Gateway Selection in Multi-hop Wireless Networks

by

Rohit Navalgund Rao

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

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

July 2005

Author . . . .
Department of Electrical Engineering and Computer Science
July 1, 2005

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

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

Gateway Selection in Multi-hop Wireless Networks

by

Rohit Navalgund Rao

Submitted to the Department of Electrical Engineering and Computer Science
on July 1, 2005, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

This thesis describes the implementation of MultiNAT, an application that attempts to provide the benefits of client multi-homing while requiring minimal client configuration, and the evaluation of a novel link-selection algorithm, called AvMet, that significantly outperforms earlier multi-homing methods. The main motivation behind MultiNAT is the growing popularity of cheap broadband Internet connections, which are still not reliable enough for important applications. The increasing prevalence of wireless networks, with their attendant unpredictability and high rates of loss, is further exacerbating the situation.

Recent work has shown that multi-homing can increase both Internet performance as well as the end-to-end availability of Internet services. Most previous solutions have required complicated client configuration or have routed packets through dedicated overlay networks; MultiNAT attempts to provide a simpler solution. MultiNAT automatically forwards connection attempts over all local interfaces and uses the resulting connection establishment times along with link-selection metrics to select which interface to use. MultiNAT is able to sustain transfer speeds in excess of 4 megabytes per second, while imposing only an extra 150 microseconds of latency per packet.

MultiNAT supports a variety of link-selection metrics, each with its own strengths and weaknesses. The MONET race-based scheme works well in wired networks, but is misled by the unpredictable nature of wireless losses. The ETT metric performs relatively well at finding high-throughput paths in multi-hop wireless networks, but can be incorrect when faced with heavy load. Unfortunately, neither of these metrics address end-to-end performance when packets traverse both wired and wireless networks. To fill this need, we propose AvMet, a link-selection scheme that tracks past connection history in order to improve current predictions. We evaluate AvMet on a variety of network configurations and find that AvMet is not misled by wireless losses. AvMet is able to outperform existing predictors in all network configurations and can improve end-to-end availability by up to half an order of magnitude.

Thesis Supervisor: Hari Balakrishnan
Title: Associate Professor

Acknowledgments

This thesis is the result of joint work done with Hari Balakrishnan and Dave Andersen. I would like to express my gratitude to Hari for supervising my thesis and allowing me to work in the Networks and Mobile Systems group. I am also deeply indebted to Dave, who was an invaluable source of advice and inspiration. He was always there to bail me out when my mind and vocabulary faltered.

None of the work in this thesis could have been possible without the help of John Bicket and Sanjit Biswas, who maintained the Roofnet testbed on which I collected data. John and Sanjit were especially patient with my questions and generous with their wireless bandwidth. I would also like to thank Michael Ernst for allowing me to TA 6.170, thereby giving me funding for my second semester.

Finally, I must thank Amma, Papa, and Priti for their love and support. I would not be where I am without them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The last several years have seen a surge in the popularity of high-speed Internet connections. Recent surveys estimate that over 55% of all online US households (42 million households overall) are connected to the Internet via broadband links [33]. Although the service is growing more popular, it is not yet reliable enough for many important technical and medical applications. Long service outages can pose anything from a minor annoyance for those wishing to check their email to a major inconvenience for telecommuters. Concerns over the reliability of broadband links can even drive potential users towards other, more expensive Internet access links.

Further exacerbating the availability problem is the growing popularity of wireless mesh networks, which traditionally experience higher loss rates and more unpredictable behavior than their wired counterparts. Participant nodes in these ad-hoc networks collaborate by forwarding packets for one another; this allows two nodes to communicate even in the absence of a direct wireless link. Specially designated gateway nodes provide a bridge between participant nodes and the rest of the Internet.

A key problem facing wireless mesh networks is that of gateway selection. While much work has gone into finding high-quality wireless paths to gateway nodes [10, 15, 21, 29], these studies ignore the performance of the wired path connecting the gateway to the Internet. Therefore, the question of how to evaluate and predict overall gateway performance has gone largely unanswered. Given a client node that has a number of possible gateways to use, which gateway should the client select in order to maximize both performance and availability?

In this thesis, we are concerned with the “end-to-end” availability of Internet services. That is, given a local client and a remote server, what percentage of the time can the two successfully contact and effectively communicate. Completely preventing failures in the Internet is impossible; it is far too large and contains many single points of failure, which by definition are impossible to route around. We therefore do not attempt to eliminate failures, but rather to design systems that adapt quickly to a wide variety of failures and reduce the overall number of points of failure, with the end goal of improving both performance and availability. The rest of this chapter examines some of the challenges in designing such systems and outlines our solutions.

1.1 Challenges

A substantial body of recent work [5, 7, 4, 1] has indicated that client-side multi-homing, or the use of multiple Internet connections, can increase both performance and the availability of Internet hosts. Unfortunately, the techniques developed in these studies were designed with wired networks in mind; we find that these solutions do not always translate successfully to the wireless landscape.

The main challenge here is that wired and wireless networks suffer from very different modes of failure, which makes it hard to devise a single solution that works equally well for both. The main source of packet loss in wired networks is congestion. When a router faces more traffic than it can handle, it is forced to drop some packets, preventing them from reaching the destination host. Hardware failures can also lead to packet loss; they are a longer-lasting but rarer source of failure [24]. As a whole, wired networks usually work, but suffer from occasional long outages.

Wireless networks, on the other hand, do not transmit data through a dedicated physical medium. Wireless networks allow for client mobility, but this freedom comes at the expense of greater packet loss due to various forms of interference, especially electromagnetic interference. These losses are fundamentally different from losses due to congestion. Interference is typically bursty and short-term, meaning a packet that was lost during one transmission may successfully get through on the next. Interference is also unpredictable, meaning a wireless network is subject to unforeseen, drastic changes in performance. To work successfully in wireless networks, systems must be able to handle frequent, short-lived outages.

