-End Available Bandwidth: Measurement Methodology, Dynamics, and Relation with TCP Throughput. In *Proc. ACM SIGCOMM*, Pittsburgh, PA, August 2002.

[74] John Jannotti. Network layer support for overlay networks. In *Proc. 5th International Conference on Open Architectures and Network Programming (OPENARCH)*, New York, NY, June 2002.

[75] John Jannotti, David K. Gifford, Kirk L. Johnson, M. Frans Kaashoek, and James W. O'Toole Jr. Overcast: Reliable multicasting with an overlay network. In *Proc. 4th USENIX OSDI*, pages 197–212, San Diego, CA, November 2000.

[76] Jaeyeon Jung, Emil Sit, Hari Balakrishnan, and Robert Morris. DNS Performance and the Effectiveness of Caching. In *Proc. ACM SIGCOMM Internet Measurement Workshop*, San Fransisco, CA, November 2001.

[77] Juniper Networks. M-series Internet Processor II ASIC Frequently Asked Questions. `http://www.juniper.net/solutions/faqs/m-series_ip2.html`.

[78] Srikanth Kandula, Dina Katabi, Matthias Jacob, and Arthur Berger. Botz-4-Sale: Surviving Organized DDoS Attacks That Mimic Flash Crowds. Technical Report LCS TR-969, Laboratory for Computer Science, MIT, 2004.

[79] Rohit Kapoor, Ling-Jyh Chen, Li Lao, Mario Gerla, and M. Y. Sanadidi. CapProbe: A Simple and Accurate Capacity Estimation Technique. In *Proc. ACM SIGCOMM*, pages 67–78, Portland, OR, August 2004.

[80] Ram Keralapura, Nina Taft, Chen-Nee Chuah, and Gianluca Iannaccone. Can ISPs Take the Heat from Overlay Networks? In *Proc. 3nd ACM Workshop on Hot Topics in Networks (Hotnets-III)*, San Diego, CA, November 2004.

[81] Angelos D. Keromytis, Vishal Misra, and Dan Rubenstein. SOS: Secure overlay services. In *Proc. ACM SIGCOMM*, pages 61–72, Pittsburgh, PA, August 2002.

[82] Atul Khanna and John Zinky. The Revised ARPANET Routing Metric. In *Proc. ACM SIGCOMM*, pages 45–56, Austin, TX, September 1989.

[83] Dejan Kostic, Adolfo Rodriguez, Jeannie Albrecht, and Amin Vahdat. Bullet: High bandwidth data dissemination using an overlay mesh. In *Proc. 19th ACM Symposium on Operating Systems Principles (SOSP)*, Lake George, NY, October 2003.

[84] Balachander Krishnamurthy and Jennifer Rexford. *Web Protocols and Practice*. Addison Wesley, Upper Saddle River, NJ, 2001.

[85] D. Richard Kuhn. Sources of failure in the public switched telephone network. *IEEE Computer*, 30(4), 1997.

[86] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian. Delayed Internet Routing Convergence. In *Proc. ACM SIGCOMM*, pages 175–187, Stockholm, Sweden, September 2000.

[87] C. Labovitz, A. Ahuja, and F. Jahanian. Experimental Study of Internet Stability and Wide-Area Backbone Failures. In *Proc. 29th International Symposium on Fault-Tolerant Computing*, June 1999.

[88] C. Labovitz, R. Malan, and F. Jahanian. Internet Routing Instability. *IEEE/ACM Transactions on Networking*, 6(5):515–526, 1998.

[89] Craig Labovitz, Abha Ahuja, and F. Jahanian. Experimental study of Internet stability and wide-area network failures. In *Proc. FTCS*, Madison, WI, June 1999.

[90] Jonathan Lemon. Resisting SYN flood DoS attacks with a SYN cache. In *BSDCon*, February 2002.

[91] K. Lindqvist. Multihoming road-map for IPv6. http://www.ietf.org/internet-drafts/draft-kurtis-multi6-roadmap-00.txt, February 2003.

