Resilient Overlay Networks

by

David G. Andersen

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

Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2001

© Massachusetts Institute of Technology 2001. All rights reserved.

Author . . . .
Department of Electrical Engineering and Computer Science
May 16, 2001

Certified by . . . .
Hari Balakrishnan
Assistant Professor
Thesis Supervisor

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

Resilient Overlay Networks

by

David G. Andersen

Submitted to the Department of Electrical Engineering and Computer Science
on May 16, 2001, in partial fulfillment of the
requirements for the degree of

Master of Science in Computer Science and Engineering

Abstract

This thesis describes the design, implementation, and evaluation of a Resilient Overlay Network (RON), an architecture that allows end-to-end communication across the wide-area Internet to detect and recover from path outages and periods of degraded performance within several seconds. A RON is an application-layer overlay on top of the existing Internet routing substrate. The overlay nodes monitor the liveness and quality of the Internet paths among themselves, and they use this information to decide whether to route packets directly over the Internet or by way of other RON nodes, optimizing application-specific routing metrics.

We demonstrate the potential benefits of RON by deploying and measuring a working RON with nodes at thirteen sites scattered widely over the Internet. Over a 71-hour sampling period in March 2001, there were 32 significant outages lasting over thirty minutes each, between the 156 communicating pairs of RON nodes. RON's routing mechanism was able to detect and recover around *all* of them, showing that there is, in fact, physical path redundancy in the underlying Internet in many cases. RONs are also able to improve the loss rate, latency, or throughput perceived by data transfers; for example, about 1% of the transfers doubled their TCP throughput and 5% of our transfers saw their loss rate reduced by 5% in absolute terms. These improvements, particularly in the area of fault detection and recovery, demonstrate the benefits of moving some of the control over routing into the hands of end-systems.

Thesis Supervisor: Hari Balakrishnan
Title: Assistant Professor

To my mother, whose unparalleled chicken stock expertise was invaluable.

Contents

List of Figures

List of Tables

These are just a few of the treasures for the stockpot — that magic source from which comes the telling character of the cuisine.

- Marion Rombauer Becker, The Joy of Cooking

Art is I, science is we.

- Claude Bernard

Acknowledgments

Emerging from my first two years at MIT with my sanity, and a completed thesis, is not something I could have completed alone. I am deeply indebted to many people for their support and patience while I paced, grumbled, bounced, went climbing and running, and even occasionally got work done. The list below doesn't stand a chance of being comprehensive, but I'll give it a try.

I must first thank my advisor, Hari Balakrishnan, for his expert guidance and support. His vision, excitement, and intelligence are an inspiration and an example toward which I strive. Working with Hari, and in his research group, has helped me grow academically in ways I hadn't even realized I needed when I started—and as a bonus, it's been terribly fun.

Frans Kaashoek and Robert Morris generously let me draw ideas, insight, and experience from them and the PDOS group, and they provided considerable, appreciated mentorship while I worked on this research. I thank Hari, Frans, and Robert for building the bridges between their research groups that allowed me to learn from the experience of both. Dorothy Curtis, Stephen Garland, and John Guttag provided much welcome feedback and advice, both technical and otherwise.

My officemates, Alex Snoeren and Allen Miu, have revealed great depths of patience while I empirically examined novel mechanisms for debugging and optimizing programs—pacing, bouncing large yellow chair-balls, and having loud conversations with said programs. More seriously, they've provided friendship, conversation, and feedback about a huge number of topics, along with a fun and stimulating environment in which to work. I'm indebted to Alex for showing me the ropes at MIT when I arrived looking confused, and for working with me on several projects.

I would like to thank the members of the Networks and Mobile Systems (NMS) and Parallel and Distributed Operating Systems (PDOS) research groups. From architecture to art, and proxies to philosophy, there was always someone with a clue to talk to, or something fun with which to distract ourselves. Allen, Alex, Benjie, Bodhi, Bryan, Chuck, Dan, D.M., Deepak, Doug, Eddie, Emma, Emmett, Frank, Godfrey, Greg, JJ, Jaeyeon, Jinyang, Kevin, Magda, Nick – thank you all.

My friends from the MIT Outing Club, LCS, and MIT provide the rich and distracting tapestry that my life needs in order to function properly. Thank you all; I'd have stumbled by now without you and your friendship through moods, all-night paper-writing sessions, and the other non-scholastic ingredients that've gone into my research.

I thank the people who have hosted RON nodes for our evaluation testbed. Numerous people have contributed their time and energy towards helping us evaluate our system, and I gratefully acknowledge their assistance.

John Carter, Wilson Hseih, and Jay Lepreau at the University of Utah helped set my feet down the path of systems research, and I'm delighted they did so. Thank you!

My mom, Mary Lou Godbe, has assisted me enormously as I adapted to life, to life at MIT, and to being a graduate student. I couldn't have made it without her; nor can I adequately express the depth of my appreciation for her support and love.

A man does not know what he is saying until he knows what he is not saying.

- G. K. Chesterton

Chapter 1

Introduction

This thesis proposes *Resilient Overlay Networks* (RONs) as a technique to make distributed applications more reliable. In our approach, each application layers a "resilient overlay network" over the underlying Internet routing substrate. The nodes comprising a RON reside in multiple autonomous systems (ASes), and cooperate with each other to forward data on behalf of any pair of communicating nodes in the RON. Because ASes are independently administrated and configured, they generally fail independently of each other. As a result, if the underlying topology has physical path redundancy, it is often possible for RONs to find paths between RON nodes, even if Internet routing protocols cannot.

1.1 The Internet and Overlay Networks

The Internet is organized as independently operating autonomous systems (ASes) that peer together. In this architecture, detailed routing information is maintained only within a single AS and its constituent networks, usually operated by some network service provider. The information that is shared with other providers and ASes is heavily filtered and summarized using the Border Gateway Protocol (BGP-4) running at the border routers between ASes [43], which allows the Internet to scale to millions of networks.

This wide-area routing scalability comes at the cost of the reduced reliability of end-to-end communication between Internet hosts. This cost arises because BGP scales by greatly limiting the number of network links (and therefore, paths), and the amount of information about those links, considered by the protocol running at each border router in the Internet. BGP's fault recovery mechanisms sometimes take many minutes before routes converge consistently [26], and there are times when path outages even lead to significant disruptions in communication [6, 36, 37]. The result is that today's Internet is easily vulnerable to link failures, router faults, configuration errors, and malice—hardly a week goes by without some serious problem affecting the connectivity provided by one or more Internet Service Providers (ISPs) [32].

An overlay network is a "virtual" network created on top of an existing network. The nodes in the overlay network use each other as routers to send data. The connections between the overlay nodes are carried over the underlying network; in this case, the Internet. Overlay networks have the potential to improve the reliability perceived by applications because they permit aggressive path exploration and maintenance. The Internet is less aggressive, but scales well; in contrast, RONs can be more aggressive by deliberately limiting the size of any given RON to between two and fifty nodes. This size limitation also allows a RON to embed application-specific choices and constraints in its path selection. As a result, RONs can be used in a variety of ways. A multimedia conferencing program may link directly against the RON library, transparently forming an overlay between all participants in the conference, and using loss rates, delay jitter, or application-observed throughput as metrics by which to choose paths. An administrator may wish to use a RON-based router application to form an overlay network between multiple LANs as an "Overlay VPN." This idea can be extended further to develop an "Overlay ISP," formed by linking (via RON) points of presence in different traditional ISPs after buying bandwidth from them. Using RON's routing machinery, an Overlay ISP could provide a more reliable and failure-resistant Internet service to its customers.

Nodes in a RON self-configure into an overlay network by exchanging packets through the underlying Internet. An N-node RON has $N(N-1)$ uni-directional "virtual links" available among its nodes, of which it uses a subset to maintain connectedness and exploit the underlying network's redundancy. A RON node detects packets destined for another, encapsulates those packets, and routes them across the overlay through one or more other RON nodes until it reaches the intended destination.

RON nodes introspectively observe and constantly monitor the quality of the Internet paths between them, picking the path that has the best quality. If the underlying Internet path is the best one, that route is used and no other RON node is involved in the forwarding path. If the Internet path is not the best one, the RON will route through a neighbor. RON nodes exchange the quality of the inter-node virtual links via a routing protocol. The routing protocol selects paths using a variety of metrics, including available bandwidth, latency, and packet loss rate. Each RON node obtains

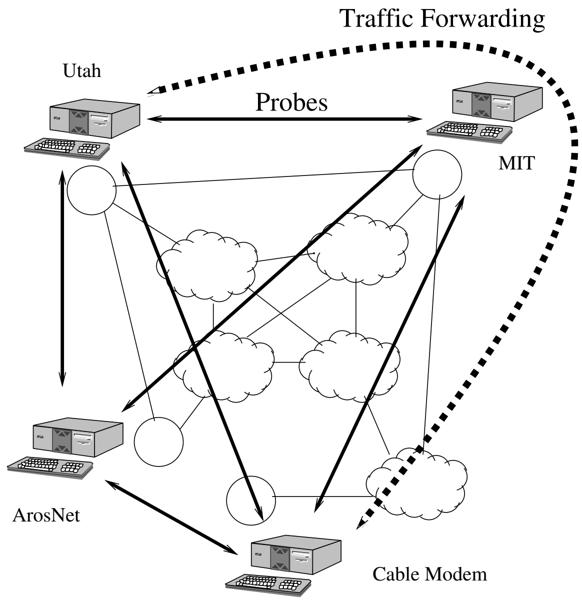


Figure 1-1: The general approach used in the RON system. Nodes send probes to determine the network characteristics between each other. Using their knowledge of the network, they potentially route traffic through other nodes. In this example, traffic from Utah to the Cable Modem site is sent indirectly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

the path metrics using a combination of active probing experiments and passive observations of on-going data transfers. RON also allows application-defined path selection metrics. An overview of the RON architecture in operation is shown in Figure 1-1

By pushing routing control further toward Internet end-points, or even directly to the distributed application, the RON architecture achieves four significant benefits:

1. Fault detection: A RON can more efficiently find alternate paths around problems even when the underlying network layer incorrectly believes that all is well.

2. Better reliability for applications: Each RON can have an independent, application-specific definition of what constitutes a fault.

3. Better performance: A RON's limited size allows it to use more aggressive path computation algorithms than the Internet. RON nodes can exchange more complete topologies, collect more detailed link quality metrics, execute more complex routing algorithms, and respond more quickly to change.

4. Application-specific routing: Distributed applications can link with the RON library and

choose, or even define, their own routing metrics.

In addition to presenting a design and implementation of RON, a significant contribution of this thesis is an evaluation of whether RON is a good idea at all. It is not obvious that it is possible to take advantage of underlying Internet path redundancy on the time-scales of a few seconds, reacting to path outages and performance failures. To answer this question and to demonstrate the benefits of RON, we have deployed a RON at nodes sprinkled across the Internet. Our current deployment has thirteen nodes located in universities, research labs, and companies across North America and Europe, over which we run a reliable IP forwarder RON client.

We have collected a few weeks' worth of experimental results of path outages and performance failures and present a detailed analysis of a few days' worth of data. Over a 71-hour sampling period in March 2001, we observed eight major outages lasting over thirty minutes each on some of the 156 communication paths in the RON. RON's routing mechanism was able to detect and recover around *all of them*, showing that there is in fact physical path redundancy in the underlying Internet in many cases. Our analysis shows that RON also improves end-to-end application performance by allowing more aggressive path computations: we found several host-host paths where RON improved the loss rate, latency, or throughput by a factor of two or more by routing along measured metrics rather than simply by hop counts. These improvements, particularly in the area of fault detection and recovery, demonstrate the benefits of moving some of the control over routing into the hands of end-systems where RON software runs. For this client, we find that RON routing works well by only considering one intermediate hop, rather than a general (but more complicated) shortest-paths path selector.

1.2 Contributions

In this thesis, we examine several causes of unreliability, and present and evaluate an architecture to work around some of these causes, providing increased reliability to distributed applications. The contributions of this thesis follow two major themes:

- A flexible architecture for application-level routing. We provide a set of libraries that programmers can use to add application-level routing capabilities to their programs, that researchers can use to conduct further networking research, and that users can use to obtain better reliability for their communications.

- Empirical validation of the effectiveness of indirect routing. Through our deployed network of RON nodes, we provide strong empirical evidence that routing data through a single indirect hop can improve the reliability and speed of Internet communication.

In Chapter 2, we provide background information about internetworking, and related work in Internet measurements and overlay networks. In Chapter 3, we examine some of the reasons that the Internet is unable to react as quickly to failures as we might hope. Building upon this basis, Chapter 4 presents the design and implementation of a system to automatically route around Internet failures faster than the Internet itself can. Chapter 5 shows that our approach is successful much of the time by examining measurement data from RON nodes deployed throughout the Internet. Finally, Chapter 6 concludes with a discussion of some of the exciting future research, both technical and otherwise, that RON enables, and provides a brief summary of the contributions of RON thus far.

I have but one lamp by which my feet are guided, and that is the lamp of experience. I know no way of judging the future but by the past.

- Patrick Henry, 1775

Chapter 2

Background and Related Work

In this chapter, we first provide background information about the Internet, and how traditional Internet backbone routing works, as a preface for understanding how RON is able to provide better paths using the Internet. Next, we discuss several other networks and projects that have used Internet overlay techniques for other purposes. Finally, we look at other Internet measurement systems and tools, to see the lessons they can contribute to a measurement-based overlay routing system.

2.1 Internet Routing

2.1.1 The Organization of the Internet

The Internet is a collection of individual networks that share a common, underlying communications protocol: the Internet Protocol (IP). Typically, networks are owned and operated by different organizations. Individuals and smaller companies usually attach to the Internet via an Internet Service Provider (ISP) such as AOL, EarthLink, or an ISP local to their city or state. Home users in the US typically connect via modems (33.6 or 56kbps), Digital Subscriber Lines (DSL, 128kbps - 1Mbps), or cable modems (128Kbps - 2Mbps). Corporations, universities, and and service providers attach in a similar manner, though frequently with higher speed T1 (1.5Mbps) or T3 (45Mbps) links. Larger corporations and many universities may connect to multiple ISPs.

A simplified view of the way many people connect to today's Internet is shown in figure 2-1. The Internet has no "backbone," per se, but it does have a collection of many large ISPs who together handle a large fraction of the traffic on the Internet. These large ISPs are often referred to as the "Tier 1" providers. The Tier 1 providers are typically national or international in scope, and

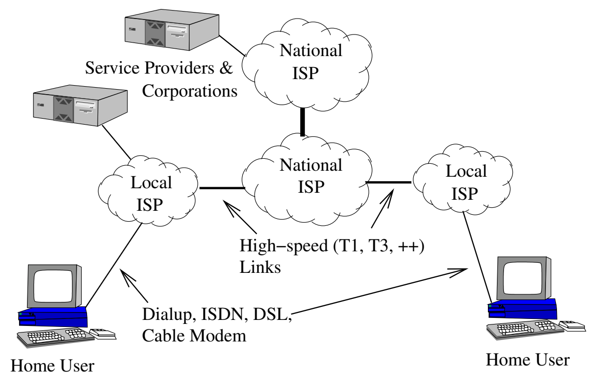


Figure 2-1: An example of some interconnected sites on the Internet, showing some of the networks that connect to each other to provide seamless connections between users.

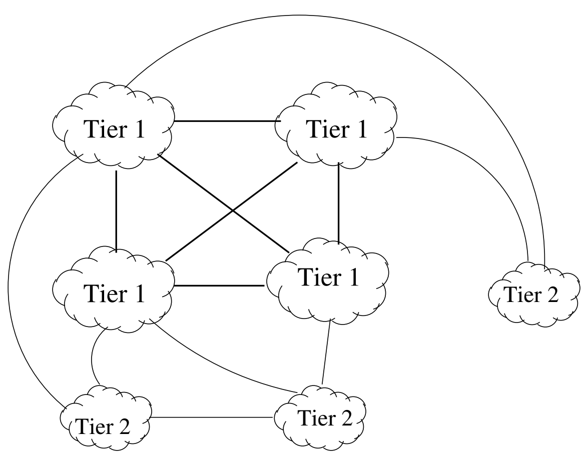


Figure 2-2: ISPs are often classified as "Tier 1", 2, or 3 providers. Tier 1 providers have national or international scope, and can provide Internet connectivity nearly everywhere. Tier two providers typically cover large regional areas, such as the west coast of the US. Tier three providers are smaller, local ISPs. ISPs typically connect to several other ISPs, not just one large upstream provider, creating a diverse mesh of interconnections between sites in the Internet.

can provide high-speed Internet connectivity nearly everywhere. The Tier 1 providers never buy Internet connectivity from anyone. Instead, they all have mutual agreements to share data with each other. Tier 2 providers typically serve specific regions, and may buy connectivity from some Tier 1 providers. However, many Tier 2 providers also have free data-sharing agreements with each other. Drawing the line between a Tier 1 and a Tier 2 provider is a fuzzy thing, though many ISPs fall solidly into one category or another.

Because the Internet is composed of many discrete networks, ISPs must provide each other with routing information, of the form, "You can reach network X through me." A connection to another ISP where the two ISPs exchange this reachability information is termed *peering*. ISPs use the Border Gateway Protocol (BGP) version 4 to perform interprovider routing.

The Internet is a mesh of interconnected networks, not a hierarchical system. ISPs connect to many other ISPs, and large customers may even "multi-home" by connecting their networks to multiple ISPs. In a mesh, there are many possible paths between any two end sites, a fact we use to our advantage in RON.