1.2 Approach

This thesis evaluates the performance of various link-selection predictors when applied to a variety of network configurations. We evaluate the performance of MONET [7], which uses connection establishment times as its main source of data, selecting whichever link connected fastest. We also look at the ETT metric [10], which uses periodic broadcast probes to find the best wireless path between two nodes. We finally compare both these predictors to single-link performance as well as to an upper bound on possible performance gains. All our results are derived from the analysis of trace data collected on live networks.

With the exception of ETT, we analyzed all of the above-mentioned predictor schemes in three network configurations: two wired links, two wireless links, and one of each. The ETT metric only applies to wireless networks, so it was only analyzed in the dual-wireless scenario. For each configuration, we experimentally fetched a set of Web objects through each link and used the collected trace data to evaluate the performance of each predictor. Our main findings are as follows:

1. **Existing predictor schemes are not effective in the face of unpredictable wireless losses.** We find that the unpredictable nature of wireless losses renders the MONET predictor ineffective. In wired networks, the MONET predictor is able to improve performance beyond what a single link could provide, but it struggles to match single-gateway performance in multi-hop wireless networks.

2. **Exploiting multi-homing does not require elaborate client-side configuration.** This thesis presents MultiNAT, a framework that exploits client-side

multi-homing to improve Internet performance and availability. MultiNAT follows in the footsteps of a number of applications that had similar goals. However, MultiNAT's main strength is its ease of use. Previous solutions to the availability problem forwarded packets through a dedicated overlay network or relied on complicated client configurations; both approaches are difficult for the average user to adopt. MultiNAT, on the other hand, is designed to be simple to install. MultiNAT can easily replace an existing Internet gateway, requiring no client configuration changes.

3. **Past history is a reliable indicator of future performance.** To properly take advantage of the benefits of multi-homing, MultiNAT requires a predictor that decides which ISP to use for a particular connection. To fill this void, we present AvMet, a general-use predictor scheme for all network configurations. AvMet uses connection establishment times as well as past performance history in order to select which link to use. When compared to existing wired and wireless prediction metrics, AvMet provides better performance and improves availability by up to half an order of magnitude.

1.3 Contributions

This thesis presents the design and implementation of MultiNAT, a technique that allows any user to easily take advantage of client-side multi-homing. We analyze the performance of various prediction metrics, providing an extensive comparison of metric performance across a variety of network configurations. We introduce and evaluate AvMet, a history-based prediction scheme that provides up to half an order of magnitude improvement over existing predictors in *all* network configurations.

The remainder of this thesis is organized as follows: Chapter 2 presents background information, describes various related work, and outlines the motivations behind this thesis. Chapter 3 details the overall architecture of the MultiNAT system. Chapter 4 evaluates the performance of various link-quality predictors in the MultiNAT framework; we analyze two existing metrics and present a third called AvMet.

We summarize our conclusions in Chapter 5.

Chapter 2

Background and Related Work

The work described in this thesis builds upon a large foundation of research in Internet reliability and wireless routing protocols. This chapter presents related work in measuring Internet availability and routing around failures using alternate paths. Described in detail is MONET [7], an existing availability solution that uses explicit client multi-homing in order to increase availability by over an order of magnitude. This chapter also describes work done to find high-throughput paths between nodes in multi-hop wireless networks. The presented metrics do not take into account the potential differences in wired paths outside the wireless network, but they are included here as background for our later analysis of their weaknesses. This chapter concludes by introducing Roofnet, the multi-hop wireless testbed that we used to evaluate metric performance, and by presenting a brief description of Network Address Translation.

2.1 Availability and alternate-path routing

From the perspective of a wireless node, the end-to-end availability of Internet services depends upon both the wireless components and the wired-Internet components of the path to Internet hosts. We first examine the current state of wired-Internet host availability, as well as the various steps that have been taken to improve a wired user's ability to successfully contact a destination server. Paxson's 1997 study [27] used periodic traceroutes between Internet sites to measure availability and identify pathologies in Internet routing. It found that the likelihood of encountering a routing pathology was 1.5% in 1994 and more than doubled to 3.3% in 1995. More recent studies estimate that end-to-end Internet availability has improved slightly to between 95-99.6% [14, 25, 28]. Unfortunately, this still implies that a user trying to reach an Internet service will be unsuccessful a non-negligible percentage of the time.

Labovitz *et al.* [25] agreed with Paxson's findings that most Internet outages are short-lived (on the order of seconds to minutes) and conjectured that these shorter outages are likely caused by persistent congestion. They found that while 40% of all failures were repaired in under ten minutes, an equal number took over half an hour to be fixed.

These studies suggest that availability of Internet hosts, defined as the fraction of the time that the host is working and reachable, varies greatly but ultimately falls short of the standards set by other major communications systems. The US telephone network, for example, provides "four to five nines" (99.99%-99.999%) of availability [23].

With this as the backdrop, a number of studies sought to determine if alternate-path routing could improve the availability of Internet systems. The Detour study [32] in 1999 was one of the first to show that using alternate paths could result in improved performance. Savage *et al.* collected traceroute measurements between pairs of Internet hosts and calculated average latency and drop rates for each path. They found that there was a lower-latency non-direct route for roughly half the host pairs in their study.

Resilient Overlay Networks (RON) [5, 6] are based on the same basic principle as Detour; they establish a cooperative set of nodes that forward packets for one another in order to increase performance. RON differs from Detour, however, in that one of its end goals was to route around path failures in the Internet, not just to decrease latency.

Unfortunately, RON is far from a panacea for the average user; it only provides path redundancy benefits to communication between member nodes. NATRON

[35] seeks to expand alternate-path benefits to all destinations by tunneling packets through RON nodes to host servers. This approach gives NATRON nodes multiple paths to all destination servers, even those not in the NATRON overlay. Yip found that using alternate paths through the overlay had the potential to reduce average download times by 18% and cut the number of downloads taking longer than 30 seconds by 16%.