[92] Ratul Mahajan, David Wetherall, and Tom Anderson. Understanding BGP misconfiguration. In *Proc. ACM SIGCOMM*, pages 3–17, Pittsburgh, PA, August 2002.

[93] Zhuoqing Morley Mao, Ramesh Govindan, George Varghese, and Randy Katz. Route Flap Damping Exacerbates Internet Routing Convergence. In *Proc. ACM SIGCOMM*, Pittsburgh, PA, August 2002.

[94] David Mazières, Michael Kaminsky, M. Frans Kaashoek, and Emmett Witchel. Separating key management from file system security. In *Proc. 17th ACM Symposium on Operating Systems Principles (SOSP)*, pages 124–139, Kiawah Island, SC, December 1999.

[95] Mazu Networks. `http://www.mazunetworks.com/solutions/`, 2002.

[96] S. McCanne and V. Jacobson. The BSD Packet Filter: A New Architecture for User-Level Packet Capture. In *Proc. Winter USENIX Conference*, pages 259–269, January 1993.

[97] Joseph Menn. Deleting online extortion. *Los Angeles Times*, October 25, 2004.

[98] Gary Miller. Overlay routing networks (Akarouting). `http://www-math.mit.edu/~steng/18.996/lecture9.ps`, April 2002.

[99] J. Mirkovic and P. Reiher. A taxonomy of DDoS attacks and defense mechanisms. *ACM Computer Communications Review*, 34(2):39–54, 2004.

[100] Paul V. Mockapetris and Kevin J. Dunlap. Development of the Domain Name System. In *Proc. ACM SIGCOMM*, pages 123–133, Stanford, California, September 1998. ACM. also in *Computer Communication Review* 18 (4), Aug. 1988.

[101] Thomas Moestl and Paul Rombouts. pdnsd. `http://www.phys.uu.nl/~rombouts/pdnsd.html`, 2004.

[102] David Moore, Geoffrey Voelker, and Stefan Savage. Inferring Internet denial of service activity. In *Proc. 10th USENIX Security Symposium*, Washington, DC, August 2001.

[103] J. Moy. *OSPF Version 2*, March 1994. RFC 1583.

[104] Athicha Muthitacharoen, Robert Morris, Thomer Gil, and Benjie Chen. Ivy: A read/write peer-to-peer file system. In *Proc. 5th USENIX OSDI*, Boston, MA, December 2002.

[105] Aki Nakao, Larry Peterson, and Mike Wawrzoniak. A divert mechanism for service overlays. Technical Report TR-668-03, Princeton University, December 2002.

[106] Acopia Networks. Optimizing file storage performance with adaptive resource networking, 2003.

[107] David Oppenheimer, Archana Ganapathi, and David A. Patterson. Why do Internet services fail, and what can be done about it? In *Proc. 4th USENIX Symposium on Internet Technologies and Systems (USITS)*, Seattle, Washington, March 2003.

[108] D. Oran. *OSI IS-IS intra-domain routing protocol*. Internet Engineering Task Force, February 1990. RFC 1142.

[109] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Throughput: A Simple Model and its Empirical Validation. In *Proc. ACM SIGCOMM*, pages 303–323, Vancouver, British Columbia, Canada, September 1998.

[110] Vivek S. Pai, Limin Wang, KyoungSoo Park, Ruoming Pang, and Larry Peterson. The dark side of the Web: An open proxy's view. `http://codeen.cs.princeton.edu/attack_defense.pdf`.

[111] Vasileios Pappas, Zhiguo Xu, Songwu Lu, Daniel Massey, Andreas Terzis, and Lixia Zhang. Impact of configuration errors on DNS robustness. In *Proc. ACM SIGCOMM*, pages 319–330, Portland, OR, August 2004.

[112] Kihong Park and Heejo Lee. On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law Internets. In *Proc. ACM SIGCOMM*, San Diego, CA, August 2001.

[113] KyoungSoo Park, Vivek Pai, Larry Peterson, and Zhe Wang. CoDNS: Improving DNS performance and reliability via cooperative lookups. In *Proc. 6th USENIX OSDI*, San Francisco, CA, December 2004.