2.1.2 Routing

Early ARPANET Routing

Early ARPANET routing was more dynamic than today's BGP-based Internet routing, responding to the delay and utilization of the network in making its routing decisions. This routing, circa 1989, used a delay and congestion-based distributed shortest path routing algorithm [23]. It underwent several revisions to deal with problems of oscillation and computing loop-free paths. In 1989, the algorithm was changed to incorporate an exponential weighted moving average (though it was not termed as such) longer-term average utilization metric to reduce rapid oscillations. The resulting algorithm propagated link costs around the network by combining a "base" value for a link, which was relative to its capacity and whether it went over a (presumably fast) land link or a slower satellite link, with a dynamically computed utilization metric.

The 1989 algorithm was tested with 1987 topology data, at which point the Internet had just passed about 10,000 hosts, with under 1000 constituent networks, and was still operating with 56 and 9.6kbps backbone links. The original ARPANET was managed by a single administrative entity, reducing the need for exterior routing protocols. 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, interior routing within systems is still sometimes performed in an ARPANET Shortest Path First (SPF)-like fashion using OSPF or other protocols, but routing *between* systems, is performed using a different protocol: BGP.

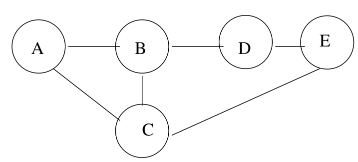
BGP

BGP version 4 is *the* backbone Internet routing protocol today. While different providers may use different interior routing protocols within their own networks (OSPF, EIGRP, IBGP, etc.), Inter-domain routing is handled primarily with BGP. While an in-depth description of BGP is beyond the scope of this thesis, we provide a brief overview.

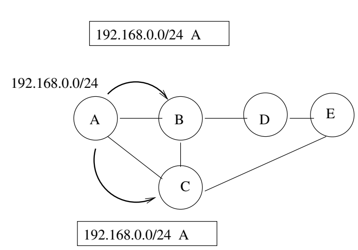
BGP relies on the idea of an *Autonomous System* (AS), an independently organized and operating network or collection of networks. Each AS has a unique Autonomous System Number (ASN). ASs range from huge network providers like MCI (AS 3561) and BBN (AS 1), to institutions like MIT (AS 3), to small companies. In fact, in the earlier days of the Internet, even some individuals could receive a registered autonomous system number for their personal networks.

ASs 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, all sites know how to get to other sites on the network.

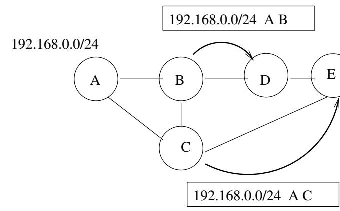
To illustrate the behavior of BGP, we consider a simple network consisting of 5 different ASs:



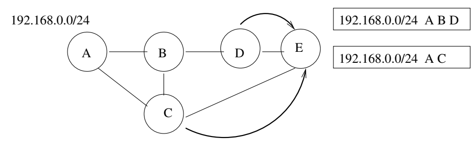
BGP is a "path vector" routing protocol. A BGP route announcement begins with a system, A, saying "I have network 1." A sends this announcement to all of its peers, in this case, B and C. For the purposes of this example, we will examine the announcement of the netblock `192.168.0/24` as it travels from A onward. The protocol begins with A informing B and C that they can reach this netblock through A:



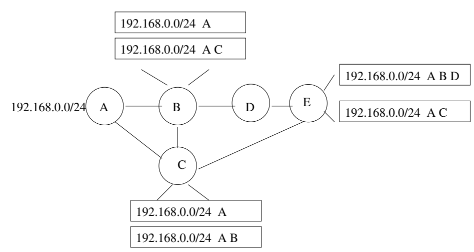
Systems B and C receive this announcement, and append their *own* AS to the announcement, and send it out again. In the case of System B, it tells system D "You can reach `192.168.0/24` through me, system B."



This process continues with system D re-sending the announcement. In reality, system E would send a similar message to system D, but we've omitted it for clarity:



After the algorithm has run for a sufficient period of time, the BGP tables at sites B, C, and E, for the example netblock, will look like:



It is noteworthy that nodes do not share *all* the routes they see with their peers, but simply the best route. By supplying the list of systems through which a path goes, BGP attempts to avoid loops in its route construction. This is effective over time, though as we discuss in Chapter 3, BGP may encounter transient loops during its route updates.

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 10,000 customers are located, a large ISP instead sends out a small routing announcement for tens to hundreds of netblocks saying simply, "You can reach anything in this network block through me." While this aggregation is necessary for the scalability of the global Internet, it does interfere with the use of alternate routes, a problem we explore further in Chapter 3, and which we exploit to construct better routes with RON.

2.2 Overlay Networks

The idea of overlay networks is not a new one. The Internet began its life as a data network overlaid on the public telephone network, and even today, a large number of Internet connections continue to take place over modem lines. As the Internet itself grew in popularity, many people began using the Internet as a substrate for other networking research. Examples of networks commonly overlaid on the Internet include the MBone for extending multicast functionality across areas where the Internet did not natively support multicast, and the 6-bone for testing the deployment of IPv6.[1]

2.2.1 MBone

The MBone is one of the most well-known large overlay networks deployed on the Internet [13]. The MBone creates virtual "tunnels" over the Internet to connect networks that support native IP multicast, enabling a global multicast architecture. MBone tunnels are configured statically by administrators. While the intent of MBone connectivity is to connect entire networks via a single MBone tunnel, or a small set of tunnels, it is possible to connect a single workstation to the MBone in the same manner. Except when network partitions occur, there is *one* global MBone, though its overlay techniques are more general.

2.2.2 6-bone

Like the MBone did for IP multicast, the 6-bone was designed to facilitate the deployment and testing of IP version 6 [18]. It is a global network of connected IPv6-supporting sites. In this network, some sites connect via native IPv6-speaking links, and sites without native IPv6 upstream links connect via tunnels configured over the IPv4 Internet. The use of the 6bone has greatly facilitated the task of designing and testing IPv6-aware software and protocols [16].

2.2.3 The X-Bone

The X-Bone [48] is an infrastructure project designed to speed the deployment of IP-based overlay networks like the MBone [24]. It provides a graphical user interface for automated configuration of endpoint IP addresses and DNS names, simple overlay routing configurations like rings, and

[1]Overlay networks were used long before the MBone, however. One early use as an explicit overlay network was RFC 1070, which proposed the use of the Internet as a substrate upon which researchers could experiment with the OSI network layer [17]. UUCP bang-path addressing [19] and 1982's SMTP mail routing [11] could be considered a form of application-layer overlays.

allows remote maintenance of the overlays via SSL-encrypted HTTP sessions. The X-Bone does not yet support fault-tolerant operation or metric-based route optimization. Its address and DNS management functions, and its mechanisms for handling packet insertion into the overlay, are complementary to our work.

2.2.4 Yoid / Yallcast

Yoid provides a "general architecture for all Internet distribution" [15]. It attempts to unify the benefits of native IP multicast (efficiency for local communicating groups) with the deployment and implementation advantages of end-system-based multicast. Yoid's core protocols generate a tree topology, for efficient content distribution, and a mesh topology, for robust content distribution. Yoid's design calls for integrated congestion management in the overlay network links by using TCP or RTP between the nodes. Yoid is designed to provide a "better" (or perhaps simply a realizable), flexible multicast architecture. Yoid is closely related to its predecessor project, Yallcast. As of this time, Yoid (like Yallcast) appears to still be in the design stage, with a pending software release.

2.2.5 End System Multicast

End-system multicast provides application-based multicast services as an overlay on the unicast Internet [20]. It is designed for many-to-many multicast for groups of moderate size (tens to hundreds). In the designer's Internet measurements, taken primarily on Internet2-connected hosts, they report a small number of links in which the "Relative Delay Penalty," the increase in delay seen by applications because of the use of the overlay, is less than one. By looking more aggressively at smaller groups, RON locates more of these situations for optimizing unicast performance.

2.2.6 ALMI

ALMI, an Application Level Multicast Infrastructure, provides many-to-many multicast for small groups (tens of nodes) [41]. Unlike End-system multicast and Yoid, ALMI uses a centralized scheduler to compute its multicast distribution trees, which reduces the complexity of the tree-building algorithms, but necessitates a centralized controller. The initial evaluation of ALMI appears promising, though the evaluation was somewhat limited: The 9 sites in their evaluation were located in 8 different domains, and all of the US sites were on the vBNS. The experiments to evaluate ALMI ran for 8 hours, with no apparent large network faults. In the evaluation, ALMI's tree generation algorithm—like RON's—used the full mesh of nodes when building its distribution tree.

2.2.7 Overcast

Overcast is an application-level overlay multicast system that provides scalable, reliable single-source multicast for large numbers of clients [22]. Overcast is targeted at the dissemination of non-realtime audiovisual content as a part of Cisco's streaming content delivery solutions. Overcast focuses on scalable tree-building and repair mechanisms.

2.2.8 Content Delivery Networks

Content Delivery Networks (CDNs) such as Akamai [49], and Inktomi [50] use overlay techniques and caching to improve the performance of content delivery for specific applications such as HTTP and streaming video. The functionality provided by RON may ease future CDN development by providing some basic routing components required by these services.

2.2.9 Peer-to-peer Networks

Many peer-to-peer networking architectures use overlay-style functionality for file transfers, network configuration, or searching. Freenet creates an application-layer overlay that routes file transfers and search requests between participants in its protocol [9]. Freenet uses randomized connections between Freenet nodes, not considering the network characteristics between them, because a major function of Freenet's message routing is to provide anonymity to both publishers and consumers of data. Gnutella[2] dynamically creates an overlay network by connecting each participant to roughly 4 other participants in the network. Gnutella broadcasts search requests and replies over the overlay, and then handles file transfers directly over the Internet.

2.3 Performance Measuring Systems and Tools

RON is a measurement-based system. It takes on-going measurements of the network, and bases its routing decisions on the results of these measurements. RON needs sources of data—performance measuring tools. There are a number of existing systems and tools that measure packet loss, latency,

[2]Few formal papers have been published about Gnutella. It is officially distributed at `http://gnutella.wego.com/`. `http://www.limewire.com/article.htm` has an online article describing and comparing several p2p systems, including Gnutella.

and bandwidth on Internet paths. Some are designed from a purely measurement-based viewpoint (e.g. NIMI), others are designed as complete systems that use the data they measure (e.g. SPAND), and others are purely tools to measure particular path properties. Regardless, we incorporate many lessons from these systems into the design of RON, and briefly detail some of them here.

2.3.1 NIMI

The Detour and Paxson studies both used traceroutes scheduled from a centralized server to determine Internet path properties. Centralized scheduling undercounts network outages, because it is unable to schedule measurements when the scheduler is disconnected from the measuring hosts. This lesson is an important one that is incorporated into the RON performance database: measurements are performed independent of a centralized scheduling authority. These same lessons were incorporated into the more recent NIMI measurement architecture [39]. In comparison to RON's measurement system, NIMI is a more complex and comprehensive measurement architecture, designed to support completely heterogenous servers with different configurations, administrators, and network policies. To this end, NIMI's focus is on the scheduling and authentication of measurement requests [40].

2.3.2 IDMaps

IDMaps, the Internet Distance Maps project, is designed to answer queries like, "What is the latency between host A and host B?" Where RON uses constant active measurements to obtain near-realtime measurements of the latency between a small group of hosts, IDMaps attempts to provide latency answers for huge groups of hosts, accurate to within a few hours. To scale, the IDMaps programs disseminate the distance information in the form of host to host latencies for a subset of the hosts, and end hosts run a spanning tree algorithm over the host to host latency to estimate the real distances between a pair of hosts.

2.3.3 SPAND

SPAND [46] uses passive measurements to store the performance of Internet transfers in a repository. When the same content can be obtained from several sites, users can query the shared repository to get an estimate of which site will give them the best performance. We leverage SPAND's ideas of a shared performance database and application-provided network data. We additionally incorporate active probing to maintain information about alternate links to survive network failures.

2.3.4 Detour

The Detour study made several observations of suboptimal Internet routing [45]. Their study of traceroute-based measurements and post-analysis of Paxson's data [36, 37] shows frequent occasions on which alternate paths may have superior latency and loss rates.

The Detour framework [10] is an in-kernel packet encapsulation and routing architecture designed to support alternate-hop routing, with an emphasis on high performance packet classification and routing. It uses IP-in-IP encapsulation for sending packets along the alternate routes, and provides its flow classification and flow database as an in-kernel data structure for high performance. In contrast to the application-independent Detour Framework, we believe that tighter integration of the application and the overlay network is important for functionality and deployment: it permits "pure application" overlays with no kernel modifications, and it allows the use of application-defined quality metrics and routing decisions. However, the RON forwarding design does not *preclude* the development of an optimized in-kernel forwarding engine.

2.3.5 Commercial Measurement Projects

The Internet Weather Report (IWR) [29] and its commercial counterpart, Matrix.Net, use a variety of probing boxes scattered around the Internet to perform probes, we believe, of latency, loss, and reachability, similar to the probes used by RON for constructing its overlay. IWR sends its pings to thousands of external hosts, every 15 minutes, and attempts to provide a larger picture of the connectivity of the Internet. RON operates closer to realtime, on the order of 15 *seconds*, but for a far smaller fully connected group of cooperating hosts.

Keynote Systems provides a distributed website monitoring and reporting service [51]. They have a large deployed infrastructure of measurement hosts that periodically make requests to client webservers and perform page views, insecure and secure transactions, and downloads of streaming media. Keynote's tests are based upon statistical customer profiles designed to estimate a "typical" user's experience with the client's site, and highlight potential performance problems with the site. Keynote provides application and site-specific monitoring. It appears that, like IWR, Keynote monitors roughly every 15 minutes.

2.3.6 Latency and Loss Probe Tools

We provide only a short list of representative programs here; interested readers should see the CAIDA tools taxonomy for more information [4].

Latency and loss samples can be obtained using the familiar `ping` utility, which sends ICMP echo requests to a remote host and waits for the reply. This functionality can also be implemented in UDP, using the `echo` service, and can even be gleaned from TCP by carefully watching the returning ACK stream, as is done in Sting [44]. Basic ICMP pings only determine one-way loss; Sting, or userlevel loss probing mechanisms, can determine the loss in each direction. Without synchronized clocks, programs can only determine round-trip latency.

2.3.7 Available Bandwidth Probe Tools

The most reliable mechanism for measuring the bandwidth available to a TCP flow is to actually attempt to use the bandwidth, which is what the `ttcp` utility does [7]. The `cprobe` utility attempts to measure the available bandwidth by using small trains of packets, instead of full-bore connections [5]. The `TReno` program uses ICMP or high-numbered UDP packets to measure the available bandwidth along each link in the path between two hosts.

2.3.8 Link Bandwidth Probe Tools

`Nettimer` [27] provides both sender and receiver–based bottleneck link bandwidth estimation, and, as of this writing, appears to provide the best balance of accuracy and number of packets required to sample. `Pathchar` [21] uses similar techniques to measure the link bandwidth of each hop between sender and receiver, a distinction that is excessive when considering the end-to-end path between two hosts. `Nettimer` and `Pathchar` measure the arrival intervals of back-to-back packets; this technique is also used in earlier programs like `bing` [2] and `bprobe` [5].

2.3.9 Standards for Tools

The IETF IPPM Working Group is in the process of standardizing a framework for Internet performance metrics and tools [38]. Though their tools were not designed for use in an on-line routing system, their work provides guidance for the development of tools suitable for use in RON.

It is the theory which decided what can be observed.

- Albert Einstein

Chapter 3

Problems with Internet Reliability

Today's wide-area Internet routing system, based on BGP4, does not handle failures well. From a network perspective, there are two broad kinds of failures. *Link failures* are caused when a router or a link connecting two routers fails because of a software error, hardware problem, or link disconnection. *Path failures* are caused for a variety of reasons, including denial-of-service attacks or other spurts of traffic that cause a high degree of packet loss or high, variable latencies.

Applications perceive all failures in one of two ways: *outages* or *performance failures*. Link failures and extreme path failures cause outages, when the average packet loss rate over a sustained period of several minutes is high (about 30% or higher), degrading the performance of most protocols, including TCP, by several orders of magnitude. Performance failures are less extreme; for example, throughput, latency, or loss-rates might degrade by a factor of two or three.

Wide-area IP routing suffers from four important drawbacks:

1. *Slow link failure recovery:* BGP4 takes a long time (on the order of several minutes) to converge to a new valid route after a router or link failure causes a path outage [26].

2. *Inability to detect path or performance failures:* BGP4 cannot detect many problems (floods, persistent congestion, etc.) that can greatly degrade performance. As long as a link is deemed "live" (i.e., the BGP session is still alive), BGP's AS-path-based routing will continue to route packets down the faulty path.

3. *Inability to effectively multi-home end-customer networks:* An oft-cited "solution" to network unreliability is to multi-home a network by connecting it to several different network providers. Unfortunately, peering at the network level by small customers (as opposed to second- or third-tier ISPs) compromises wide-area routing scalability, and is discouraged.

4. *Blunt policy expression:* BGP is incapable of expressing fine-grained policies aimed at users or remote hosts; it can only express policies at the granularity of entire remote networks. This reduces the set of paths available in the case of a failure.

3.1 Slow link failure recovery

Labovitz *et al.* [26] use a combination of measurement and analysis to show that inter-domain routers in the Internet may take tens of minutes to reach a consistent view of the network topology after a fault. The reason for this has to do with routing table oscillations that occur during the operation of BGP's rather complicated path selection process. They find that during this period of "delayed convergence," end-to-end communication is adversely affected. In fact, outages on the order of minutes cause active TCP connections[1] to terminate when TCP does not receive an acknowledgment for its outstanding data. They also find that, while part of the convergence delays can be fixed with some changes to the deployed BGP implementations, long delays and temporary oscillations are a fundamental consequence of the BGP path vector routing protocol.