SOSR [18] is Scalable One-hop Source Routing, a technique to route around path failures. When SOSR detects a failure, it attempts to send packets indirectly through a randomly-chosen intermediary; like NATRON, SOSR increases the likelihood of a member node successfully communicating with destination servers, even those not in the overlay. Gummadi *et al.* found that routing packets through a set of 4 intermediary nodes allowed SOSR to recover from 56% of network failures. Their results were tempered, however, by the finding that up to 60% of the failures on paths to broadband hosts were on the last-hop, failures from which it was impossible to recover.

Akamai's SureRoute [1] is a commercial product that implements a similar alternate-path failover scheme. When SureRoute detects that the direct path to a server is inaccessible, it automatically searches for and routes traffic over an alternate path. SureRoute is not meant for the home user, however. It requires a dedicated "Edge Server" that monitors paths and fails over to alternate routes. Because of its complicated setup, SureRoute is targeted at large corporations that can bear its associated setup and maintenance costs.

Akella *et al.*'s analysis of multi-homing [4] found that using multiple Internet Service Providers (ISPs) increased performance and resilience for both content providers and content consumers. For the consumers, multi-homing with two ISPs reduced average latency by 25% or more. Akella conjectured that a NAT-based scheme similar to the one presented in this thesis could take advantage of multi-homing by routing traffic through the best ISP.

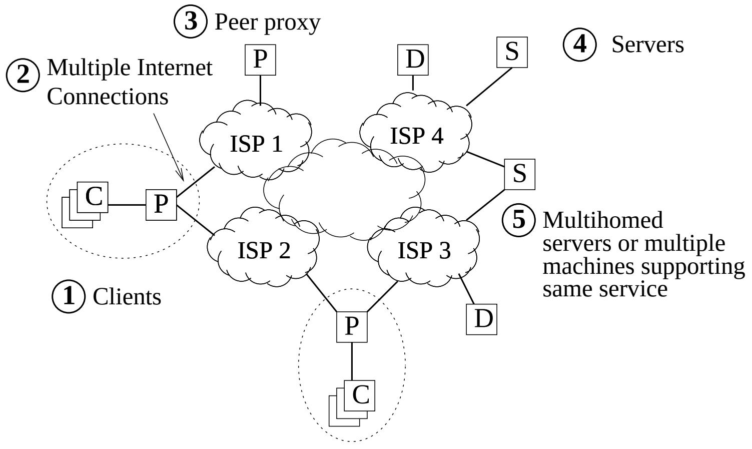

3 Peer proxy
2 Multiple Internet Connections
P
D
S
4 Servers
ISP 1
ISP 4
C
P
S
ISP 2
ISP 3
5 Multihomed servers or multiple machines supporting same service
1 Clients
P
D
C

Figure 2-1: The MONET environment. Clients (1) forward Web requests via their local MONET proxy (2). These local proxies attempt to contact the destination servers (4) either directly, or indirectly through a peer proxy (3). MONET increases the availability of all Internet hosts, but its improvements are greatest for multi-homed servers (5). Figure reproduced with permission from [7].

2.1.1 MONET

This thesis builds upon the work done by Andersen *et al.* on MONET (Multi-homed Overlay Networks) [7]. MONET seeks to reduce perceived delay and increase Web site availability by (a) using multi-homed clients; (b) taking advantage of multi-homed servers; and (c) forwarding requests via an overlay network of peer proxies, which provides multiple indirect paths between the client and server. Figure 2-1 shows the MONET environment; multi-homed proxies issue Web requests on behalf of their clients, sometimes forwarding these requests indirectly through peer proxies.

The three techniques listed above can create hundreds of possible paths between a client and a server; testing them all would impose an unnecessarily large burden on the server. One of the main contributions of MONET is its waypoint selection algorithm, which dynamically selects what paths to use, the order in which to try them, and how long to wait before testing the next path. The algorithm uses the results of periodic probes to estimate link quality and path success rates. By reducing a set of hundreds of paths down to only a handful of the most promising ones, the waypoint selection algorithm is able to provide similar benefits as trying all paths concurrently, but with much less overhead.

Analysis of data collected from six MONET installations shows that MONET overcomes 60% of all outages. In addition, MONET is able to route around nearly all non-server failures. MONET with waypoint selection provides “one to two nines” of availability improvement, while imposing less than 10% packet overhead and negligible ($< 1\%$) byte overhead compared to using a single link.

This thesis analyzes the effectiveness of a MONET-style “race” scheme, but we omit the waypoint selection algorithm in order to make our implementation more light-weight. The basic link-selection mechanism in MONET uses the link with the fastest connection time; MONET only uses waypoint selection as a means of reducing the number of paths to probe. In our analysis, connection times are the only external inputs used when making predictions, and all paths to the server are tested.

2.2 Wireless routing protocols

Although little work has been done to determine which gateway in a multi-hop wireless network provides the best end-to-end wired-Internet performance, many studies have attempted to find the best wireless paths to these gateways. All ad-hoc wireless networks need to address two main questions: 1) How do member nodes estimate link quality; and 2) How do nodes advertise routes to their neighbors. Our results in Chapter 4 show that classic wireless link-selection techniques do not provide the best end-to-end Internet performance; nevertheless, they are presented here as background for subsequent chapters.

Minimum hop-count is a widely-used quality estimation metric. When attempting to choose between multiple paths to a host, nodes select a path with the fewest number of hops, or the fewest number of intermediate nodes. Minimum hop-count is simple to implement and requires little information transfer between nodes. When broadcasting route updates, nodes need only provide a list of neighbors; other nodes can then use this information to calculate overall hop-counts for potential paths. One drawback to minimum hop-count, however, is that minimizing the number of hops also maximizes the distance traveled during each hop. This longer distance exposes transmissions to more interference and greater loss rates. Recent research has shown that the highest-quality paths often contain more than the minimum number of hops [15].

The ETX metric [15] seeks to find high-quality wireless paths in relatively static topologies. It works by calculating the expected number of transmissions and retransmissions required to send a packet from its source to its destination. A higher ETX metric means more transmissions are required, and therefore the link is of poorer quality. The ETT metric [10] is a slight variant of ETX. ETT measures the "expected time for transmission" and is essentially the ETX metric with variable bit-rate selection.