[114] C. Partridge. *Using the Flow Label Field in IPv6*. Internet Engineering Task Force, June 1995. RFC 1809.

[115] V. Paxson. End-to-End Routing Behavior in the Internet. In *Proc. ACM SIGCOMM*, pages 25–38, Stanford, CA, August 1996.

[116] V. Paxson. End-to-End Internet Packet Dynamics. In *Proc. ACM SIGCOMM*, pages 139–152, Cannes, France, September 1997.

[117] V. Paxson. End-to-End Routing Behavior in the Internet. *IEEE/ACM Transactions on Networking*, 5(5):601–615, 1997.

[118] Vern Paxson. Strategies for sound Internet measurement. In *Proc. ACM SIGCOMM Internet Measurement Conference*, Taormina, Sicily, Italy, October 2004.

[119] Vern Paxson, Andrew Adams, and Matt Mathis. Experiences with NIMI. In *Passive & Active Measurement (PAM)*, April 2000.

[120] Vern Paxson and Mark Allman. *Computing TCP's Retransmission Timer*. Internet Engineering Task Force, November 2000. RFC 2988.

[121] C. Perkins. *IP Mobility Support*. Internet Engineering Task Force, October 1996. RFC 2002.

[122] Larry Peterson, Tom Anderson, David Culler, and Timothy Roscoe. A blueprint for introducing disruptive technology into the Internet. In *Proc. 1st ACM Workshop on Hot Topics in Networks (Hotnets-I)*, Princeton, NJ, October 2002.

[123] J. B. Postel. *Transmission Control Protocol*. Internet Engineering Task Force, September 1981. RFC 793.

[124] Lili Qiu, Yang Richard Yang, Yin Zhang, and Scott Shenker. On selfish routing in Internet-like environments. In *Proc. ACM SIGCOMM*, Karlsruhe, Germany, August 2003.

[125] Barath Raghavan and Alex C. Snoeren. A system for authenticated policy-compliant routing. In *Proc. ACM SIGCOMM*, Portland, OR, August 2004.

[126] Y. Rekhter and T. Li. *A Border Gateway Protocol 4 (BGP-4)*. Internet Engineering Task Force, March 1995. RFC 1771.

[127] Network Reliability and Interoperability Council (NRIC). Network reliability: A report to the nation, 1993. `http://www.nric.org/pubs/nric1/index.html`.

[128] Network Reliability and Interoperability Council (NRIC). Network reliability—the path forward, 1996. http://www.nric.org/pubs/nric2/fg1/index.html.

[129] Zona Research. The need for speed II. *Zona Market Bulletin*, 5, April 2001. `http://www.keynote.com/downloads/Zona_Need_For_Speed.pdf`.

[130] Jennifer Rexford, Jia Wang, Zhen Xiao, and Yin Zhang. BGP routing stability of popular destinations. In *Proc. ACM SIGCOMM Internet Measurement Workshop*, Marseille, France, November 2002.

[131] Pablo Rodriguez and Ernst W. Biersack. Dynamic parallel access to replicated content in the Internet. *IEEE/ACM Transactions on Networking*, 10(4), August 2002.

[132] Tim Roughgarden. *Selfish Routing*. PhD thesis, Cornell University, May 2002.

[133] RouteScience. Whitepaper available from `http://www.routescience.com/technology/tec_whitepaper.html`.

[134] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Proc. 18th IFIP/ACM International Conference on Distributed Systems Platforms*, November 2001.

[135] Justin Ryburn. Re: possible l3 issues, February 2004. `http://www.merit.edu/mail.archives/nanog/2004-02/msg00814.html`, plus private communication from list members who wished to remain anonymous.

[136] Jerome H. Saltzer and M. Frans Kaashoek. Topics in the engineering of computer systems. Unpublished manuscript; MIT 6.033 class notes, draft release 1.18, January 2004.

[137] Jon Salz, Alex C. Snoeren, and Hari Balakrishnan. TESLA: A transparent, extensible session-layer framework for end-to-end network services. In *Proc. 4th USENIX Symposium on Internet Technologies and Systems (USITS)*, Seattle, Washington, March 2003.