Paxson finds using probe experiments that routing pathologies prevent selected Internet hosts from communicating up to 3.3% of the time averaged over a long time period, and that this percentage has not improved with time [36]. Labovitz *et al.* find, by examining routing table logs at Internet backbones, that 10% of all considered routes were available less than 95% of the time, and that less than 35% of all routes had an availability of higher than 99.99% [25]. Furthermore, they find that about 40% of all path outages take more than 30 minutes to repair and are heavy-tailed in their duration. More recently, Chandra *et al.* find using active probing that 5% of all detected failures last more than 10,000 seconds (2 hours, 45 minutes), and that failure durations are heavy-tailed and can last for as long as 100,000 seconds before being repaired [6]. These findings do not augur well for mission-critical services that require a higher degree of end-to-end communication availability. Table 3.1 provides a brief summary of these findings.

3.2 Inability to detect performance failures

Wide-area routing mechanisms do not adequately handle performance faults that can greatly degrade the quality of end-to-end communication between applications. For example, consider a link that

[1] An active TCP connection is one in the `ESTABLISHED` state *and* that has outstanding data that one or both sides is trying to send.

Paxson	Serious routing pathology rate, 1995: 3.3%
Labovitz	10% of routes available less than 95% of the time
Labovitz	Less than 35% of routes available 99.99% of the time
Labovitz	40% of path outages take 30+ minutes to repair
Chandra	5% of faults last more than 2 hours, 45 minutes

Table 3.1: Internet link failure recovery times, as reported by several studies.

is excessively loaded because of legitimate traffic or a denial-of-service attack. BGP is generally unaware of this, and even if an alternate path exists, it will not use it. This lack of path diversity makes it hard to handle performance failures, and leads to serious problems.

Conventional inter-domain IP-layer routing is application-independent; it does not generally allow route selection based on the nature of the application. While some research projects like FIRE [35] provide flexible intra-domain routing, the problem is considerably harder for inter-domain routing. As a result, applications running over the Internet are not able to influence the choice of paths to best suit their latency, loss, or bandwidth requirements.

3.3 Inability to effectively multi-home

The IP routing architecture depends on heavy aggregation of the addresses announced into the system. A small- or medium-sized organization that wishes to obtain better Internet service might purchase service from two different ISPs, hoping that an outage to one would leave it connected via the other. Unfortunately, to limit the size of their routing tables, many ISPs will not accept routing announcements for fewer than 4096 contiguous addresses (a "/20" netblock) [2] Small companies, regardless of their reliability needs, may not even require 256 addresses, 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 its hosts. While provider-based addressing can provide long-term fail-over if one ISP connection fails, it is unclear how on-going connections on one address set can seamlessly switch on a failure in this model.

[2] The size of the announcements providers accept fluctuates. Nearly all providers guarantee that they will accept /20 announcements. Depending on router capacity and the number of prefixes currently in the backbone routing tables, providers may accept smaller announcements.

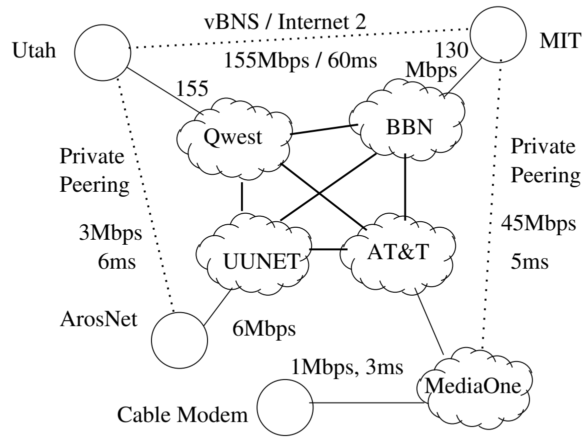


Figure 3-1: The complex financial arrangements governing the announcements of links on the Internet, combined with the blunt "yes or no" per-system announcement policies mandated by BGP, means that links may not be available to authorized users, despite the policies governing the use of those links. In this figure, the dotted links are *private* and are not announced globally.

3.4 Blunt policy expression

Figure 3-1 shows the AS-level network connectivity between a small subset of our measurement hosts; the full graph for only 12 hosts consists of 36 autonomous systems with hundreds of different network links. The figure gives a hint of the considerable underlying path redundancy available in the Internet—the reason RON works—and shows situations in which BGP's policy expression limits fail-over. For example, if the Aros-UUNET connection failed, a user at Aros would be unable to reach MIT—even if they were authorized to use Utah's network resources to get there. Without outside agreements, BGP is incapable of even expressing the policy, "I'll announce this network to you, but you can't announce it to anyone else." While some policies of this nature can be applied at the AS level by manually coordinating BGP filters with peering organizations, this is a fragile mechanism, and still incapable of expressing more complex policies like the authorized user case discussed earlier.

3.5 Defining Outages

The performance of a network can range from 100% delivery of packets within a reasonable timeframe, to perfect delivery of those same packets 8 seconds too late, or to a complete failure to deliver packets. The health of the network can be viewed as a continuum along the latency and loss

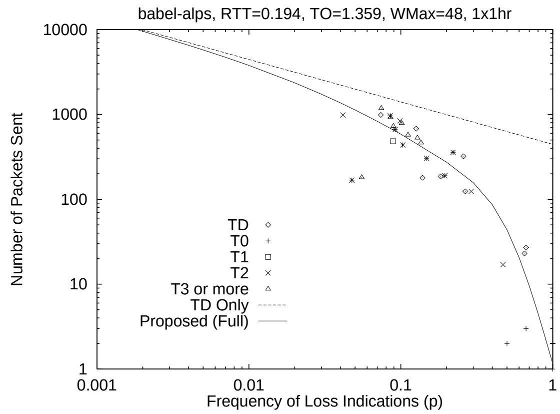


Figure 3-2: This figure, taken from Padhye et al.'s SIGCOMM'98 paper [33], shows a drastic knee in TCP throughput when the loss rate begins to exceed 30% packet loss. Note that both scales are logarithmic. (Figure reprinted with the permission of the author. It is copyright ©1998 Jitendra Padhye.)

axes; the question is, "What defines an outage?" Earlier work has framed applications by the way their utility curves vary in response to latency and loss [3]; while such a categorization is outside the scope of this work, where possible, we provide sufficient data that the reader can apply to our research the utility curve of whatever application is of interest.

In cases where we wish to look at a generic form of "outages," we look only at the loss rate. First, we note that a complete outage (100% packet loss) for several minutes will tear down active TCP connections: BSD-derived systems commonly time-out after 511 seconds, and some Solaris TCP stacks time out in as little as 120 seconds. Second, we believe that long-term packet loss rates of greater than 30% are sufficiently deleterious to the performance of most applications that this rate can be considered a "performance outage."

To understand this benchmark loss rate, Figure 3-2 shows a response curve of TCP throughput vs. loss obtained by Padhye et al. [33]. Their work provides evidence that the throughput obtained by TCP drops off quite drastically at high loss rates. Many of the cases they study, including Figure 3-2, exhibit a knee in the response curve at about 30% loss, suggesting that loss rates above this are extremely destructive for Internet communication.

3.6 Summary

Internet backbone routing is optimized for scalability and relative simplicity. These traits, which are in large part responsible for the Internet's ability to support its exponential growth rates of the last decade, come at the expense of fast outage detection and recovery. Compared to a highly reliable system like the public telephone network, the Internet is able to support an incredibly more rich set of interconnections, links, and protocols, but this same diversity in the underlying network suggests that for much time to come, the global Internet will *not* achieve the reliability levels needed for some applications.

Architecture begins when you place two bricks carefully *together.*

- Miles van der Rohe

Chapter 4

RON Design and Implementation

4.1 Design Overview

At a conceptual level, the design of RON, shown in Figure 4-1, is quite simple. RON nodes cooperatively route packets for each other to route around failures in the underlying Internet. This "network on top of the Internet" is an *overlay network*; and since, in the case of RON, we form this network between cooperating applications, we term RON 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 direct 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.

RON is a software library that programs link against. We call such programs *RON clients*. The overlay network is defined by a single group of clients that collaborate to provide a distributed service or application. This group of clients can use service-specific routing metrics to decide how to forward packets in the group. Our design accommodates a variety of RON clients, 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.

A RON client interacts with the RON libraries across an API called a *conduit*, which it uses to send and receive packets. On the data forwarding path, the first node that receives a packet via the conduit classifies the packet to determine the type of path on which it should be forwarded (e.g., low-latency, high-throughput, etc.). This node is called the *entry node*: it determines a path from

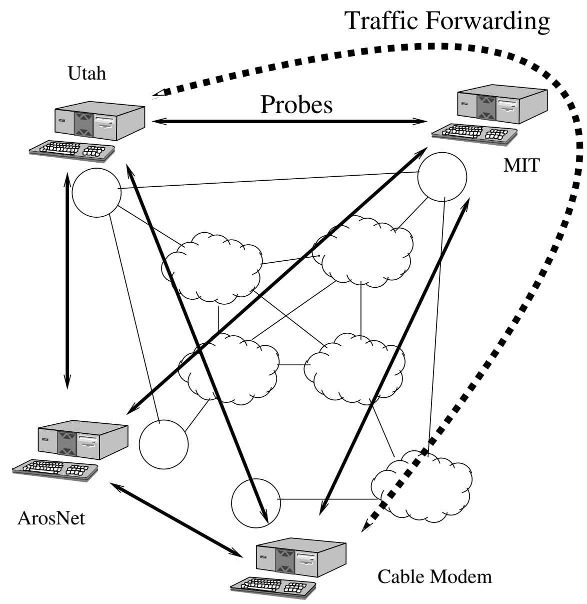


Figure 4-1: The general approach used in the RON system, also shown in Figure 1-1. Nodes send probes to determine the network characteristics between each other. Using their knowledge of the network, they potentially route traffic through other nodes. In this example, traffic from Utah to the Cable Modem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

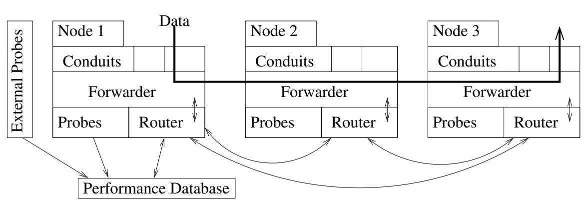


Figure 4-2: The RON system architecture. Data is inserted into the RON by RON clients via a *conduit* at an *entry node*. At each node, the *RON forwarder* consults with its *router* to determine the best path for the packet, and sends it to the next node. The selection of the path is done at the entry node, which also tags the packet, simplifying the forwarding path at all other nodes. When the packet reaches the RON *exit node*, the forwarder there hands it to the appropriate output conduit, whence it is passed on to the client. Path selection is facilitated by the RON nodes monitoring the quality of their virtual links using a combination of active probing and passive observation. RON nodes use a link-state routing protocol to disseminate the topology and virtual-link quality of the overlay network.

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 determines the next forwarding hop based on the destination address and the tag. The final RON node that gives the packet to the RON application is called the *exit node*. Figure 4-2 shows the overall RON system architecture.

Unlike traditional networks where each node selects a path independent of where the packet might previously have been, RON performs path selection primarily at the entry node. This node *tags* the packet with some information in the RON header that signals the chosen path to downstream RON nodes, thereby allowing the entry node to maintain control over the path and reducing the amount of processing required at the internal nodes. (This is akin to the way packet forwarding works with MPLS [12].) This control allows for a more predictable understanding of the resulting performance for the packet stream across the RON: unless a virtual link fails, the downstream node will not generally change a previously chosen path. Early path selection also means that downstream failures must be propagated to the entry node for proper re-routing to be done. RON accomplishes this by tightly constraining the number of RON nodes along any path, thereby realizing a simple routing strategy that overcomes path outages and performance failures in practice.

The small size of a RON relative to the Internet allows it to probe and explore paths more aggressively and to use a range of quality metrics, exchanging this information using a link-state routing protocol. It also facilitates the implementation of flexible policies that govern what types of packets may be forwarded, at what rates, and along which paths in the RON. We describe RON's topology dissemination, path monitoring and selection mechanisms, and enforcement of routing policy in Section 4.6.

4.2 System Model

A node joining an existing RON uses a bootstrap protocol to advertise itself to the other participating nodes. RON nodes also need to run a membership maintenance protocol to maintain up-to-date information about currently active nodes.

Client programs link against the RON library and interact with RON via a conduit. Conduits are a "translation service" between the client program's data and RON packets. They encapsulate packets to ship across the RON, and decapsulate received RON packets for delivery to the client. Data transfer across the RON uses a protocol that runs over UDP. The conduits access RON via two functions:

- `send(pkt, dst, via_ron)` allows a node to forward a packet to a destination RON node either along the RON or using the direct Internet path.
- `recv(pkt, via_ron)` is a callback function that is called when a packet arrives for the client program. 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.

The RON library also provides a timer registration and callback mechanism to perform periodic operations, and a similar service for network socket data availability.

The *RON forwarder* is the basic building block for all RON clients. Each client must instantiate a forwarder and must hand to it two modules: a *RON router* and a *RON membership manager*. We have developed two RON routers and two membership managers.

RON routers and membership managers exchange packets using RON as their forwarding service, rather than using direct IP paths. This feature of our system allows these messages to be forwarded even when some underlying direct IP paths fail.

4.2.1 Implementation

The RON system is implemented as a flexible set of C++ libraries to which applications link. The applications can pick and choose the components that best suit them. This presents us with an infrastructure to explore both the network benefits of RON, and the development of the control algorithms used within RON. The basic RON functionality is provided by the `forwarder`.

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):

```
register_type(int packet_type)
register_socket(int sockno)
register_timer(int msec)
```