While minimum hop-count and ETT provide algorithms for estimating link quality, they do not explain how nodes receive route information in the first place. Routing protocols such as DSDV and DSR provide a mechanism for disseminating such route information.

DSDV [29] is a distance-vector routing protocol that uses periodic route broadcasts to find paths in multi-hop wireless networks. Each node in the network maintains a table of the best route to each destination. When a node receives a packet, it forwards the packet as indicated by its routing table. When the node receives an advertisement that contains a better route than the one it has, the node updates its routing table. DSDV also uses a settling time mechanism to avoid unnecessarily propagating inferior routes. Each node in the network keeps a weighted average of the time elapsed between hearing the first update for a route and hearing the best update for that route. This allows each node to estimate the amount of time needed to hear a good sampling of available route updates. No node will advertise a new route until at least twice this average settling time has elapsed. DSDV acts independently of the metric used to estimate link quality; the original implementation used minimum hop-count as its metric, but cost or expected loss rates can be used in its place.

DSR [21] is a reactive routing protocol; where DSDV broadcasts periodic announcements to keep nodes up-to-date on current path condition, DSR waits until it has data to send and then broadcasts a route request. When a node receives a route request, it appends itself to the current route and forwards the new route on to other nodes. The node also uses the multi-hop network to returns a route reply back to the original sender, which allows the requesting node to select the best path from among the replies it receives. Nodes using DSR route around highly asymmetric links by maintaining a blacklist of neighbors with unidirectional links. These asymmetric links would otherwise degrade performance, for they allow data to be sent in one direction with high probability but lose many of the return acknowledgments.

2.3 Roofnet

We collected much of our data on Roofnet [13], an 802.11b mesh network. The testbed is currently in the experimental stages and contains about 40 nodes scattered through Cambridge, MA. Access to nodes outside of the Roofnet network is provided by a set of gateways, which bridge the wireless network and the rest of the Internet. The Roofnet developers are also installing another mesh network in a housing development in downtown Boston. The goal is to turn this second mesh into a self-sustaining network community that provides Internet access for the development [2].

As an ad-hoc network, Roofnet performs relatively well. Their website cites typical latencies of dozens of milliseconds and throughputs that can reach over 100 kbyte/s. However, it is still a wireless network, and therefore all the problems that plague wireless networks also affect Roofnet. While typical latencies between Roofnet nodes are low, they can vary greatly from minute to minute. Roofnet also relies on its users to provide gateways to the Internet, and these gateways can be unreliable at times. For these reasons, Roofnet is difficult to trust as one's main access link to the Internet.

A deterrent to widespread Roofnet use is the fact that using Roofnet is an "all-or-nothing" commitment. There is no existing framework to simultaneously use both Roofnet and a broadband connection to the Internet. This thesis seeks to remedy this by presenting a NAT daemon that automatically routes connections along the predicted best path. The goal is to provide users with the benefits of high Roofnet bandwidth when it is available, while avoiding the long outages which Roofnet is subject to.

2.3.1 Click

The majority of Roofnet's routing protocols and link quality estimation metrics are written in Click [22]. Therefore, we introduce Click here to provide background for the changes described in Chapter 3. Click users can create complex router functions by linking together primitive elements, which are included in the base distribution. If customized element functionality is desired, those elements must be written in C++. Click can be run either at userlevel, which requires no superuser privileges, or as a kernel module, which provides better performance.

2.4 Network Address Translation

Network Address Translation (NAT) is a technique that maps one set of IP addresses to another set of alias IP addresses [20]. Packets sent by a client are intercepted by an intermediary NAT router, which rewrites the source IP of the packet to one of the alias addresses. The NAT router then forwards the packets to the destination host. The destination host responds to the NAT router's alias address, and the router translates the reply packets back to using the client's IP address. The overall effect of NAT is to mask the client's actual IP address and make it appear that packets are originating from the NAT router. Figure 2-2 shows a typical NAT network configuration.

Our solutions for improving Internet availability all use a generalized form of NAT called Network Address and Port Translation (NAPT). While network address translation requires a one-to-one mapping between client IPs and alias IPs, NAPT rewrites TCP and UDP port numbers as well as IP addresses, allowing multiple clients to share one alias IP address. NAPT is frequently used in homes to share a single Internet link among many computers. For the purpose of simplicity, we follow the colloquial meaning of NAT; all references to NAT in future chapters really imply NAPT functionality.

A NAPT router acts as a gateway for its clients and rewrites the source IP ad-

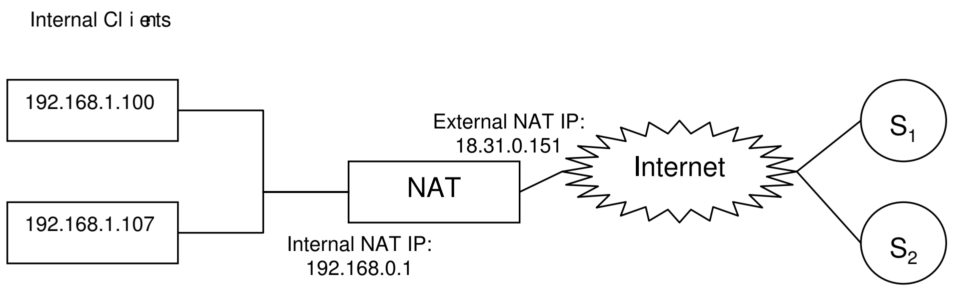


Figure 2-2: An example NAT setup. The internal client nodes all use the NAT middlebox as their gateway to the Internet. The middlebox rewrites the source address of outgoing packets to be the external alias IP. It then passes the modified packets along to the destination server. When the server replies, the NAT middlebox rewrites the destination IP to be that of the appropriate internal client. NAT allows multiple internal clients to share a single external IP address.

dresses of their packets. However, unlike NAT, NAPT must deal with the possibility that two internal hosts may attempt to establish a TCP connection to the same destination IP, with the same source and destination ports. Doing so would violate the uniqueness of the TCP *(source ip, source port, destination ip, destination port)* tuple, and the destination host would not be able to tell the two connections apart. NAPT handles this situation by changing the source port of one of the connections to an unused port. This alias information is then stored in a translation table so that the NAPT router can unalias the packets when they return from the destination host. There is a theoretical limit on the number of internal clients and simultaneous connections that a NAPT router can support, but this limit is rarely reached by the average user.