[138] Salvatore Sanfilippo. Posting about the IP ID reverse scan. `http://www.kyuzz.org/antirez/papers/dumbscan.html`, 1998.

[139] Salvatore Sanfilippo. hping home page. `http://www.hping.org/`, 2000.

[140] Stefan Savage, Andy Collins, Eric Hoffman, John Snell, and Tom Anderson. The End-to-End Effects of Internet Path Selection. In *Proc. ACM SIGCOMM*, pages 289–299, Cambridge, MA, September 1999.

[141] Stefan Savage, David Wetherall, Anna Karlin, and Tom Anderson. Network support for IP traceback. *IEEE/ACM Transactions on Networking*, 9(3), June 2001.

[142] S. Seshan, M. Stemm, and R. H Katz. SPAND: Shared Passive Network Performance Discovery. In *Proc. 1st USENIX Symposium on Internet Technologies and Systems (USITS)*, pages 135–146, Monterey, CA, December 1997.

[143] Aman Shaikh, Lampros Kalampoukas, Anujan Varma, and Rohit Dube. Routing Stability in Congested Networks: Experimentation and Analysis. In *Proc. ACM SIGCOMM*, pages 163–174, Stockholm, Sweden, September 2000.

[144] W. Simpson. *The Point-to-Point Protocol (PPP)*. Internet Engineering Task Force, December 1993. RFC 1548.

[145] Emil Sit and Robert Morris. Security considerations for peer-to-peer distributed hash tables. In *Proc. 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, Cambridge, MA, March 2002.

[146] Philip Smith. Weekly routing table report. `http://www.merit.edu/mail.archives/nanog/msg01931.html`, October 2004.

[147] Alex C. Snoeren. *A Session-Based Architecture for Internet Mobility*. PhD thesis, Massachusetts Institute of Technology, December 2002.

[148] Alex C. Snoeren and Hari Balakrishnan. An end-to-end approach to host mobility. In *Proc. ACM Mobicom*, pages 155–166, Boston, MA, August 2000.

[149] Alex C. Snoeren, Craig Partridge, Luis A. Sanchez, Christine E. Jones, Fabrice Tchakountio, Beverly Schwartz, Stephen T. Kent, and W. Timothy Strayer. Single-packet IP traceback. *IEEE/ACM Transactions on Networking*, 10(6), December 2002.

[150] Sockeye. `http://www.sockeye.com/`.

[151] Neil Spring, Ratul Mahajan, and David Wetherall. Measuring ISP topologies with Rocketfuel. In *Proc. ACM SIGCOMM*, Pittsburgh, PA, August 2002.

[152] Squid Web Proxy Cache. `http://www.squid-cache.org/`.

[153] Tyron Stading, Petros Maniatis, and Mary Baker. Peer-to-peer caching schemes to address flash crowds. In *Proc. 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, Cambridge, MA, March 2002.

[154] Angelos Stavrou, Dan Rubenstein, , and Sambit Sahu. A lightweight, robust P2P system to handle flash crowds. In *IEEE International Conference on Network Protocols (ICNP)*, Paris, France, November 2002.

[155] W. R. Stevens. *TCP/IP Illustrated, Volume 1*. Addison-Wesley, Reading, MA, 1994.

[156] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proc. ACM SIGCOMM*, San Diego, CA, August 2001.

[157] Stonesoft. Multi-link technology white paper. `http://www.stonesoft.com/files/products/StoneGate/SG_Multi-Link_Technol%ogy_Whitepaper.pdf`, October 2001.

[158] Jacob Strauss, Dina Katabi, and Frans Kaashoek. A measurement study of available bandwidth estimation tools. In *Proc. ACM SIGCOMM Internet Measurement Conference*, Miami, FL, October 2003.

[159] Lakshminarayanan Subramanian, Volker Roth, Ion Stoica, Scott Shenker, and Randy Katz. Listen and whisper: Security mechanisms for BGP. In *Proc. First Symposium on Networked Systems Design and Implementation (NSDI)*, San Francisco, CA, March 2004.