The forwarder provides a basic interface for sending data to other RON nodes:

```
send_packet(packet, dst, int 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 as:

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

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

4.3 Bootstrap and Membership Management

In order to form a RON, the participating nodes need to know about each other. We allow RON clients to define their own node membership mechanisms, which may be useful for applications, such as Internet conferencing programs, that maintain their own membership lists. We also provide two system membership managers: a simple static membership mechanism that loads its nodes from a file, and a dynamic announcement-based, soft-state membership protocol.

To bootstrap the 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 to the other nodes already in the RON.

To facilitate such group communication, RON provides a generic flooding mechanism. The flooder is simply another RON client. It registers with the forwarder to receive `TYPE_FLOOD` packets. Each flood packet has a unique 64-bit identifier; recipients cache the received IDs to suppress duplicates, forward new packets to all of their neighbors except the originator of the packet, and then decapsulate the packet and hand it back to RON for further processing.

The main challenge in the dynamic membership protocol is to avoid confusing a path outage to a node with the node leaving the RON. Each node builds up and periodically *floods* to all other nodes its list of peer RON nodes. This is done every five minutes on average. If a node has not heard about a peer in sixty 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 arbitrarily long periods of time: as long as there is *some* path in the RON between the two nodes, neither will think that the other has left.

This membership mechanism is a trade-off between simplicity and robustness on the one hand, and potentially excessive overhead on the other. In practice, the overhead of broadcasting is minuscule compared to the traffic caused by active probing and routing updates, especially given the limited size of a RON. Membership announcements are robust: each node receives up to $N - 1$ copies of the peer list sent from a given other node. This degree of message 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.

To improve the chances of reconnecting to a RON network 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 it. It does not, however, add these nodes into its `peerlist` until it hears back from them.

4.3.1 Implementation

Membership managers provide one function: `peers()` which 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 as a RON conduit; it configures itself with the RON forwarder to receive membership packets and timer callbacks:

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

<table>
<tr><td>Version</td><td>Hop Limit</td><td>Routing Flags</td></tr>
<tr><td colspan="3">RON Source Address</td></tr>
<tr><td colspan="3">RON Destination Address</td></tr>
<tr><td colspan="2">Source port</td><td>Dest port</td></tr>
<tr><td colspan="3">Flow ID</td></tr>
<tr><td colspan="3">Policy Tag</td></tr>
<tr><td colspan="3">Packet Type</td></tr>
</table>

Figure 4-3: The RON packet header. The routing flags and flow ID are set by the input conduit. The packet type is a demux key to send the packet to the appropriate conduit or protocol at the receiver.

4.4 Data Forwarding

RON uses UDP to forward data, since TCP's reliable byte-stream is mismatched to many RON clients, and using IP-based encapsulation would restrict its application-specific forwarding capabilities. By avoiding the use of a congestion-controlled protocol, RON avoids interactions between multiple feedback-based control loops when carrying TCP data. The RON core services can run without special kernel support, and without requiring elevated privileges. The conduit at the entry node is the client's gateway to the RON, and it classifies every packet. This classification is client-specific; it labels the packet with information that decides what routing metric is later used to route the packet. As an example, a RON IP forwarder conduit might label all DNS and HTTP traffic as "latency-sensitive" and all FTP traffic as "throughput-intensive," which would cause the downstream 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 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 further forwarding within the RON is independent of client-specific logic. In addition to reducing the per-packet overhead at intermediate nodes, it also centralizes the client-specific computation required on each packet. This means, for example, that a RON node implementing an IP forwarder client may also participate in an overlay with a conferencing application, and help improve the reliability of data delivery for the conference. The conduits at the conference nodes will implement the application-specific path selection methods and label packets to tell the IP forwarder which routing metrics to use for conferencing packets.

The RON packet header is shown in Figure 4-3. It is inspired by the design of IPv6 [34] and its flow identification and tagging is similar to that of MPLS [12]. RON does not fragment packets, but

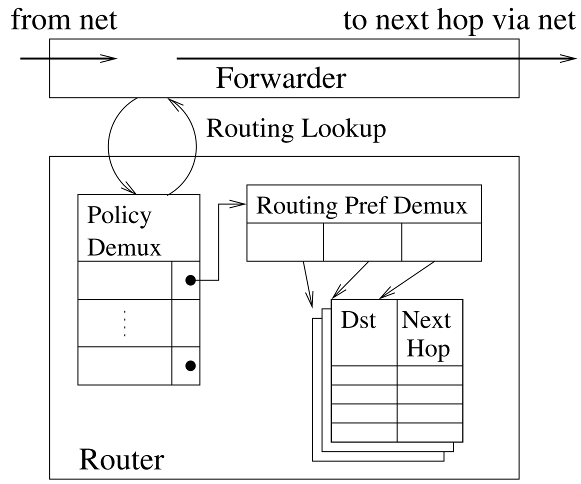


Figure 4-4: The forwarding control and datapath. The forwarder passes the packet header to the routing table, which performs three levels of lookups. The first level is based upon the policy type; the second is based upon the routing preference, and this lookup leads to a hash table of next hops indexed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

does inform applications if they've exceeded the Maximum Transmission Unit (MTU) by sending back ICMP "must fragment" messages if necessary. As in IPv6, applications must perform end-to-end path MTU discovery. RON also 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.

When a packet is received at a RON node, the forwarder examines the packet to determine if it is destined for the 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 4-4.

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 is used to direct 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 client-defined metrics, and still provide them with good default metrics to use. 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.

4.4.1 Implementation

A RON forwarder may be created with no conduits at all; these forwarders run in user-mode with no privilege, and they forward RON packets between nodes. As a more functional example, we implemented an IP encapsulating forwarder tha uses FreeBSD's divert sockets to automatically send IP traffic over the RON and emit it at the other end. This forwarder provides improved IP delivery without application modification. The IP encapsulator provides classification, encapsulation, and decapsulation of IP packets through a special conduit called the `ip_conduit`. The architecture of the IP forwarder is shown in Figure 4-5.

The forwarder first configures the system's IP firewall to *divert* interesting packets to the RON forwarder. For instance, a RON forwarder trying to speed up web browsing within the RON would be interested in traffic to TCP port 80. Thus, the IP forwarder would first install the following two rules in the FreeBSD firewall table:

Source	Destination	Protocol	Src Port	Dst Port	Action
me	not me	`TCP`	80	*	`DIVERT to RON`
me	not me	`TCP`	*	80	`DIVERT to RON`

The forwarder 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. We have not yet needed this optimization in our prototype, but it is straightforward.

Once the firewall rules are installed, the forwarder opens a `divert socket`, a special construct that allows it to receive raw IP packets that match the firewall rules. Through this mechanism, the forwarder can run as a user-space program instead of as a kernel component, though it must run with root privileges to open the socket. The forwarder then waits on the divert socket using `select`, and receives incoming packets using `recvmsg`. The rest of the processing proceeds as described in the design. If the forwarder 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.

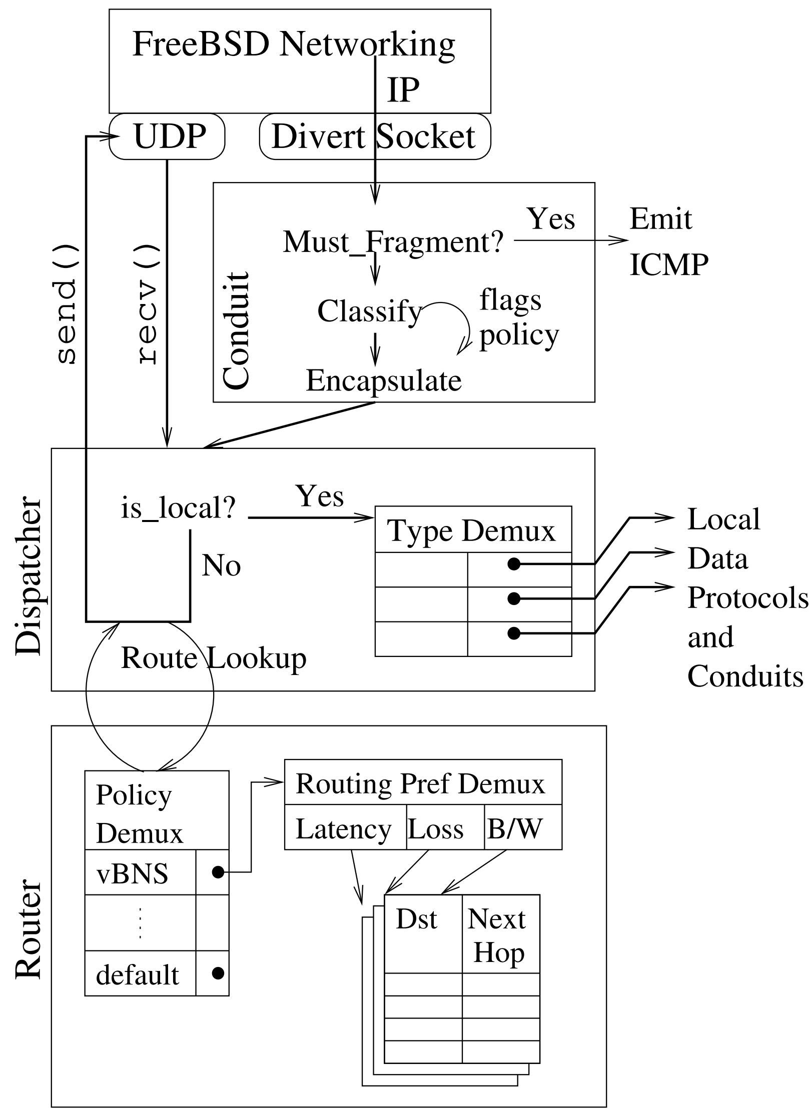


Figure 4-5: The implementation of the IP encapsulating RON client. Packets enter the system through FreeBSD's divert sockets. They are handed to the RON forwarder, which looks up the next hop in the multilevel routing table: The packet is first matched to the appropriate policy table, then dispatched based upon its routing prefix, and finally by its destination node. Using this lookup, the packet is forwarded to the next hop. When the packet arrives at its destination, the conduit writes the packet to a local raw socket, where it re-enters IP processing.

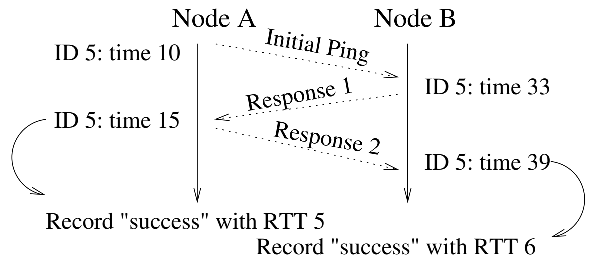


Figure 4-6: The active prober's probing mechanism. With three packets, both participants get an RTT sample, without requiring synchronized clocks.

4.5 Monitoring Virtual Links

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 time-decrementing `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 probe protocols is shown in Figure 4-6. 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 can get reachability and RTT information from 3 packets.

The prober contacts each other host every $PROBE\_INTERVAL$ seconds, plus a random additional offset of $\frac{1}{3}PROBE\_INTERVAL$ seconds. If the packet does not come back within $PROBE\_TIMEOUT$ seconds, then 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 fast probes.

The prober relies upon a timer to perform the following periodic actions:

- Check for outstanding packets that have expired
- Select the node that was probed the longest time ago
- If that time is longer than the probe interval, send a probe

4.5.1 Implementation

The probing protocol, like the membership protocols, is implemented as a RON client using the packet registration services:

```
probing_protocol::configure(forwarder *f) {
   f->register_type(RON_PROBES, this, NULL);
   f->register_timer(PING_TIME, this, NULL); // 100 miliseconds
}
```

By default, $PROBE\_INTERVAL$ is 10 seconds, so the prober sends a probe to each system every 12 seconds on average. $PROBE\_TIMEOUT$ defaults to 3 seconds; if the prober does not hear a response from the remote system within 3 seconds, it reports the packet as lost. The probing protocol uses the performance database client library to submit latency and loss information to the PDB.

With these parameters, the prober will detect an outage within $\frac{4}{3}PROBE\_INTERVAL + 4\ PROBE\_TIMEOUT$ seconds, or about 25 seconds.

4.6 Routing

Routing is the process of building up the forwarding tables that are used in packet forwarding. RON differs from most current packet-switched network routing protocols in that it maintains information about multiple alternate routes and selects the path that best suits the RON client according to a client-specified routing metric. By default, RON maintains information about three specific metrics for each virtual link: (i) latency, (ii) packet loss rate, and (iii) throughput, as might be obtained by a bulk-transfer TCP connection between the end-points of the virtual link. 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 a routing policy and a routing metric.

RON routing considers hops only through a single other node when making routing decisions. This decision allows RON to use non-linear routing metrics like “throughput,” which standard shortest-paths algorithms cannot handle. In practice, we find that restricting RON to only a single indirect hop works well; we show data to support this in Chapter 5.

4.7 Link-State Dissemination

The RON router component uses a link-state routing protocol to disseminate topology information between routers, which in turn is used to build the forwarding tables. Each node in an N-node

RON has $N - 1$ virtual links. Each node's router periodically requests summary information of the different performance metrics to the $N - 1$ other nodes from its local performance database and disseminates its view to the others. The router sends out a routing update on average every $ROUTING\_INTERVAL$ seconds (within 10% in either direction). In our implementation, $ROUTING\_INTERVAL$ is 15 seconds. Routing updates go out regardless of the state of the local routing table, to ensure that clients that may have lost their state, or missed previous routing announcements, are able to function properly.

This 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 approach ensures that the only time a RON router has incomplete information about another RON router is when *all* paths in the RON from one node to another are unavailable.

4.8 Path Evaluation and Selection

The RON routers need a set of algorithms with which to evaluate potential paths. The responsibility of these *metric evaluators* is to provide a number quantifying how "good" a path is according to that metric. 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.

Alternating rapidly between multiple paths is harmful to applications sensitive to jitter and reordering. While each of the evaluation metrics below applies some smoothing, this is not enough to avoid "flapping" between two nearly equal routes: RON routers therefore employ hysteresis. Based on an analysis of 5000 snapshots from a RON node's link-state table, we chose to apply a simple 5% hysteresis bonus to the "last good" route. While this value appears to be a reasonable trade-off between responsiveness to changes in the underlying paths and unnecessary route flapping, more analysis is needed to determine the right hysteresis to use.

Every RON router implements *outage detection*, which it uses to determine if the virtual link between it and another node is still working. It uses the active probing mechanism for this—if the last `OUTAGE_THRESH` (set to 4 in our implementation) probe packets were all lost, then an outage is flagged. 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.

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 requesting an exponential weighted moving average (EWMA) of round-trip latency samples with parameter α from the database. For any link l, its latency estimate lat_l is updated as:

$$lat_l \leftarrow \alpha \cdot lat_l + (1 - \alpha) \cdot new\_sample_l$$

For a RON path, the overall latency is the sum of the individual virtual link latencies: $lat_{path} = \sum_{l \in path} lat_l$.

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 value used by TCP ($7/8$) in its round-trip time estimator [42].

To estimate loss rates, RON uses the average of the last $k = 100$ probe samples as the current average. Like Floyd *et al.* [14], 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 using further techniques developed by Floyd *et al.* that unequally weight some of the k samples. Note that we do not use the loss rates to set transmission rates, but only to compare alternatives.

Loss metrics are multiplicative on a path: if we assume 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 [1, 37], we believe this is an appropriate goal. From the standpoint of improving the reliability of path selection in the face of performance failures, avoiding bad paths is more important than optimizing to eliminate small bandwidth differences between paths. While a characterization of the utility received by programs at different bandwidth rates, similar to that developed by Breslau and Shenker [3], is beyond the scope of this work, we believe that more than a 50% difference is likely to reduce the utility of many programs, and also falls outside the typical variation observed [1]. We concentrate on avoiding throughput faults on this order of magnitude.

The performance of bulk TCP transfers is a function of the connection's round-trip latency and the packet loss rate it observes. Throughput optimization combines the latency and loss metrics using a simplified version of the TCP throughput equation [33]. Our granularity of loss rate detection is 1%, and the throughput equation is more sensitive at lower loss rates. We set a minimum packet loss rate of 2% to prevent infinite bandwidth, and to prevent large oscillations from single packet losses. The formula used is:

$$Score = \frac{1}{rtt \cdot \sqrt{\frac{2\rho}{3}}}$$

where ρ is the *one-way* packet loss probability. Our samples, however, give us the *two-way* packet loss probability. Assuming that losses are independent on the two paths A-B and B-A, then we define $loss_{ABA}$ as the round-trip loss for probe packets sent from A, to B, and back to A. Thus:

$$\begin{aligned} loss_{ABA} &= 1 - (1 - loss_{AB})(1 - loss_{BA}) \\ loss_{AB} &= 1 - \frac{1 - loss_{ABA}}{1 - loss_{BA}} \end{aligned}$$

The maximum absolute error occurs when all of the loss is in the other direction, resulting in an error equal to $\frac{loss_{aba}}{2}$. Combined with the minimum loss rate of 2%, we preclude a relative error of more than 50% when choosing between two high-quality links. Our results show that much of the time, our error for high-quality links is low, but for links with highly asymmetric losses, we sometimes disregard a potentially good link. The benefits of avoiding the high loss rate path seem to outweigh the cost of missing one good link.

This method of throughput comparison is not without its faults. The TCP equation we use assumes that there will be some packet loss on the link—if all else fails, a TCP will start using the bandwidth. An unused 9600 baud modem link, for instance, would be predicted to receive *infinite* throughput! While in practice, RON's throughput optimization works reasonably well, this is an area ripe for further research.

4.8.1 Implementation

Routers implement the `router` virtual interface, which has only a single function call, `lookup(pkt *mypkt)`. We provide a trivial static router, and a dynamic router that provides routing based upon different metric optimizations. The dynamic router is extensible by linking it with a different set of `metric descriptions`. Metric descriptions provide an evaluation function that returns the "score" of a link, and 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()
```

4.9 Policy Routing

Policy routing allows users or administrators to define the types of traffic allowed on particular network links. For instance, network administrators may wish to define a policy that "Only students in the Computer Science department are allowed to use MIT's connection to the Internet2." Other administrators may wish to apply different, less advantageous, policies, of course.

On the Internet, "type" is typically defined only by its source and destination addresses [8]; RON extends this notion to a more general notion of packet type. RON separates policy routing into two components: Classification and routing table formation. Packets are given a policy tag when they enter the RON, and this policy tag is used to perform lookups in the proper set of routing tables. A separate set of routing tables is constructed for each policy by re-running the routing computation without links disallowed by the policy. The generation of the routing tables is shown below:

```
foreach P in policies                  // For each source,dest
  foreach M in metrics                 // pair, find the best next hop
    foreach Dest in peers              // permitted by each particular
      foreach Hop in peers             // policy

        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;
```

The *policy classifier* component provides a policy tag, a conduit-specific data classifier that answers the question, "Is this packet my type?", and a `permits` function to tell the router if the policy permits the use of a particular link. 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

Internet 2 research backbone. This policy classifier takes only a list of networks and subnet masks to match against. Our general policy component accepts a BPF-like packet matcher [30], and a list of links that are denied by this policy. We provide a "first-matched" policy implementation, and do not currently address the issue of policy composition. With the general policy classifier, users may create composed versions of their own policies.

Implementation. We have implemented the clique classifier, and use it extensively in our evaluation. We are in the process of implementing a general policy classifier. Both provide classifiers for the IP encapsulator.

4.10 Performance Database

Measurements may be well-known *system-defined* metrics, such as latency or loss rate, or *application-defined*, such as "the time to download a 10K HTTP object," similar to the approach used in SPAND [46]. We do not know the metrics that will be important to future users; a flexible system must be able to adapt to the changing needs of its users. We provide a performance repository that is capable of storing both system- and user-defined metrics.

Hosts on the same LAN will frequently experience similar network conditions when communicating with other hosts. Just as Web caching is based upon the premise that users on the same network will often correspond with an overlapping set of remote hosts, we wish to leverage information collected by multiple hosts on the local network. The RON performance database should be optionally *shared*. Because the database may not be integrated with the application, programs should not rely on the presence of particular data in the DB—it must be treated as *soft-state*. We define the database API as (potentially) unreliable asynchronous callbacks, and eliminate hard state from the database, thus protecting clients from database faults and making it possible to share the database contents between multiple clients.

It is impractical to send large performance histories to all participants in the RON. Also, measurement data is often noisy, and different clients may have different ways in which they wish to use that data. An outage detector, for instance, may want to know if no packets were successfully sent in the last 15 seconds, but a throughput improver may be interested in a longer-term packet loss average. Therefore, the system needs a flexible *summarization* mechanism.

To support these requirements, the RON architecture uses a separate *performance database* to store samples (Figure 4-7). The database is a generalization of SPAND [46], supporting different

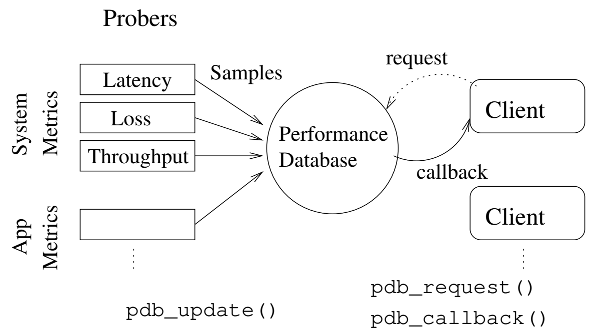


Figure 4-7: The RON performance database.

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.

4.10.1 Implementation

We have implemented the performance database as a stand-alone application running on the Berkeley DB3 backend. 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.