While NAT and NAPT may seem simple in theory, they face a number of challenges in practice. A number of protocols embed local IP information within their messages, which means that NAT routers must also translate these embedded IP addresses [19]. Some protocols, such as FTP and IRC, store this embedded information in predictable places, making them relatively easy to translate. Other protocols, on the other hand, use embedded IP information that NAT cannot translate. For example, the IPSec protocol running in Authentication Header mode is encrypted, so it impossible to even read its embedded IP information; IPSec in AH mode cannot be used in conjunction with NAT.

NAPT also relies exclusively on the information stored in the router's translation table. If the NAPT router rebooted for any reason (such as power outages or software failures), the table would be reset and all end-to-end connections would be dropped. The end effect on clients could range anywhere from a minor annoyance to a major inconvenience.

Chapter 3

Design

This chapter presents the architecture of MultiNAT, a service that exploits local access multi-homing but requires little client configuration. We describe the behavior of MultiNAT across a variety of scenarios and detail the changes made to the Roofnet Click implementation to support MultiNAT functionality. We also summarize the changes made to `libalias`, an open-source packet rewriting library originally written for FreeBSD.

3.1 System Architecture

We set out to provide users with a "plug-and-play" solution to exploit client multi-homing. The previously discussed MONET proxies provide a point solution for World Wide Web access, but suffer two drawbacks. First, they require explicit client configuration to use them, and second, they do not improve the resilience of other important protocols.

Our MultiNAT service functions in similar fashion to the MONET proxies: it attempts to connect to the destination host via all local links and completes the three-way handshake only on the link that connects fastest. Unlike MONET, MultiNAT requires little client configuration. If clients are already using some form of NAT, MultiNAT can simply replace their existing NAT service. Services such as DHCP (Dynamic Host Configuration Protocol) [17] can also help minimize the amount of

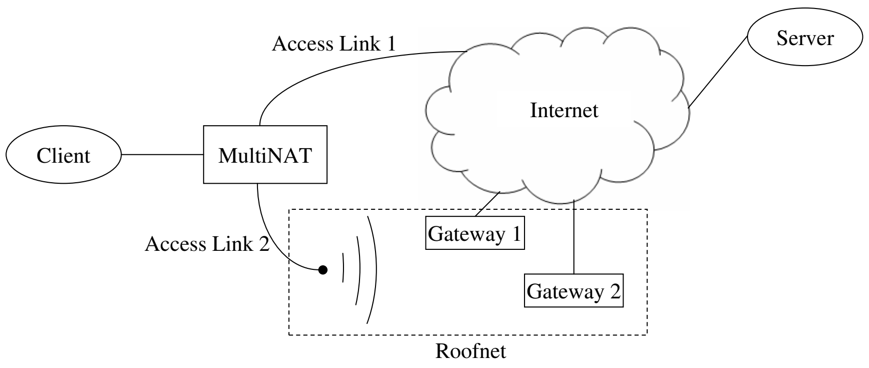


Figure 3-1: This diagram shows a typical MultiNAT configuration, with one wired link to the Internet as well as one wireless link (through Roofnet). Roofnet can route data through multiple gateways; in this particular setup, MultiNAT can take advantage of three separate paths to the destination server: the wired link, and one wireless path via each of the two gateways.

client configuration; DHCP automatically assigns IP addresses to clients and configures gateways and DNS servers. Switching to MultiNAT requires *no* configuration changes for most clients, and minimal work for the rest. Figure 3-1 shows a typical MultiNAT configuration.

Normal TCP connection establishment is a three-step process [31]. First, the client sends a `SYN` packet to the destination server. Next, the server replies with a `SYN ACK` packet, which indicates that the server is willing to establish a connection. Last, the client sends an `ACK` packet, which completes the handshake and finalizes connection establishment on the server. Either party can abort the process by replying with a reset (`RST`) packet at any step.

In MultiNAT, the initial `SYN` packet is forwarded via all the local interfaces, allowing the NAT [1] to concurrently attempt connection establishment via all local links. Figure 3-2 shows the normal scenario for MultiNAT connection establishment. The client first sends a `SYN` packet to the destination host. This packet is intercepted by MultiNAT, which forwards a copy over each local interface. The destination server, therefore, sees one attempted connection per local interface. When the server responds, MultiNAT processes the replies and passes the first received `SYN ACK` back to

[1]In the rest of this thesis, unless mentioned otherwise, we follow the convention of using the term "NAT" to mean "NAPT."

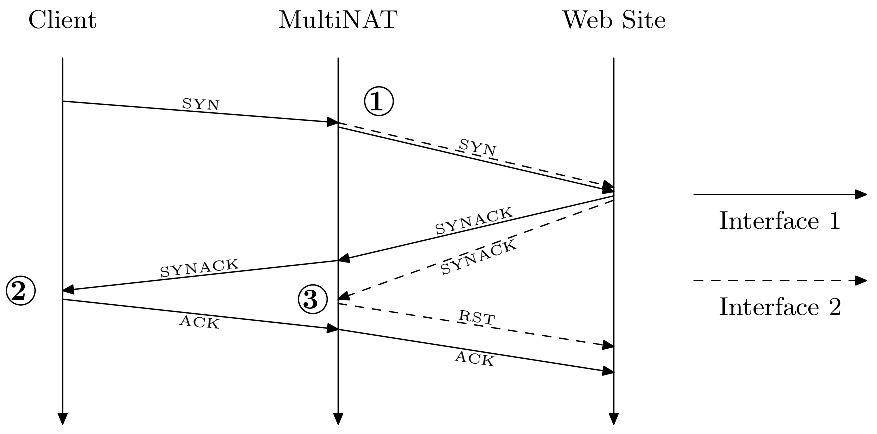


Figure 3-2: Steps in normal MultiNAT operation. 1) The initial SYN packet is forwarded via both local interfaces. 2) The first `SYN ACK` reply is returned to the client. 3) MultiNAT responds to further `SYN ACK`s by sending a `RST` packet.

the client. MultiNAT then sends `RST` packets for all other received `SYN ACK` packets; resetting on the other links avoids completing more connections than is necessary, which in turn reduces server load. MultiNAT effectively masks the details of which interface is used, so from the client's perspective, MultiNAT acts as a normal NAT daemon.