[160] Liying Tang and Mark Crovella. Virtual landmarks for the Internet. In *Proc. ACM SIGCOMM Internet Measurement Conference*, Miami, FL, October 2003.

[161] R. Thayer, N. Doraswamy, and R. Glenn. *IP Security Document Roadmap*. Internet Engineering Task Force, November 1998. RFC 2411.

[162] The VINT Project. *The* ns *Manual*, April 2002. http://www.isi.edu/nsnam/ns/ns-documentation.html.

[163] J. Touch and S. Hotz. The X-Bone. In *Proc. 3rd Global Internet Mini-Conference in conjunction with IEEE Globecom*, pages 75–83, Sydney, Australia, November 1998.

[164] Joe Touch. Dli95 dartnet overview. `http://www.isi.edu/~touch/dli95/dartnet-dli.html`, 1995.

[165] U.S. Department of Transportation Federal Aviation Administration (FAA). Administrator's fact book. `http://www.atctraining.faa.gov/site/library/factbooks/2003/dec03.pdf`, December 2003.

[166] Fyodor Vaskovich. Nmap stealth port scanner. `http://www.insecure.org/nmap/index.html`, 2002.

[167] C. Villamizar, R. Chandra, and R. Govindan. *BGP Route Flap Damping*. Internet Engineering Task Force, November 1998. RFC 2439.

[168] Paul Vixie and Duane Wessels. *Hyper Text Caching Protocol (HTCP/0.0)*. Internet Engineering Task Force, January 2000. RFC 2756.

[169] Jia Wang. A survey of Web caching schemes for the Internet. *ACM Computer Communications Review*, 25(9):36–46, 1999.

[170] Lan Wang et al. Observation and analysis of BGP behavior under stress. In *Proc. ACM SIGCOMM Internet Measurement Workshop*, Marseille, France, November 2002.

[171] Lan Wang, Xiaoliang Zhao, Dan Pei, Randy Bush, Daniel Massey, Allison Mankin, S. Felix Wu, and Lixia Zhang. Protecting BGP Routes to Top Level DNS Servers. In *Proc. 23rd Intl. Conf on Distributed Computing Systems*, pages 322–331, Providence, RI, May 2003.

[172] D. Wessels and K. Claffy. *Application of Internet Cache Protocol (ICP), version 2*. Internet Engineering Task Force, September 1997. RFC 2187.

[173] D. Wessels and K. Claffy. *Internet Cache Protocol (ICP), version 2*. Internet Engineering Task Force, September 1997. RFC 2186.

[174] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An integrated experimental environment for distributed systems and networks. In *Proc. 5th USENIX OSDI*, pages 255–270, Boston, MA, December 2002.

[175] Alec Wolman, Geoffrey M. Voelker, Nitin Sharma, Neal Cardwell, Anna Karlin, and Henry M. Levy. On the scale and performance of cooperative web proxy caching. In *Proc. 17th ACM Symposium on Operating Systems Principles (SOSP)*, Kiawah Island, SC, December 1999.

[176] Xiaowei Yang. NIRA: A New Internet Routing Architecture. In *ACM SIGCOMM Workshop on Future Directions in Network Architecture*, Karlsruhe, Germany, August 2003.

[177] Alexander Yip. NATRON: Overlay routing to oblivious destinations. Master's thesis, MIT, August 2002.

[178] Chad Yoshikawa, Brent Chun, Paul Eastham, Amin Vahdat, Thomas Anderson, and David Culler. Using smart clients to build scalable services. In *Proc. USENIX Annual Technical Conference*, Anaheim, CA, January 1997.

[179] Ming Zhang, Chi Zhang, Vivek Pai, Larry Peterson, and Randy Wang. PlanetSeer: Internet Path Failure Monitoring and Characterization in Wide-Area Services. In *Proc. 6th USENIX OSDI*, pages 167–182, San Francisco, CA, December 2004.

[180] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John D. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications (J-SAC)*, 22(1), January 2004.