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 wish. 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. AVG-4, then, provides the average of the 4 most recent samples.

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

4.11 Conclusion

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 basic IP-layer "RON forwarder" that encapsulates selected IP traffic flowing out of a computer and forwards it over the RON network.

RON creates its network using a flooding-based membership protocol that periodically beacons soft-state announcements to the other nodes in the network. Once a RON node knows about other nodes, another RON component, the probe protocol, sends periodic probes to the other nodes to measure the network conditions on the virtual links between them. The prober inserts its results into the performance database, where they are accessible to other components. The RON router extracts performance summaries from the database, transmits them periodically to the other RON nodes, and uses the knowledge of the links in the network to compute better paths along which to forward packets.

For it is the greatest truth of our age: information is not knowledge.

- Caleb Carr

Chapter 5

Evaluation

The goal of the RON system is to overcome path outages and performance failures, without introducing excessive overhead or new failure modes worse than those it overcomes. In this section, we present an evaluation of how well RON meets these goals, and show that RONs are beneficial to applications because they can successfully route around path outages and performance failures. In many cases, RONs can also improve the latency and throughput even when a failure has not occurred. We evaluate the performance of our RON-aware IP forwarder client, described in Section 4.4.1, which uses RON for outage detection and uses its loss, latency, and throughput-optimizing routers.

Our evaluation has several parts. For the first parts, we instrument and study our thirteen-site RON deployment to see how effective it is under real Internet conditions. For the second, we study RON in a controlled testbed environment and via simulation, to examine behavior and scenarios we could not safely observe on the real Internet. In this chapter, we examine:

1. RON's response to path outages and performance failures
2. RON's improvements to throughput, loss, and latency
3. RON's effectiveness at routing around packet floods
4. Additional failure modes introduced by RON
5. The overheads introduced by RON
6. RON's routing behavior and stability

Name	Description
`Aros`	ISP in Salt Lake City, UT
`CCI`	.com in Salt Lake City, UT
`Cisco`	.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
Dataset 2 additional hosts	
NC-Cable	MediaOne Cable in Durham, NC
PDI	.com in Palo Alto, CA
Mazu	.com in Boston, MA

Table 5.1: The hosts in our RON deployment, which we study in detail to determine the effectiveness of RON in practice. Italicized hosts are U.S. Universities on the Internet2 backbone. The two European universities, `Lulea` and `VU-NL` are classified as non-Internet2.

Through our experiments, we find that RON is generally successful at routing around Internet failures and performance problems, and that the failure modes and overhead introduced are slight. At first glance, RON's routing stability seems to be adequate, but this is an area deserving of further study.

5.1 RON Deployment Measurements

5.1.1 Methodology

To measure and evaluate RON in real settings, we conducted several data transfer experiments in our deployed RON. Table 5.1 identifies the hosts and where they are located. Figure 5-1 shows the deployed RON nodes geographically, and Figure 5-2 shows a snapshot of the network topology of 12 of the deployed nodes. We determined the network topology from `traceroute`[1] data,

[1]`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.

Figure 5-1: The current 13-node RON deployment. Five sites are at universities in the USA, two are European universities (not shown), two are "broadband" home Internet hosts, one is located at a US ISP, and three are at US corporations.

examining one path from each source to each destination. The network topology graph if Figure 5-2 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.

In our deployment of $N = 13$ nodes, there are potentially $N(N-1) = 156$ different paths between these nodes; `traceroute` data shows that 36 different autonomous systems were traversed, with 74 distinct inter-AS links.[2] The $O(N^2)$ scaling of the path diversity suggests that even small numbers of confederating hosts are able to expose a large set of Internet paths to examination, as mentioned by Paxson [36, 37, 39]. We do not claim that our experiments and results are typical or representative of anything other than our deployment, but we present them in detail to demonstrate the kinds of gains one might realize with RONs.

Most of our hosts were Intel Celeron/733-based machines running some version of FreeBSD, with 256MB RAM and 9GB of disk space. In none of the wide-area tests was the processing capability of the host ever a bottleneck.

Network use policy. Several of our host sites are Internet2-connected educational sites, and we were concerned that this high-speed experimental network might present opportunities for path improvement not available to the general Internet—unsurprisingly, the simulation results in Sec-

[2] Multiple links between an AS pair are treated as a single link.

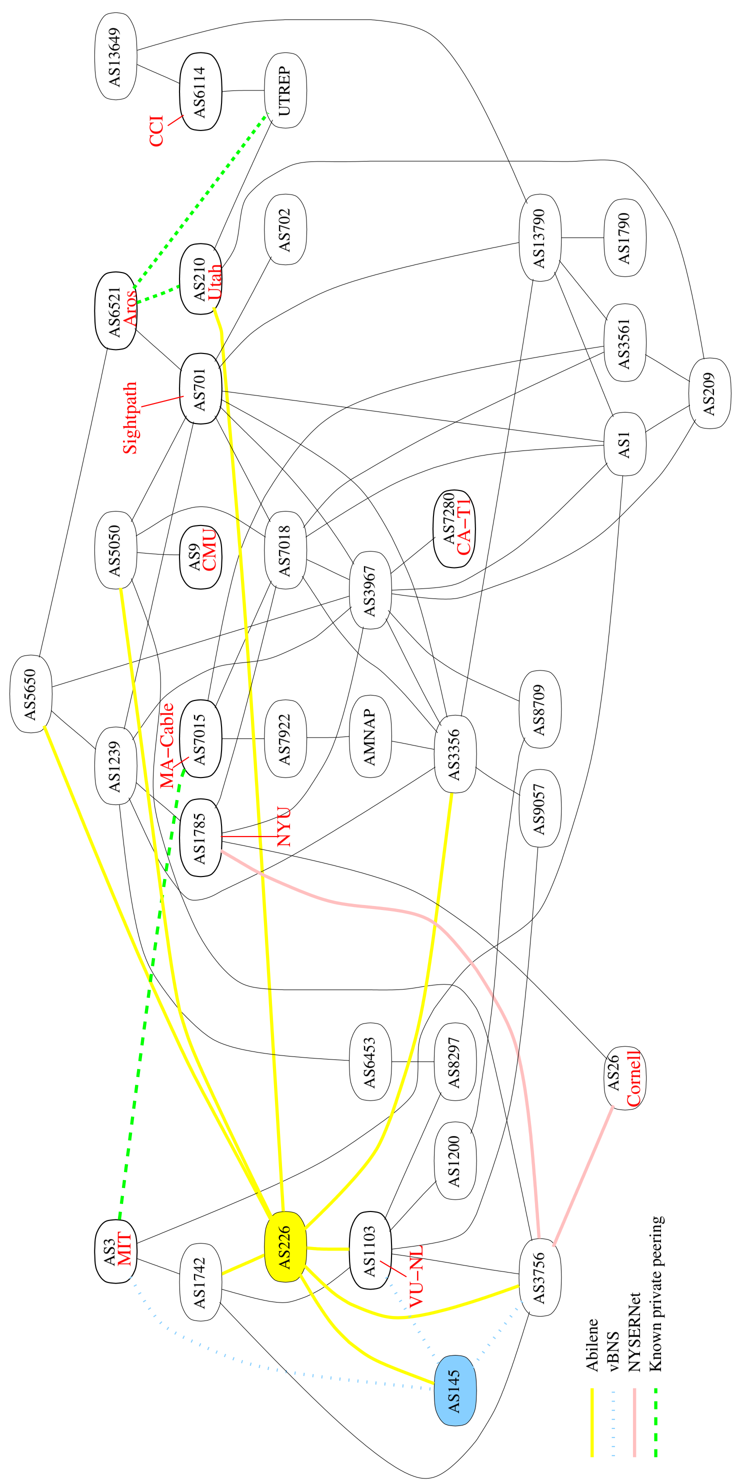


Figure 5-2: 12 of the 13 RON nodes shown in terms of network-level connectivity. Between these 12 sites are 36 different autonomous systems, presenting a rich set of paths from which to choose. Known research networks and private peering links are highlighted.

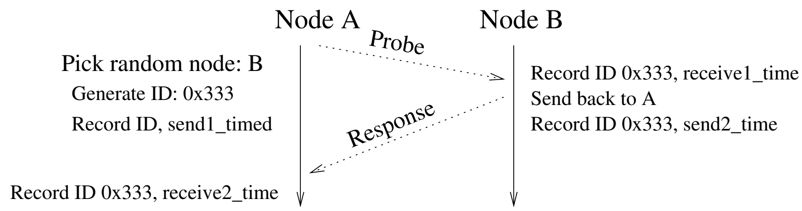


Figure 5-3: RTT and loss sampling. Each node picks a random other node, and sends a probe to it, recording the ID of the probe and the time at which the probe was sent. The receiving host records the time at which the probe was received on the Ethernet interface, retransmits the packet to the sender, and records the time at which it was sent, to account for processing time at the receiver. Finally, the sender records the time at which the packet was received again.

tion 5.5.1 confirm this hypothesis. To demonstrate that our policy routing module works, and to make our measurements closer to what Internet hosts in general would observe, all RON Internet measurements 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 is consistent with the Acceptable Use Policy of Internet2 that precludes commercial traffic, and is therefore not a fictitious policy constraint.

Data collection and availability. To collect data with which to evaluate RON, we set up an independent measuring framework to periodically measure RTT, loss, and throughput. To measure RTT and loss, each host periodically picked a random RON node (other than itself, of course) and sent a UDP probe packet to that node, assigning a random 64-bit ID to the probe, and recording the ID and time of transmission. Upon receipt of the packet, the recipient node stored the Ethernet interface timestamp of the packet's arrival time, sent the packet back to the sender, and recorded the ID and time of retransmission. For these experiments, each host would sleep for a period of time uniformly distributed between 1 and 2 seconds between probes. Each host performed these measurements independently, to avoid missing outages that would have affected a centralized scheduler. All of the measurements incorporated a random jitter to avoid being affected by periodic effects. The process by which we took samples is illustrated in Figure 5-3. The sampling programs do no data processing at the time, keeping the run-time overhead and necessary state minimal. Instead, we post-process all samples in a database, at which time we can correct for clock differences, computer crashes, and other errors. We also found that by using a database, we were able to retroactively extract one-way loss rates without using a complicated probing protocol.

We took throughput measurements in an analogous fashion, running a `ttcp` bulk TCP transfer to a random host every 20-40 minutes. During the throughput samples, we recorded the elapsed time at which each power of two's worth of data (2k, 4k, 8k, 16k, ...) was successfully sent, and the total duration of the transfer. The throughput measurements ran for either 30 seconds, or until 1 megabyte of data was sent, whichever was faster.

In this thesis, we analyze two sets of data. The first was taken for 64 hours between 21 Mar 2001 and 23 Mar 2001. It consists of 10.9 million packet departure and arrival times, producing 2.6 million RTT, one-way loss, and jitter data samples. During this period we also collected 8,855 throughput samples. The second dataset was taken for 85 hours between 7 May 2001 and 11 May 2001, with a slightly larger number of hosts, consisting of 13.8 million raw times, producing 3.5 million RTT, loss, and jitter samples, and 13,283 throughput samples. The data for the host `utah` was removed from the second dataset because of a software configuration error. In this chapter, we use the second dataset primarily to confirm that the results of the first dataset were not a fluke, by examining a slightly different set of hosts at a different time.

5.1.2 Overcoming Path outages

Precisely measuring a path outage, which we defined in Chapter 3 as a failure of the end-to-end Internet connectivity between two hosts, is harder than one might think. One possibility would be to define an outage as the length of time when no packets get through the path, such that applications will fail. As a yardstick, we could use ranges of time in which TCP implementations will time out and shut down a connection; these vary from 120 seconds for some early Solaris versions, to 511 seconds for most BSDs[3] This gives us a metric for when batch applications will fail. However, from our own interactive experience on the Web, we believe that most users have a far lower threshold before declaring the Internet connectivity "dead."

The problem with this small time-scale definition is that robustly measuring this and differentiating between a very high packet loss rate and true disconnections is hard. However, since distributed applications suffer in either case, we find that 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. I.e.,

$$outage(\tau, p) = \begin{cases} 1 & \text{if packet loss over the period } \tau > p \\ 0 & otherwise \end{cases}$$

[3] With Solaris 2.5 and greater, the timeout has been increased to 8 minutes. BSD systems use a table of exponential backoff values [31] (page 459); in FreeBSD, this table is found in `sys/netinet/tcp_timer.c` as `tcp_backoff`.

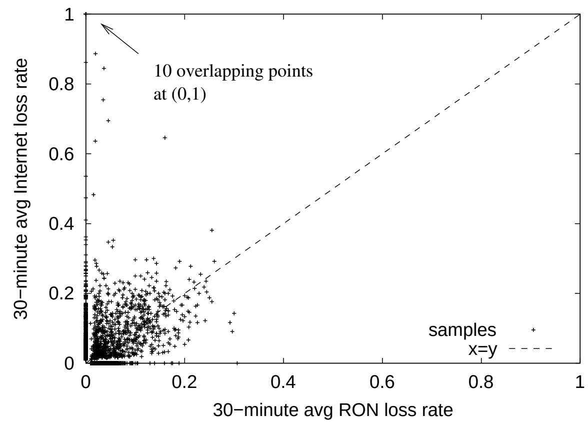


Figure 5-4: Scatterplot showing packet loss rate averaged over 30-minute intervals for direct Internet paths vs. RON paths. There are eight points above the $p = 0.5$ line parallel to the horizontal axis, and seventeen points above the $p = 0.3$ line (30% packet loss). In contrast, RON never experienced a 30-minute loss-rate larger than 30%, and successfully routed around the 50% loss-rate situations.

As noted in Section 3.5, for values of τ on the order of several minutes, a measured value of p that is larger than 30% degrades TCP performance by orders of magnitude, by forcing TCP into frequent timeout-based retransmissions. For practical purposes, a sustained loss rate of 30% or higher renders a path unusable.

Our randomized, decentralized sampling gives us confidence that we did not miss many serious outages on the Internet. There are three ways in which we could fail to observe an outage that occurred: First, if a monitoring machine crashed during the outage; second, if we sent no sample packets along the failed path during the outage; and third, if a complete network partition isolated one of our machines for more than an hour, it would be dropped from the RON until partial network connectivity was restored. During the first measurement set, on which we concentrate, no machines crashed, and there were no complete network partitions. During the second set, a few machines crashed, but we don't believe that these crashes had a serious effect on the results. Finally, to ensure that we sent enough pings during the analyzed intervals, we next examine the sampling methodology. Each host picks a random other host every 2 seconds, worst case (1.5 seconds on average). Since we can detect a path outage by either sending to a host or having a remote host send to us, we therefore have twice the chances of picking a host. The odds of sending to a particular alternate host are $\frac{1}{N-1}$, or in our case, $\frac{1}{12}$. We can treat the event that we select a remote host as a binomial random variable, and view the time between samples on a particular path as a geometric random variable. Basic probability tells us that the expected time between samples is therefore 12

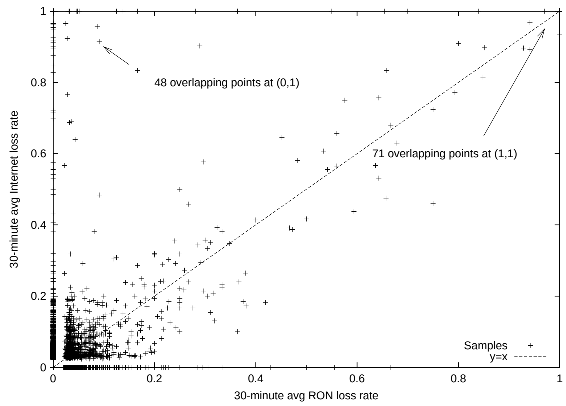


Figure 5-5: Scatterplot for the 30-minute average loss rates in dataset 2. This dataset has 34,534 samples. The dataset includes more monitoring noise, because computers that crashed remained in the monitoring setup for an hour before being automatically removed. Therefore, this graph does not distinguish between an outage and a crash; RON has no chance of routing around a crash, which creates additional situations in which RON could not improve the outages.

seconds. Applying the Chernoff bound, we can state with extremely high probability that we will have sufficient samples in a 5 minute period to detect outages.

How often do these outages occur, and how often is RON able to route around them? Figures 5-4 and 5-5 show a scatterplot of the improvement in loss-rate, averaged over $\tau = 1800$s achieved by RON. To identify outages in the first figure, 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 such points to the right of the $p = 0.3$ line parallel to the vertical axis. Note also that the scatterplot shows many points that overlap, so simply visually counting the points will underestimate the true number.

The precise number of times $outage(\tau, p)$ was equal to 1 for $\tau = 1800$s is shown in Table 5.2 These are statistics obtained by examining 13,650 measured loss-rate averages obtained over 71 hours, spread over all 156 communicating paths in our deployed RON.

These results show that RON generally makes things better, but is not perfect, especially at loss rates smaller than 30%. However, RON was in fact able to successfully route around *all* the outage situations! This is especially revealing because it suggests that all the outages we encountered were not on "edge" links connecting the site to the Internet, but elsewhere, where path diversity allowed RON to provide connectivity even when Internet routing could not. We believe that it's coincidental that RON escaped *every* outage, as we have observed outages during other sample runs in which

Loss Rate	RON Better	No Change	RON Worse
10%	479	57	47
15%	270	16	21
20%	127	4	15
30%	32	0	0
40%	23	0	0
50%	20	0	0
60%	19	0	0
70%	15	0	0
80%	14	0	0
90%	12	0	0
100%	10	0	0

Table 5.2: Improvement in 30-minute loss rates with and without RON, out of 13,650 samples. The numbers for the second dataset are similar, though with more situations where RON could not help, and are shown later in Table 5.4.