One of the goals of MultiNAT is to exploit client multi-homing to route around wide-area and server failures, so it has to be especially careful when dealing with server `RST` messages. Servers sometimes respond differently to different clients: specifically, a server can reject a connection from one set of IP addresses, but accept a connection from a different IP.

Figure 3-3 shows two scenarios in which the server sends resets. In both of these scenarios, the MultiNAT box has three local access links. In Figure 3-3a, the server sends a `RST` packet in response to each of the attempted connections. MultiNAT silently ignores the first two `RST` packets, because there is still a chance that another link could successfully connect. When the third `RST` packet arrives, however, this is no longer a possibility, and MultiNAT passes the `RST` packet along to the client.

In Figure 3-3b, the server replies with a `RST` on the first and third links but allows the connection to complete on the second link. As in the first scenario, MultiNAT

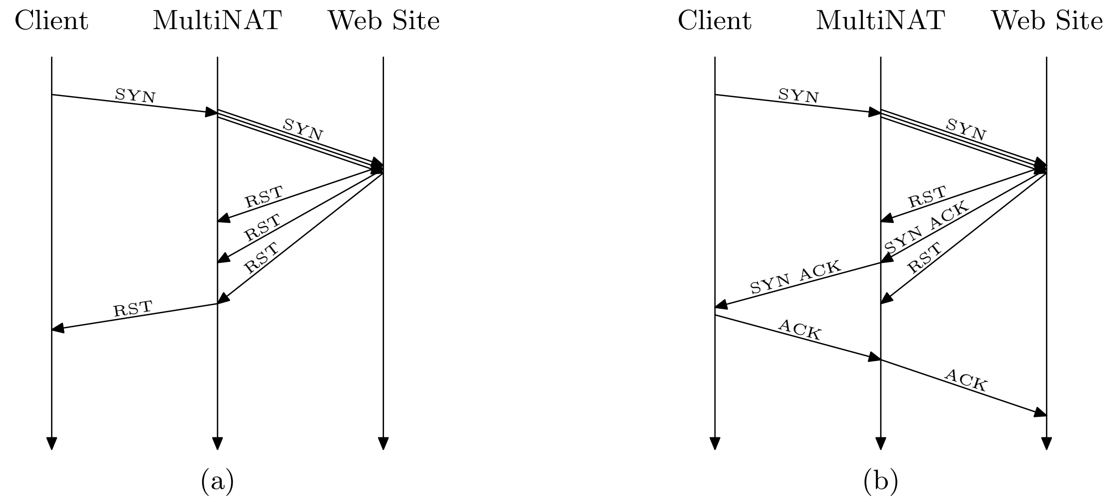


Figure 3-3: MultiNAT operation when the server sends `RST` packets. (a) The server sends reset packets on all links, so MultiNAT passes the last `RST` along to the client. (b) The server accepts the connection on one of the three local interfaces, allowing the client to successfully connect to the destination server.

silently discards the first `RST` packet, because there is still a chance that the other two links might successfully connect. When the `SYN ACK` arrives on the second link, MultiNAT passes the `SYN ACK` to the client, who responds with an `ACK` and completes the TCP handshake. Finally, when the `RST` packet arrives on the third link, MultiNAT recognizes that a connection has already been established. It checks whether or not the `RST` packet arrived on the same link as the established connection. In this case, it arrived on a different link, so MultiNAT again silently discards the packet. By behaving in this way, MultiNAT is able to increase its chances of successfully connecting to the destination server.

Our results in Chapter 4 show that connection times are not always reliable predictors of download times; when one link is consistently better than another, it is often helpful to bias slightly in favor of the better link. MultiNAT implements this biasing using delay queues; `SYN ACK` packets on a poorer link can be placed on a delay queue and processed later. The benefits of delay queuing are explained further in Section 4.4.

MultiNAT periodically polls the delay queue, removes `SYN ACK` packets that have been delayed the appropriate amount of time, and processes them as if they had just arrived. If the other link received a `SYN ACK` in the meantime, the queued connection is reset. Otherwise, the queued `SYN ACK` is passed on to the client.

3.1.1 NATD

NATD is an open-source NAPT implementation, originally written for and distributed with FreeBSD. NATD uses the `libalias` packet aliasing library, which provides all the basic functions of NAT [20], including support for multiple machines to share a single external IP address. Since some protocols internally embed IP addresses that also need to be translated [19], `libalias` includes hooks to call protocol-specific rewriter modules. The base `libalias` distribution provides rewriter modules for FTP and IRC, and it is easy to extend the program to support other protocols.

We chose to base our implementation on NATD in part because it is a user-level application, which makes it easier to deploy and test than the Linux kernel-space NAT implementation [3]. It is also released under a favorable BSD-style license, which allows us to distribute modified source code.

3.2 Roofnet Gateway selection

To function in Roofnet, MultiNAT requires the ability to specify the egress gateway for each connection. Unfortunately, the current Roofnet distribution does not provide this ability. Roofnet automatically routes packets through the gateway with the lowest ETT metric. This routing is done on a connection-by-connection basis; the routing decision is made when Roofnet transmits a `SYN` packet, and all other packets sent through that connection are routed through the same gateway. This approach allows Roofnet to respond to changing wireless network conditions without disrupting existing connections. Roofnet also allows a user to manually specify a gateway, in which case all traffic is routed through that gateway.

We chose to encode gateway selection information in the TOS bits of the IP header. This approach has two advantages. First, most operating systems supply a system call that sets TOS bits [34, 30], so this encoding provides a general mechanism by which any application can select its egress gateway. Second, this approach fits in well with our designed framework; the MultiNAT program need only change one byte in the IP header, allowing it select a gateway without affecting packet sizes.

The Roofnet software is written in Click, so we had to modify the Roofnet Click distribution to support TOS gateway selection. Within Click, the `GatewaySelector` element is responsible for marking packets with the appropriate gateway. We modified this element to examine the TOS bits before selecting a gateway and then clear the TOS bits afterwards. Since the TOS bits only comprise one byte of the IP header and IPv4 addresses are 32 bits long, the `GatewaySelector` element must also maintain a mapping of TOS bits to gateway IPs. Users can set and modify these mappings at run-time through the existing Click framework. Each mapping also has a valid bit; the `GatewaySelector` element defaults to the gateway with the lowest ETT metric if a particular mapping is invalid. In order to maintain backwards compatibility, a TOS setting of 0 automatically uses the lowest metric gateway, and a globally specified gateway takes precedence over a gateway chosen through TOS bits.

3.3 MultiNAT

Once we had decided how to manually select a gateway, we could begin work on the MultiNAT service. This section details the various changes made to NATD in order to support MultiNAT functionality.

3.3.1 Packet Aliasing

As described in Section 2.4, `libalias` works by rewriting the source address of outgoing packets and the destination address of incoming packets. The client thinks it has a connection to the destination server, and the destination server thinks it has a connection to the NAT middlebox. `libalias` stores the relevant information for each connection in a “link” structure and uses these links to rewrite packets.

Internally, `libalias` stores references to each link in two master link tables. The OUT table indexes links based on the client and destination IP addresses, and the IN

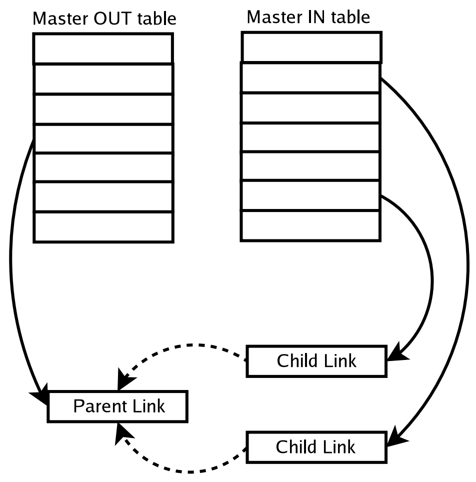


Figure 3-4: Diagram of the MultiNAT master link tables. The master OUT table contains parent links, while the master IN table contains child links, which in turn reference their parent links.

table indexes links based on the alias and destination IPs. When `libalias` receives a packet from the local network, it looks for the relevant link in the OUT table; likewise, queries on packets received from the Internet are performed on the IN table. When a new link is created, `libalias` inserts it into both the IN and OUT tables.

Links only store one alias IP and port, however, making them unsuitable for use in the MultiNAT framework. MultiNAT does not choose an alias address for a new connection until it receives a `SYN ACK`, so it is impossible to know where in the IN table to place a link before the `SYN` packets are even sent. To work around this limitation, we introduced a new type of link: the child link. The link created when MultiNAT processes a new connection is called a parent link; it contains the same information as the original `libalias` link, but the alias address is intentionally left blank. MultiNAT also creates several child links, one for each local interface, that all contain a reference to the parent. The parent link contains several additional fields that track the connection states of its children. Figure 3-4 shows the relationship between parent and child links.

When a packet is received from the Internet, the appropriate child link is retrieved from the IN table. Its parent is then located, and this parent link is used for the rest of the aliasing process. If this is the first child to return with a `SYN ACK` packet, the alias address field in the parent is filled in with the child's values. If a `SYN ACK` has already been received by another child, the alias address field is no longer blank, and MultiNAT knows to send a `RST` packet. Essentially, the child links function only as placeholders in the master IN table. All the important aliasing data is stored in the parent link.

Once MultiNAT receives a `SYN ACK` from the destination server and passes it on to the client, control is passed to the `libalias` packet rewriting routines. MultiNAT only changes the manner in which connections are established, so the main body of rewriting routines is unchanged. For packets that are not part of the TCP handshake, the only change is that the parent link is used in place of the child link.

After a connection is established, the child links are left in the master IN table to await `SYN ACK` arrivals on the other local interfaces. As described in Section 3.1, only one `SYN ACK` packet is passed to the client; MultiNAT must send resets on the other connections in order to reduce server load. When MultiNAT receives a `SYN ACK` packet, it checks the parent link to see if this is the first successful reply. If it is not, control is passed to a new routine that builds and sends a `RST` packet. The appropriate TCP sequence and acknowledgment values are taken from the received `SYN ACK` packet, and the `RST` packet is immediately sent to the destination server. This effectively closes the connection on the server.

3.3.2 Packet Delay

A key feature of MultiNAT is its ability to delay packets for future processing. At any point in the packet aliasing routines, MultiNAT can flag a packet to be queued. MultiNAT then periodically checks the delay queue to see if any packets are ready to be processed, in which case they are removed from the queue and re-sent through the aliasing routines.

We made two key assumptions that allowed us to simplify our implementation of the delay queue, at the cost of decreased flexibility. First, we restricted the delay queue to hold only `SYN ACK` packets, ensuring that all the packets in the delay queue would be less than 60 bytes in length. Second, we do not allow arbitrary delays; all packets in the queue are delayed for the same amount of time.

These simplifications allowed us to implement the delay queue as a ring buffer of 60-byte packets. Because all packets are delayed by the same amount, the queue is guaranteed to always be in sorted order by time; packets come out of the queue in the exact same order as they went in. Using an array also eliminates the need for dynamic memory allocation; we designed this service for use on the typical Roofnet machine, which has limited memory and processing power.