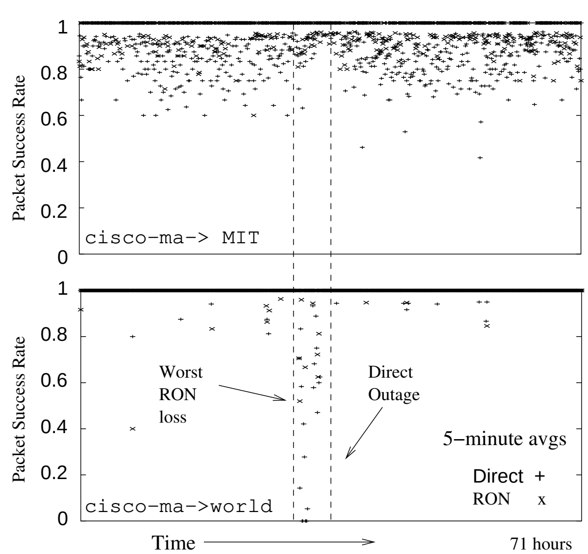


Figure 5-6: Between the dotted lines, the lower figure shows a 10-minute outage from `cisco-ma` to most places on the Internet (the few notches on the horizontal axis at the bottom). The Y axis is the percent of packets that successfully go through; higher is better. The 'x' marks show the packet loss received by RON, and the '+' marks show the direct path. Note that RON never dropped below 50%, where the direct path had occasional bursts of 100% packet loss. RON was able to route around the outage, because `cisco-ma`'s packet loss rate to `MIT` was relatively unaffected during this outage.

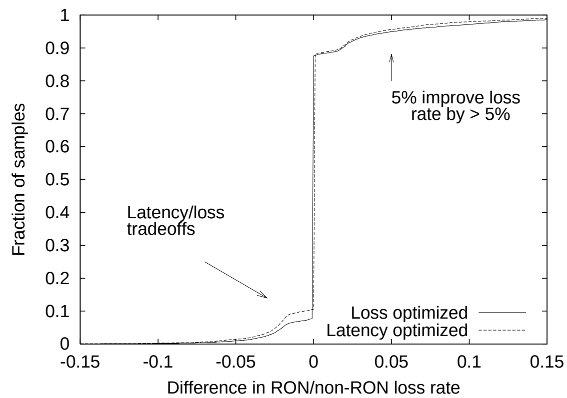


Figure 5-7: The cumulative distribution function (CDF) of the improvement in loss rate achieved by RON. The samples detect unidirectional loss, and are averaged over $\tau = 1800$s intervals.

RON was unable to route around the failures, but it is a strong indication that the simple routing mechanisms we've implemented are effective at routing around many Internet failures.

One way of viewing our data is to observe that there are a total of about 13,650/2 = 6,825 "path hours" represented. There were 5 "path hours" of complete outage (100% loss rate); RON routed around all these. We also encountered numerous outages of shorter duration, often three minutes or so. These are consistent with BGP's detection and recovery time scales. In many of these cases (but not all of them), RON recovered quicker than three minutes.

As one particular example of outage recovery, see Figure 5-6, which shows a 10-minute interval when no packets made it between `cisco-ma` and most of the Internet (the few notches on the horizontal axis of the lower figure are important to note). In fact, among our RON sites, the only site to which `cisco-ma` had any connectivity at this time was `MIT`. RON was able to detect this and successfully re-route packets between Cisco and the other RON sites, by way of MIT.

5.1.3 Outage Detection Time

Upon closer analysis, we found that the *outage detection* component of RON routing was instrumental in detecting bad situations promptly and in triggering a new path. Recall from Section 4.5 that the RON outage detection time can be calculated from the probing rate. RON probes are sent every `PROBE_INTERVAL` seconds, plus a random jitter of up to $\frac{1}{3}$ `PROBE_INTERVAL` extra seconds. Packet loss is determined to occur after `PROBE_TIMEOUT` seconds. When a packet loss occurs, the next probe packet is sent immediately, up to a maximum of 4 "quick" probes. The sys-

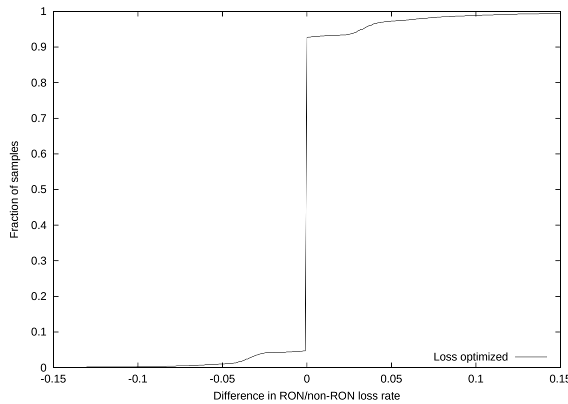


Figure 5-8: The cumulative distribution function (CDF) of the improvement in loss rate achieved by RON with dataset 2. The samples detect unidirectional loss, and are averaged over $\tau = 1800$s intervals.

tem designer can trade off bandwidth (short probe intervals) or resilience to congestion and latency (a shorter probe timeout) for node recovery times. Our implementation detects failures in 12–25 seconds, with an average of 18 seconds.

This suggests that passive monitoring of currently active links will improve the single-link failure recovery case considerably, since the traffic flowing down the link can be treated as "probes." Although recovery from multiple link failures will still take on the order of twenty seconds, it is still significantly faster than recovery with wide-area inter-domain routing.

5.1.4 Overcoming Performance Failures

Loss Rate. In the previous section, we analyzed extreme packet loss situations, characterizing them as outages because high loss rates for sustained periods of time essentially disable most distributed Internet applications. We now analyze the improvements provided by RON under less extreme loss situations. Figures 5-7 and 5-8 summarize these results as a CDF: in our deployment, which has a large fraction of well-connected (to each other) US university sites, RON improved the loss rate by more than 5% (in absolute terms) a little more than 5% of the time—this could mean RON took the loss rate from 15% loss to 10% loss, or from 100% to 95%. Tables 5.3 and 5.4 show breakdowns of the amounts by which RON was able to improve the loss rate, depending on the loss rate with which users are concerned. A 5% improvement in loss rate is substantial for applications that use TCP.

This figure also shows that RON routing is not infallible. There is a tiny, but noticeable, portion

Loss Rate	RON Better	Δ loss	No Change	Δ loss	RON Worse	Δ loss
5%	910	-0.09	220	-0.03	179	0.06
10%	526	-0.15	58	-0.02	49	0.09
20%	142	-0.27	4	-0.03	15	0.10
30%	32	-0.67	0	0.00	0	0.00
40%	23	-0.80	0	0.00	0	0.00
50%	20	-0.86	0	0.00	0	0.00
60%	19	-0.88	0	0.00	0	0.00
70%	15	-0.95	0	0.00	0	0.00
80%	14	-0.96	0	0.00	0	0.00
90%	12	-0.99	0	0.00	0	0.00
100%	10	-1.00	0	0.00	0	0.00

Table 5.3: The change in loss rates with RON, over 30 minute averages. This dataset has 13,650 30-minute samples. The table shows the number of incidents of the specified loss rate where RON could bring the loss rate to under the benchmark, couldn't help, and took the loss rate from under the benchmark to over it. The Δ column shows the average absolute loss rate change in these cases; for instance, at a loss rate of 10%, when RON improved things, the average improvement was a 15% improvement in loss rate.

Loss Rate	RON Better	Δ loss	No Change	Δ loss	RON Worse	Δ loss
5%	1413	-0.09	358	-0.03	395	0.08
10%	568	-0.20	202	-0.03	117	0.13
20%	175	-0.50	142	-0.03	38	0.20
30%	132	-0.65	113	-0.02	18	0.26
40%	111	-0.79	103	-0.01	7	0.36
50%	108	-0.80	96	-0.02	7	0.35
60%	101	-0.83	88	-0.01	5	0.26
70%	93	-0.89	83	-0.01	1	0.29
80%	87	-0.92	80	-0.00	0	0.00
90%	85	-0.89	74	0.00	2	0.04
100%	67	-0.90	71	0.00	1	0.06

Table 5.4: The change in loss rates with RON, over 30 minute averages, for the second dataset. The dataset consists of 34,534 30-minute samples. This dataset is more noisy than the first, since several monitoring computers were rebooted during the experiments, and we did not filter those crashes out; these effects resulted in a higher number of situations where RON could not correct the outages. However, the overall shape of the data is the same, with a larger number of sites participating in the measurements, suggesting that the results from the first dataset are repeatable.

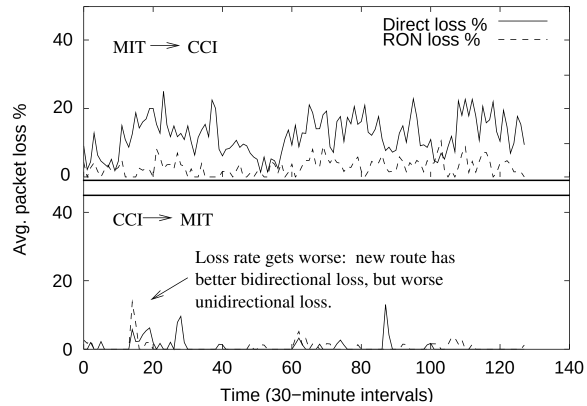


Figure 5-9: The benefits and limitations of bidirectional loss probes. The MIT-CCI link is heavily congested and plagued by outages; RON is able to find a much better path. However, it finds a path with better *bidirectional* loss characteristics—the direct path from CCI to MIT was sometimes better *in 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%.

of the CDF to the left of the -0.05 region, showing that RON can make loss rates substantially worse too. This occurs for two reasons: 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.

The use of bidirectional information by RON's loss-optimizing router leads to a combination of substantial successes but also some observable failures, shown in detail for one path in Figure 5-9. This figure examines three days of loss behavior between `MIT` and `CCI` averaged over 5-minute intervals[4]. 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, there are times when RON makes things substantially worse, although these occurrences are infrequent. This problem arises because our current prober returns bi-directional loss estimates, which the optimizer uses to infer a uni-directional loss rate by assuming symmetric loss rates. During our measurements, the `MIT−CCI` Internet path had a highly asymmetric loss rate. We did not think it would be too important to worry about loss-rate asymmetry in the path selection process, but this data clearly indicates that we should.

[4]Recall that we expect to send 25 samples between every pair of hosts in a 5 minute time period.

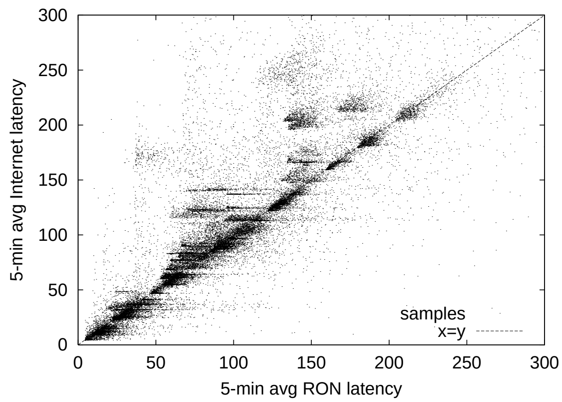


Figure 5-10: 5-minute average latencies. Dots above the x=y line signify cases where the RON latency was lower than the direct Internet latency. The clustering and banding of the samples is reflective of the latency improvements achieved on different host-host pairs at different times.

Latency. RON also reduces communication latency between nodes in many cases. Figures 5-10 and 5-11 show scatterplots of the round-trip latencies achieved via RON against those found directly on the Internet, averaged in 5-minute blocks. During outages, RON may find paths of considerably greater latency than the direct path; for dataset 1, we eliminated 113 out of 39,683 samples due to gross outages. We find that despite the additional hop taken by RON, latency is generally lower.

To understand the source of the improvements in loss and latency, we next examine the loss and latency values over time for *each* communicating pair. Figure 5-12 shows the matrix of cumulative distributions of round-trip latency with and without RON for each path. From this figure, we can see that many paths, particularly those without high-speed Internet 2 connections, experienced some improvement from RON. In some cases, such as the CMU to MA-Cable connection, this improvement was extreme and across the board. In other cases, such as VU-NL to Aros, the improvement only occurred when the latency on the direct link was particularly bad. It is noteworthy that *many* of the paths in our network gained some benefit from RON; the benefit was not limited to a handful of egregiously bad paths.

Figure 5-13 shows the same latency data as it evolves over time. In this view, we can see instances where RON was constantly better, instances where RON was unable to make gains, and instances where RON provided a benefit some of the time. Several of the sites exhibit periodic effects, suggesting that a part of RON's benefit comes from its ability to adapt to dynamic network conditions.

Finally, Figure 5-14 examines the evolution of the 30-minute one-way loss rates over time. In

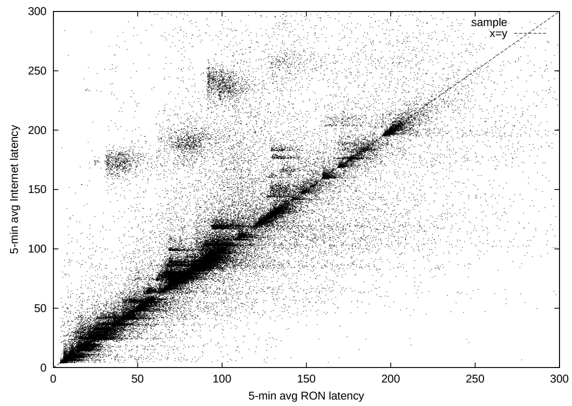


Figure 5-11: 5-minute average latencies from dataset 2. The similarities in some of the outlying clusters and bands suggest that in these cases, RON is improving long-term inefficiencies in routing, not transient congestion-based latency.

this figure, we again observe cases in which the loss rate was constantly bad (e.g. MIT's Internet connection), as well as specific outages where the non-RON loss rate shows sudden, dramatic increases. Some of the loss rate benefits from RON come from long-term loss avoidance, and some of them come from its ability to rapidly detect and route around outages.

Overcoming Throughput Performance Failures

RON can also improve throughput between communicating nodes in many cases. RON's throughput-optimizing router does not attempt to detect or change routes to obtain small changes in throughput since underlying Internet throughput is not particularly stable on most paths; rather, it seeks to obtain at least a 50% improvement in throughput on a re-routed path.

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 over the direct Internet path. Figures 5-15 and 5-16 shows the distribution of these ratios. For dataset 1, Out of 2,035 paired quartets of throughput samples, 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 9 samples achieved a factor of 10 during periods of intermittent Internet connectivity failures. The situation was slightly improved in dataset 2, 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—a case most likely to happen between sites

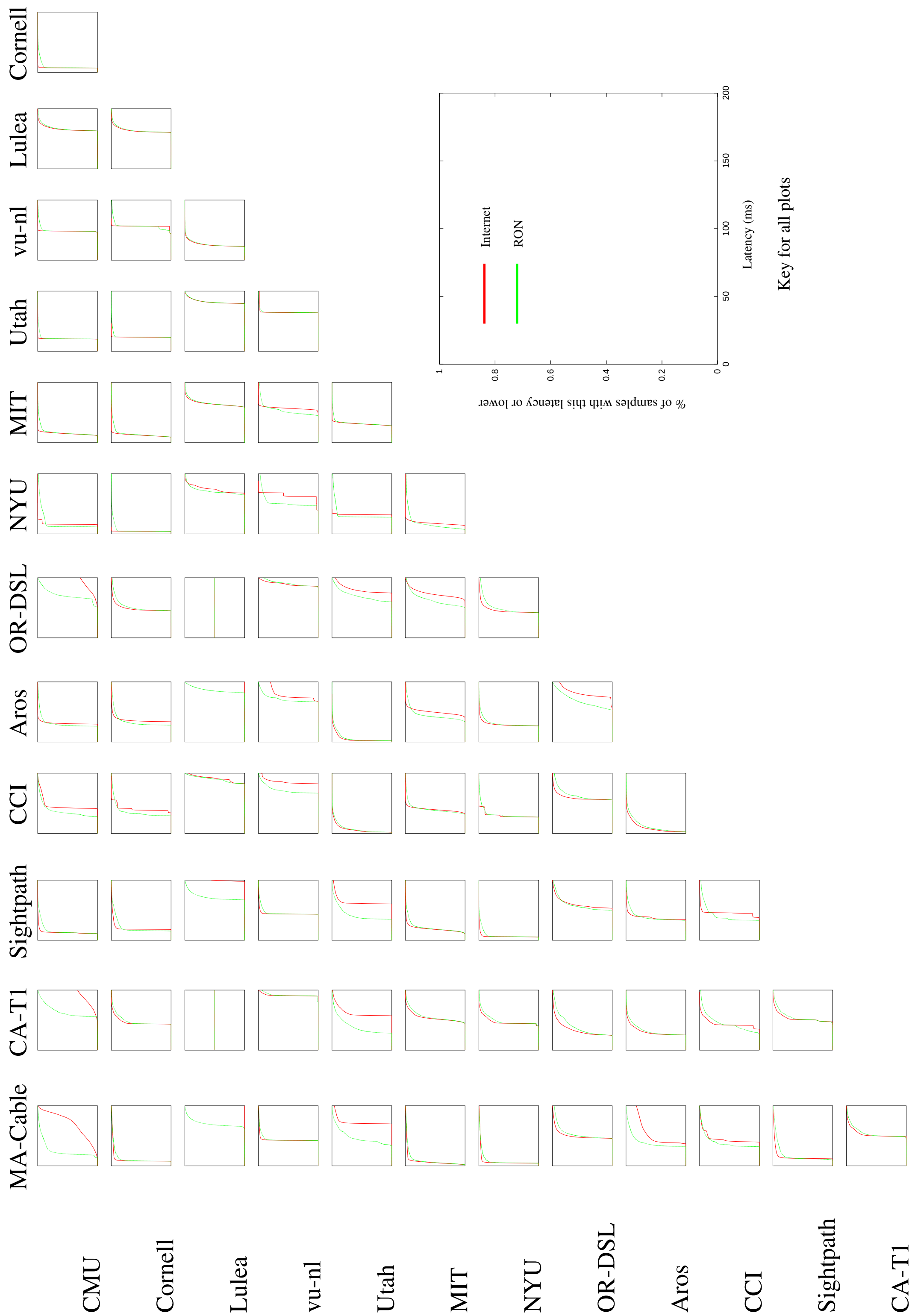


Figure 5-12: The CDFs of the round-trip latencies experienced with (green) and without RON (red) on each set of links examined in the RON experiment. Cases where the red line occurs to the right indicate times when RON improved the latency of the links. To expose interesting detail, the range of latencies is limited to 200ms.

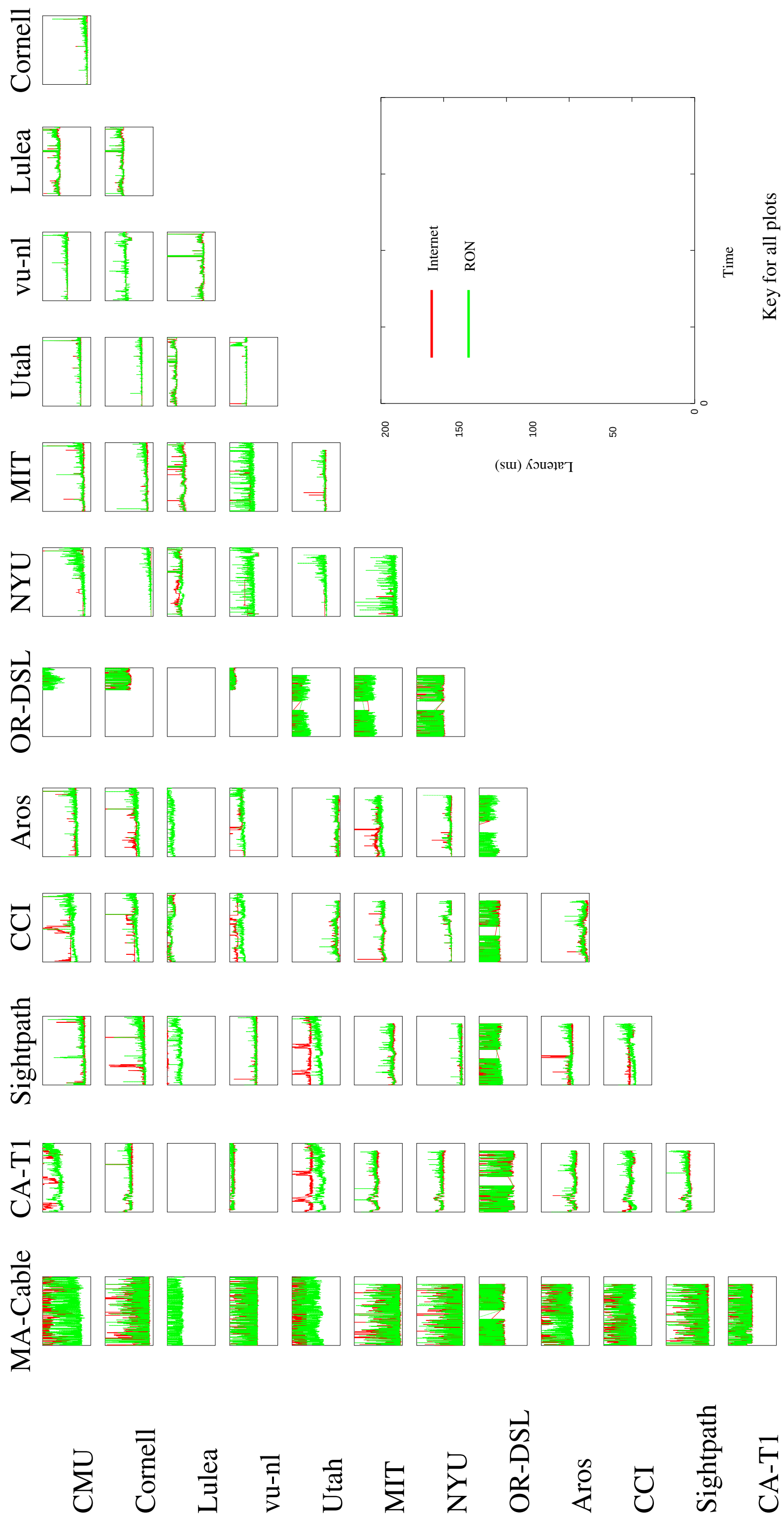


Figure 5-13: The evolution of path round-trip latencies on the Internet (red) and with RON (green) over time. This figure shows several periodic effects in the times when RON is beneficial, as well as links where RON is almost always—or almost never—a win. The two figures where no data appears are cases where the total round-trip latency, regardless of RON, was greater than 200ms.

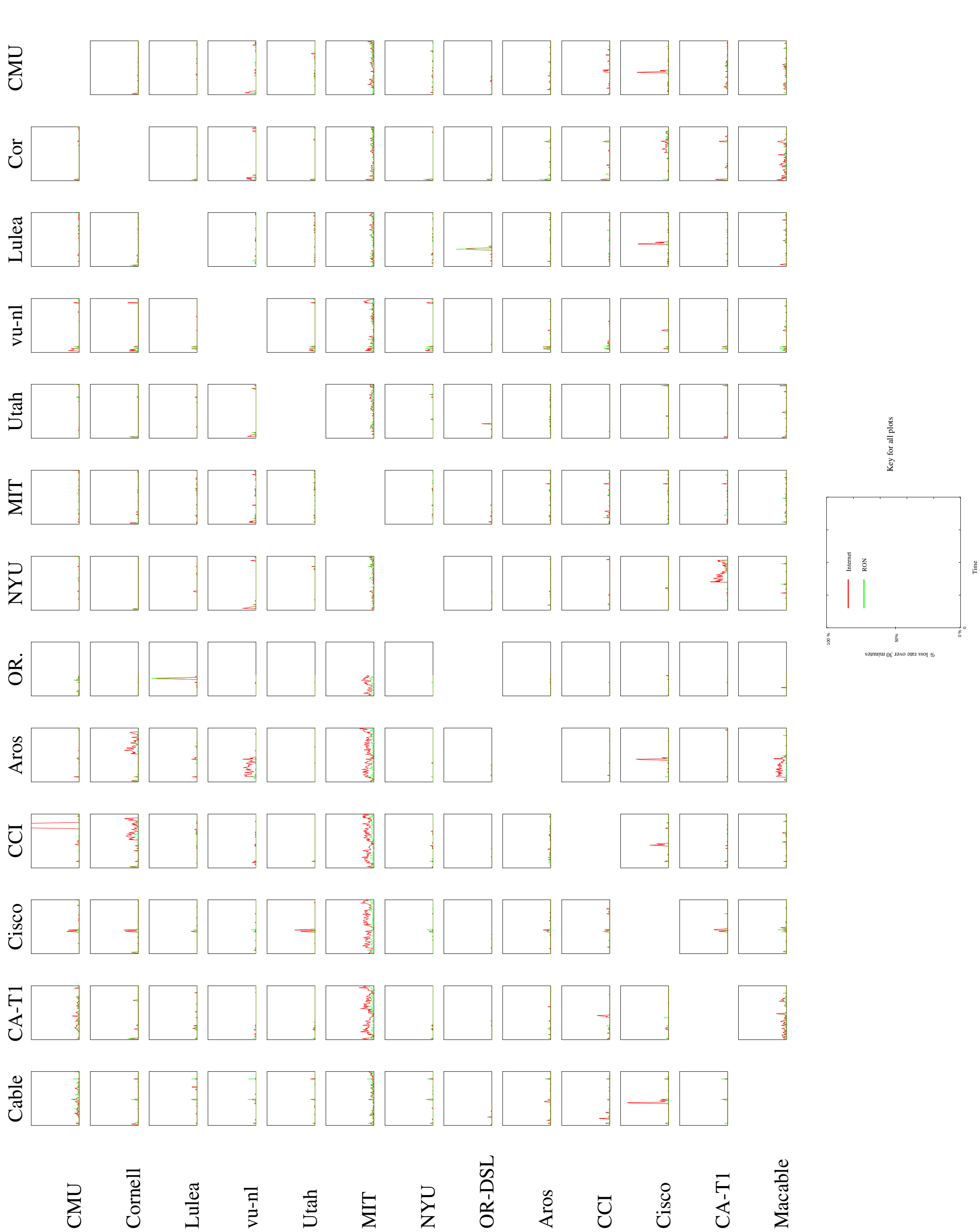


Figure 5-14: The evolution of link loss rates on the Internet (red) and with RON (green) over time. This figure shows several periodic effects in the times when RON is beneficial, as well as links where RON is almost always — or almost never — a win. The two figures where no data appears are cases where the total round-trip latency, regardless of RON, was greater than 200ms.

	cci → macable	MIT → cisco-ma
Direct	112ms, 0.77% loss	38ms, 12.1% loss
Lat-Opt	100ms, 2.9% loss	43ms, 10.5% loss
Loss-Opt	114ms, 0.6% loss	189ms, 3.62%

Table 5.5: Cases in which the latency and loss optimized paths differ.

with fast default paths. In dataset 2, the bandwidth obtained by the *median* connection was virtually unchanged, at just over 99% of its non-RON bandwidth. The *average* connection, however, received 123% of its non-RON bandwidth, showing that RON was able to make a significant contribution to some of the flows, some of the time.[5]

Not surprisingly, the distribution of the improvement in throughput was heavily biased towards slower sites. For obvious reasons, RON was never able to improve the connectivity between two Internet2-connected sites, but sites with network connections in the megabits and under range experienced more frequent improvements. One such Internet2 path, where the average packet loss rate was almost zero, but where RON was able to find an alternate route with lower latency through another Internet2 site because of a backbone router misconfiguration, contributed heavily towards the cases in which RON reduced throughput.

It is not obvious to what degree the best paths for different metrics (latency, loss, throughput) are correlated. It is possible that one "super path" exists between any pair of hosts, and that other paths are less desirable in all respects. While certain links—such as satellite downlinks—have obvious metric asymmetry, these links aren't often encountered in today's Internet, especially in areas of good connectivity. When considering paths through a wide variety of links—from OC12 to 256k cable modems—this metric asymmetry again emerges. We note, however, that we have observed situations in which RON's latency, loss, and throughput-optimizing routers pick different paths. We show one example in Table 5.5.

Between `cci` and `ma-cable`, RON was able to find a moderately faster path, but it had nearly three times the loss rate. In contrast, between `MIT` and `cisco-ma`, RON's latency optimizer made the latency *worse* because its outage detector was triggered so frequently. It managed a 43ms path

[5]These numbers include times when RON was able to find 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 yet 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 dataset 2 is very encouraging.

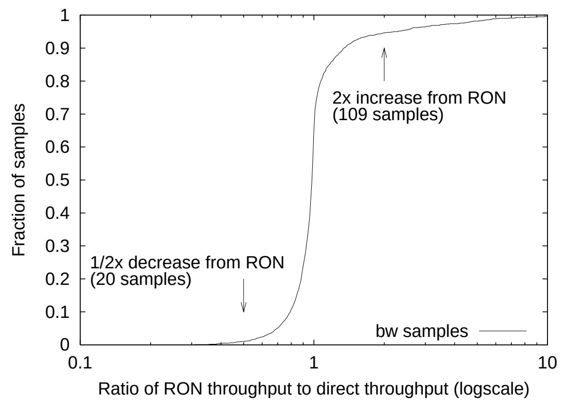


Figure 5-15: CDF of the *ratio* of throughput achieved via RON to that achieved directly via the Internet. 2035 ratios shown.

(on average) while avoiding the larger outages. The loss avoidance, in contrast, reduced the loss rate much more significantly, but at the cost of a factor of five increase in latency.

The existence of these trade-offs (although we obviously do not know the extent to which they occur in the global Internet), and the lack of a single, obvious, "best path" reinforces our belief that a flexible, application-informed routing system can be beneficial to distributed Internet applications.

5.2 Handling Packet Floods

To measure RON's response to events we didn't want to see happen on real networks, and for more detailed performance evaluation, we conducted tests on the Utah Network Testbed [28], which has Intel PIII/600MHz machines on a quiescent 100Mbps switched Ethernet with Intel EtherExpress Pro/100 interfaces. The network topology emulated three hosts connected in a triangle, with 40ms latency links between each. Indirect routing was possible through the third node, but the latencies made it less preferable. The testbed configuration is shown in Figure 5-18.

We first examined RON's response to a flooding-based outage. Figure 5-17 shows the TCP sequence traces of three FTP transfers, from A to C, taken at the receiver. The nodes are all connected by 256k links, with a 30ms round-trip time. The leftmost trace is an uninterrupted TCP transfer. Not surprisingly, it finishes in about 35 seconds. The middle trace shows the transfer running over RON, with a flooding attack beginning at 5 seconds. RON takes about 20 seconds to reroute the connection through B, at which point it proceeds normally and completes after 50 seconds. The horizontal dots show the normal TCP connection in the presence of the flooding attack. TCP traffic

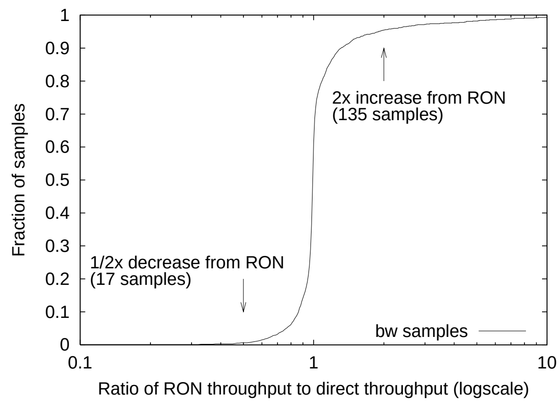


Figure 5-16: CDF of the *ratio* of throughput achieved via RON to that achieved directly via the Internet for dataset 2. 3029 ratios shown. The numbers are very similar to dataset 1; RON realized a slight improvement due to the elimination of a bug during the collection of dataset 1 that added 5% overhead to some of the RON paths.

was still getting through at a very slow rate; BGP would not have marked this link as down. Had it been able to do so, an analysis of the stability of BGP in congested networks [47] suggests that BGP recovery times are at least an order of magnitude larger than RON's—assuming that a BGP route change would not just carry the flooding traffic along the new links.

The flooding was only in one direction (the direction of the forward data traffic). RON still routed the returning ACK traffic along the flooded link—so if BGP had declared the link "dead," it would have eliminated a perfectly usable link.

This analysis shows an advantage of responding to outages on an end-to-end basis, and re-routing only your own traffic. Dealing only with its own "friendly" traffic, RON route changes can avoid flooding traffic in a way that a general BGP route change may not be able to do. Of course, RONs themselves open potential new venues of attacks by attacking the RON nodes and protocols themselves; we leave an examination of the security of RONs for future work.

5.3 Overhead Analysis

RON routing imposes overhead in the form of active probe messages, an extra packet header, and additional processing at an intermediate node.

The RON header is 24 bytes long, so an empty RON payload consumes 52 bytes including the IP and UDP headers. RON probes are 17 bytes long, which means that 69 bytes are sent

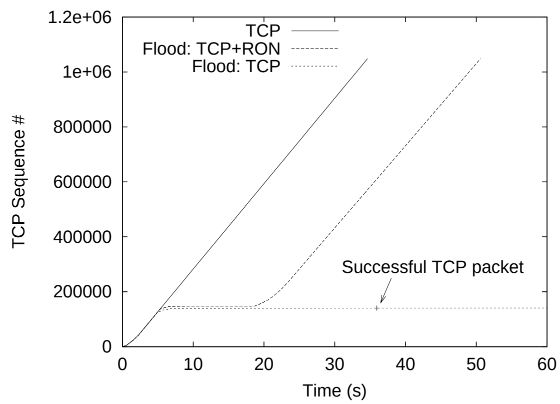


Figure 5-17: A TCP sequence trace at the receiver of a connection re-routed by RON during a flooding attack on its primary link. The recovery time is a combination of RON's link detection time and TCP's retransmission timers. Without RON, the connection is unable to get packets through at more than a crawl, though the link is still technically "working."

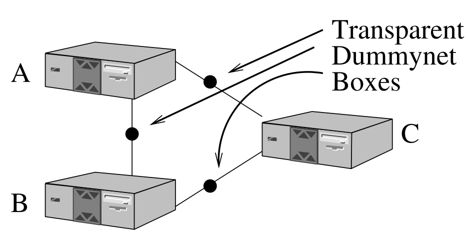


Figure 5-18: Testbed configuration.

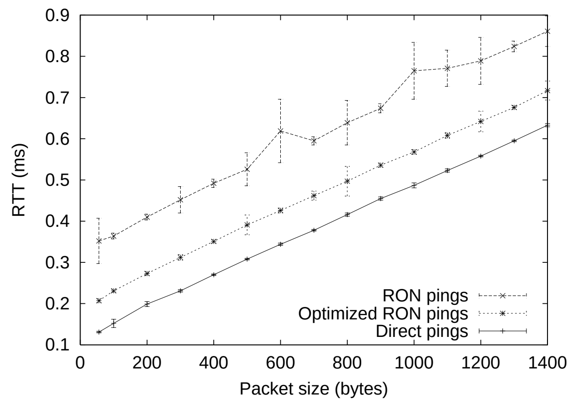


Figure 5-19: Latency increases from RON encapsulation on a 100Mbps LAN. On average, RON adds about 220 microseconds from packet size increases and additional host processing. With optimized direct sending, RON adds only 75 microseconds.

to every other host in the RON every PROBE_INTERVAL seconds, on average. RON also sends routing updates to its peers; the routing header is 8 bytes long, and each virtual link requires 20 bytes of information sent every ROUTING_INTERVAL seconds. So the total routing packet size is $20(N-1)+60$ bytes long, where N is the number of nodes in the RON. Thus, RON routing traffic requires $20N(N-1)+60N$ bytes every ROUTING_INTERVAL seconds. For a large RON of 50 nodes with a 14-second routing interval and 12-second probing interval, the probe traffic consumes (only) about 30 Kbits/s of outgoing bandwidth per node.

RON's per-packet processing overhead is shown in Figure 5-19. To measure the overhead, we measured ping latencies between two machines. RON adds about 70 bytes to the size of the packet; on the 100Mbps Ethernet, this adds about 11 microseconds of actual round-trip transmission time. The rest of the overhead, and the increased jitter, comes from RON copying the packet into user-space through FreeBSD's divert sockets. RON is aimed at lower-throughput wide-area connectivity where latency increases of microseconds are irrelevant; we have not yet tuned our implementation for low packet processing overhead. Our IP traffic encapsulator optimizes the case where the existing network connections are adequate by emitting the packets directly as IP traffic, modulo the MTU reduction noted earlier. Our measurements of the per-packet latency imposed on the forwarding path by a RON forwarder shows that it adds about 220 μs. The middle line in figure 5-19 shows the results of the direct IP sending optimization.

Table 5.6 shows the overhead of RON encapsulation on throughput. RON was designed for operation on slower WAN links, but the 600MHz PIII processors in our test machines were easily

Test	Direct	Opt RON	RON
100 Mbps	11563 KB/s	11562 KB/s	11170 KB/s
1 Gbps	37945 KB/s	11422.19 KB/s	5454.41 KB/s

Table 5.6: TCP throughput with and without RON on 100Mbps and 1000Mbps switched networks. The 100Mbps throughput reduction of 3.6% is almost exactly the amount of extra space added by the RON encapsulation (4%). On a second test with a slower machine (66MHz bus) and Gigabit Ethernet, the cost of memory copies to access the divert sockets dominated the overhead.

able to encapsulate traffic at 100Mbps. Our second test stacked the deck against RON, using a Celeron/733 machine with a 66MHz bus, with an Intel PCI Gigabit Ethernet card. Each data packet is copied into and out of user-land twice (once on the sending end, once on the receiving end). The cost of these copies dominates the packet processing cost. We assume that users on a LAN will instruct their RON IP conduit not to install divert rules for local traffic, which eliminates the overhead without preventing peering with other nodes on the LAN.

5.4 RON-induced Failure Modes

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. As detailed in Section 4.5, our outage detector detects losses in at most $\frac{4}{3}PROBE\_INTERVAL + 4 * PROBE\_TIMEOUT$ seconds.

Recovering from a *remote* link failure between the indirect host and the destination requires that the intermediate host detect the link failure and send a routing update. Assuming small transmission delays, a remote link failure adds between 0 and ROUTING_INTERVAL seconds to the recovery times. In our implementation, a remote link failure takes about 25 seconds to detect, during which time the intermediate node may or may not be able to re-route the packet. Packet losses can increase this time; we require $\frac{RI}{2success\ rate}$ with high loss. On extremely variable networks, the underlying network may heal before RON is able to. If some links are good, however, RON's loss adaptation mechanisms will already have routed the connection around the bad links.

5.5 Simulations of 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 trace of the table containing 5616 individual snapshots. The trace showed the routing information used to determine 876,096 different point-to-point routes over the period of the capture. We analyzed the trace with a shortest-paths algorithm to determine how well RON's single-hop indirect routing worked for latency optimization, compared to a more complicated shortest-paths based routing algorithm.

5.5.1 Single-hop v. Multi-hop RONs

We compared the behavior of RON's single-hop routing to the shortest-paths algorithm. We did not simulate the effects of our no-Internet2 policy in this case, and only considered latency optimization *without* outage avoidance. In the cases when a RON-based path was better than the direct path (51.2% of the time), the shortest path involved only one intermediate RON node 97.8% of the time. This shows that RON's approach of considering only one intermediate node, required to tractably implement throughput optimization, works well.

5.5.2 RON Route Instability

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 instability. The "# Changes" column of Table 5.7 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. On average, the time between samples was 14 seconds.

The median run-length with the 5% hysteresis we implemented was 5, 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.

5.6 Summary

The combination of RON's outage detection and measurement-based routing can provide significant gains in practice. In this chapter, we studied one dataset in depth, which showed that RON can

Method	# Changes	Avg	Med	Max
No Hyst	26205	19.3	3	4607
Hyst-5%	21253	24	5	3438
Hyst-10%	9436	49	10	4607
Hyst-25%	4557	94	17	5136
Hyst-50%	2446	138	25	4703
Random process	260,000	2	1	$<$16

Table 5.7: The duration of a particular route in RON, expressed as the number of concurrent samples in which that route was present.

reduce outages by nearly an order of magnitude. The second dataset, taken with a larger set of hosts at a later time, confirms that the observations in the first dataset are repeatable. The performance of many links can be improved using RON's latency, loss, and throughput optimizing routers, though we believe we have much interesting work remaining in this area. 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.

For so simple an affair, the controversy involved in the making of this dish is vast indeed.

- Marion Rombauer Becker, The Joy of Cooking *[discussing Cheese Fondue].*

Chapter 6

Conclusion and Future Work

In this thesis, we have shown that a Resilient Overlay Network (RON) can greatly improve the reliability of Internet packet routing. A RON works by deploying nodes in different Internet routing domains, which cooperatively route packets for each other. Each RON is an application-layer overlay; each node in a RON monitors the quality of the underlying Internet between themselves and use this information to route packets according to application-specified routing metrics.

We are deploying a working RON at sites in the Internet, and currently have sixteen sites in the USA and Europe. In a 13 node test, over a 71-hour sampling period in March 2001, there were 32 significant outages (sustained loss rates of 30% or higher) lasting over thirty minutes between the 156 communicating pairs of RON nodes. RON's routing mechanism was able to detect and route around them *all*, showing that there is, in fact, physical path redundancy in the underlying Internet in many cases. While their primary benefit is improved resistance to path outages, RONs are also able to improve the loss, latency, or throughput perceived by data transfers; for example, about 1% of the transfers doubled their TCP throughput and 5% of our transfers saw their loss rate reduced by 5% in absolute terms.

The measured performance of our deployment, particularly in the area of fault detection and recovery, demonstrate the benefits of moving some of the control over routing into the hands of end-systems. However, our implementation is not infallible: there are some situations, small compared to the number of times RON works well, when RON makes things worse. We have not found failures or serious performance problems because of RON, but we have found periods when RON worsens a connection's loss rate or throughput. We are working on improving RON's path evaluation and selection algorithms to reduce these occurrences.

RONs create the possibility of misuse, violation of Acceptable Use Policies (AUPs) or violation of BGP transit policies. RON provides a flexible policy mechanism that allows users to implement *better* network policies than those permitted by BGP. Because RONs are deployed only between small groups of cooperating entities, they cannot be used to find "back-doors" into networks without the permission of an authorized user of that network. Users who wish to violate network policy are dealt with at a human level, and that should remain unchanged with the development of RON. On the positive side, if an Overlay ISP buys bandwidth from traditional ISPs and uses RONs for reliable routing, RON allows the expression and enforcement of policies that are not easy to express with BGP.

While our measurements are very promising, and show that a single RON deployed in the Internet can provide improved service for its application, RONs also raise several key research questions that we have addressed only partially, or not at all, in this work. RON's parameters are currently untuned; can we provide better values, or provide an adaptive strategy for determining them? We see several questions in this area:

1. *How many intermediate hops?* We have shown evidence that a single indirect hop is adequate to avoid most problems, but we have certainly not provided a definitive answer. In great part, the answer to this question relies upon a more fundamental understanding of the degree to which the triangle inequality applies to Internet loss and latency.

2. *How do we choose routes?* Route selection involves summarizing link metrics, combining them into a path metric, and applying hysteresis to come up with an estimate of the route quality. How do we best perform these actions for different link metrics? How do we filter out bad measurements, and perform good predictions? How do we combine link metrics (such as loss and latency) to meet application needs? The small size of RONs allows us to potentially perform more expensive optimization steps, such as $A*$ or multi-criteria optimizations, to satisfy user constraints. Are these necessary, and will they be effective?

3. *How frequently do we probe?* The frequency of probing trades off responsiveness and bandwidth consumption. The speed with which failed routes can be detected will determine how well RONs will improve end-to-end reliability.

More fundamentally, RON raises questions about its basic deployability and stability. The RON design trades scalability for improved reliability because it enables the aggressive maintenance and exploration of alternate paths, and facilitates prompt outage detection and re-routing. If RONs become popular, we do not expect this fundamental trade-off to change, but we expect to see many RONs co-existing and competing on Internet paths. This scenario raises more fundamental questions:

1. *How do RONs interact?* What happens if RONs become wildly popular in the Internet? How do independent RONs sharing network links interact with one another and would the resulting network be stable? Understanding these interactions is a long-term goal of our future research.

2. *What are the attacks against RON?* Can we make RONs reasonably secure against attacks? What new security problems does the RON architecture create?

Finally, we would like to develop more applications for RON. Logical next steps include the development of Virtual Private Network (VPN) applications, modifying current Internet conferencing applications to use the RON libraries, and making RON's libraries as easy to use in other research projects as possible.

We hope that the work reported in this thesis, and the software developed in conjunction with it, will help us and others to answer many of these questions in future research by encouraging the widespread deployment of RONs.

Bibliography

[1] H. Balakrishnan, S. Seshan, M. Stemm, and R.H. Katz. Analyzing Stability in Wide-Area Network Performance. In *Proc. ACM SIGMETRICS '97*, pages 2–12, Seattle, WA, June 1997.

[2] Bing Home Page. `http://spengler.econ.duke.edu/~ferizs/bing.html`, 1996.

[3] Lee Breslau and Scott Shenker. Best-effort versus reservations: A simple comparative analysis. In *Proc. ACM SIGCOMM '98*, pages 3–16, Vancouver, British Columbia, Canada, 1998.

[4] CAIDA Tools. `http://www.caida.org/tools/`, 2001.

[5] R. L. Carter and M. E. Crovella. Measuring bottleneck-link speed in packet-switched networks. Technical Report BU-CS-96-006, Computer Science Department, Boston University, March 1996.

[6] B. Chandra, M. Dahlin, L. Gao, and A. Nayate. End-to-end WAN Service Availability. In *Proc. 3rd USITS*, pages 97–108, San Francisco, CA, 2001.

[7] Chesapeake Computer Consultants, Inc. Test tcp (ttcp). http://www.ccci.com/tools/ttcp, 1997.

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

[9] Ian Clarke. A distributed decentralised information storage and retrieval system, 1999.

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

[11] D. H. Crocker. *Standard for the Format of ARPA Internet Text Messages*. Internet Engineering Task Force, August 1982. RFC 822.

[12] Bruce Davie and Yakov Rekhter. *MPLS: Technology and Applications*. Academic Press, San Diego, CA, 2000.

[13] H. Eriksson. Mbone: The multicast backbone. *Communications of the ACM*, 37(8):54–60, 1994.

[14] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-Based Congestion Control for Unicast Applications. In *Proc. ACM SIGCOMM '00*, pages 43–54, Stockholm, Sweden, September 2000.

[15] Paul Francis. Yoid: Extending the internet multicast architecture. `http://www.aciri.org/yoid/docs/yoidArch.ps`, 2000.

[16] I. Guardini, P. Fasano, and G. Girardi. Ipv6 operational experience within the 6bone. 2000.

[17] R. Hagens, N. Hall, and M. Rose. *Use of the Internet as a Subnetwork for Experimentation with the OSI Network Layer*. Internet Engineering Task Force, Feb 1989. RFC 1070.

[18] Robert M. Hinden. IP Next Generation overview. *Communications of the ACM*, 39(6):61–71, 1996.

[19] Mark R. Horton. *UUCP Mail Interchange Format Standard*. Internet Engineering Task Force, Feb 1986. RFC976.

[20] Yang hua Chu, Sanjay G. Rao, and Hui Zhang. A case for end system multicast. In *Proceedings of ACM Sigmetrics*, pages 1–12, 2000.

[21] V Jacobson. pathchar – A Tool to Infer Characteristics of Internet Paths. ftp://ee.lbl.gov/pathchar/, 1997.

[22] John Janotti et al. Overcast: Reliable multicasting with an overlay network. In *Proc. 4th USENIX OSDI*, pages 197–212, October 2000.

[23] Atul Khanna and John Zinky. The revised ARPANET routing metric. In *Proc. ACM SIGCOMM '89*, pages 45–56, 1989.

[24] V. Kumar. *MBone: Interactive Multimedia on the Internet*. New Riders Publishing, Indianapolis, IN, 1996.

[25] Labovitz, Malan, and Jahanian. Internet routing instability. *IEEE/ACM Transactions on Networking*, 6, 1998.

[26] Craig Labovitz, Abha Ahuja, Abhijit Bose, and Farnam Jahanian. Delayed internet routing convergence. In *Proc. ACM SIGCOMM '00*, pages 175–187, Stockholm, Sweden, 2000.

[27] Kevin Lai and Mary Baker. Measuring link bandwidths using a deterministic model of packet delay. In *Proc. ACM SIGCOMM '00*, pages 283–294, Stockholm, Sweden, 2000.

[28] Jay Lepreau, Chris Alfeld, David Andersen, and Kevn Van Maren. A large-scale network testbed. Unpublished, in 1999 SIGCOMM works-in-progress. `http://www.cs.utah.edu/flux/testbed/`, September 1999.

[29] Matrix Information and Directory Services. Beyond mere latency. `http://www.matrix.net/isr/library/beyond_mere_latency.pdf`, 2000.

[30] S. McCanne and V. Jacobson. The BSD Packet Filter: A New Architecture for User-Level Packet Capture. In *Proc. Winter '93 USENIX Conference*, San Diego, CA, January 1993.

[31] M. K. McKusick, K. Bostic, M. J. Karels, and J. S. Quarterman. *The Design and Implementation of the 4.4 BSD Operating System*. Addison-Wesley, Reading, MA, 1996.

[32] The North American Network Operators' Group (NANOG) mailing list archive. `http://www.cctec.com/maillists/nanog/index.html`, November 1999.

[33] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: A Simple Model and its Empirical Validation. In *Proc. ACM SIGCOMM*, Vancouver, Canada, September 1998.

[34] C. Partridge. *Using the Flow Label Field in IPv6*. Internet Engineering Task Force, 1995. RFC 1809.

[35] Craig Partridge, Alex C. Snoeren, W. Timothy Strayer, Beverly Schwartz, Matthew Condell, and Isidro Castineyra. Fire: Flexible intra-as routing environment. 19, 2001.

[36] V. Paxson. End-to-End Routing Behavior in the Internet. In *Proc. ACM SIGCOMM '96*, Stanford, CA, August 1996.

[37] V. Paxson. End-to-End Internet Packet Dynamics. In *Proc. ACM SIGCOMM '97*, Cannes, France, September 1997.

[38] V. Paxson, G. Almes, J. Mahdavi, and M. Mathis. *Framework for IP Performance Metrics*. IETF, May 1998. RFC 2330.

[39] V. Paxson, J. Mahdavi, A. Adams, and M. Mathis. An Architecture for Large-Scale Internet Measurement. *IEEE Personal Communications*, 36(8):48–54, August 1998.

[40] Vern Paxson, Andrew Adams, and Matt Mathis. Experiences with nimi. In *Proceedings of the Passive & Active Measurement Workshop*, 2000.

[41] Dimitrios Pendarakis, Sherlia Shi, Dinesh Verma, and Marcel Waldvogel. Almi: An application level multicast infrastructure. In *Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems (USITS)*, pages 49–60, 2001.

[42] J. B. Postel. *Transmission Control Protocol*. Internet Engineering Task Force, September 1981. RFC 793.

[43] Y. Rekhter and T. Li. *A Border Gateway Protocol 4 (BGP-4)*. Internet Engineering Task Force, 1995. RFC 1771.

[44] S. Savage. Sting: a TCP-based Network Measurement Tool. In *Proc. Usenix Symposium on Internet Technologies and Systems (USITS '99)*, 1999.

[45] Stefan Savage, Andy Collins, Eric Hoffman, John Snell, and Tom Anderson. The end-to-end effects of Internet path selection. In *Proc. ACM SIGCOMM '99*, pages 289–299, 1999.

[46] S. Seshan, M. Stemm, and R. H Katz. SPAND: Shared Passive Network Performance Discovery. In *Proc. Usenix Symposium on Internet Technologies and Systems (USITS '97)*, Monterey, CA, December 1997.

[47] Aman Shaikh, Lampros Kalampoukas, Anujan Varma, and Rohit Dube. Routing stability in congested networks: Experimentation and analysis. In *Proc. ACM SIGCOMM '00*, pages 163–174, Stockholm, Sweden, 2000.

[48] J. Touch and S. Hotz. The X-Bone. In *Proc. Third Global Internet Mini-Conference in conjunction with Globecom '98*, Sydney, Australia, November 1998.

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

[50] Inktomi Content Networking. http://www.inktomi.com/products/cns/, 2001.

[51] Zona Research. Internet business review: Keynote systems. http://www.keynote.com/services/assets/applets/zona_keynote.pdf, 2000.