When a new packet is sent to the delay queue, it is added to the next free spot in the buffer. If the queue is full, MultiNAT processes the packet immediately. Because the queue is always in sorted order, removing packets from the queue is simple as well. MultiNAT periodically checks the queue to see if any packets are ready to be sent. If so, MultiNAT starts at the current head of the queue and walks down the array, processing packets if they have already been delayed for the correct amount of time. Once MultiNAT finds a packet that is not ready to be sent, it resets the `head` pointer to point to this packet and stops searching for packets.

The current MultiNAT implementations back their delay queues with a thousand-element array, which requires only 60 kbyte of memory. With a 5-second delay and two local interfaces, 200 connections can be established per second without any packet drops. The Roofnet median is around 100 connections over a 24-hour period [10], so the delay queue should not fill up under normal circumstances.

3.3.3 Packet Diversion

The existing NATD implementation for FreeBSD uses divert sockets to transfer packets from the kernel into a userspace application. Divert sockets have been ported to Linux in the past, but implementations are not available for recent kernels, so we had to find a different solution to run on Roofnet nodes. Our Linux MultiNAT implementation uses `libipq` to pull packets from the kernel to userspace and back. `libipq` functions similarly to divert sockets, though it is more limited. For example, only one userspace application at a time can register to receive packets.

One problem that is common to both `libipq` and divert sockets, however, is the fact that only one packet can be injected back into the kernel for each packet that is taken out. This presents a problem for MultiNAT, which needs to send out multiple SYNs to the destination host for each SYN it receives from the client.

To work around this limitation, we send the duplicate SYNs through raw sockets. Using raw sockets allows us to directly modify the IP header of a packet as well as send packets to a destination server without first connecting to that server. Because we could not get the Linux kernel to automatically route packets out the correct interfaces, we create and bind one raw socket to each local interface. Packets leaving MultiNAT are directed to the appropriate raw sockets, and from there they are routed out the correct interfaces.

3.4 Performance

To test the maximum throughput and expected latency of MultiNAT, we used the `ttcp` utility [26] to measure transfer speeds between two machines on a 100 mbit/s network. We tested throughputs both with and without a MultiNAT middlebox, and both machines used 50 kbyte TCP send and receive buffers in order to maximize TCP performance. We found that the machines were able to sustain a 4700 kbyte/s transfer rate when directly connected. This speed dropped to 4400 kbyte/s when MultiNAT was used, from which we infer that the maximum transfer speed sustained by MultiNAT is 4400 kbyte/s. We also tested the latency overhead of MultiNAT by using ping to measure round-trip times between the two machines. We saw average times of 0.8 ms for directly-connected machines, which increased to 0.95 ms when MultiNAT was introduced.

Overall, MultiNAT is able to sustain transfer speed in excess of 4 mbyte/s while only imposing 0.15 ms latency overhead. Given that Roofnet gateways transfer only 160 kbit/s on average and that most residential DSL connections are limited to 3 mbit/s downstream, MultiNAT is more than able to handle typical loads.

Chapter 4

Evaluation

The goal of this chapter is to analyze the performance of various link-selection schemes in a variety of network configurations. We show that while MONET predictions work well in wired networks, the unpredictability of wireless losses leads to disappointing performance in wireless networks. We present a new prediction scheme, AvMet, that outperforms existing predictors in all network configurations. AvMet also increases availability in wireless networks by half an order of magnitude. This chapter concludes by presenting a few best-case scenarios for AvMet and MONET.

4.1 Experimental setup

We collected experimental data at three sites with different network configurations: one had two wired links (wired+wired), another had a number of nodes with two wireless links (wireless+wireless), and the last had one of each(wired+wireless). Figure 4-1 shows our experimental setup for a wired+wireless configuration. For each data set, we used a Perl script to fetch a set of 10,000 objects through each of two different links to the Internet; multiple data sets were collected at each site. These 10,000 URLs were taken from a trace of user activity logged on a MONET proxy–we felt that the tests should mimic user behavior as closely as possible. This method is different from the testing methods used by Yip in NATRON [35], who fetched only one object from each server in order to avoid over-representing working sites. We

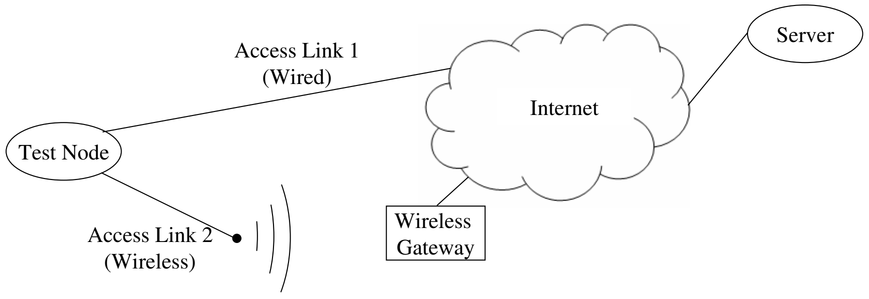


Figure 4-1: Network topology for a typical wired+wireless test. The test node fetches each web object through both the wired and the wireless links. We collect connection times and total download times for each fetch, which allows us to later analyze various link-selection schemes.

filtered out any dynamically-generated content to ensure that the same object was fetched over each link, because some sites may deliver different content based on the IP of the client or the time of day.

We collected wireless+wireless data on a number of Roofnet nodes over a two-week period in May 2005. We chose Roofnet nodes that had similar metrics to multiple gateways as well as independent multi-hop paths to those gateways. We used a machine at MIT that had a supplemental DSL line as a testbed to collect wired+wired data over two separate one-week periods, one in May 2005 and one in June 2005. In February 2005, we collected wired+wireless data at an apartment in Pittsburgh; this data set compared a 6-megabit DSL line and a wireless connection to CMU.

Our testing script recorded the start and end times for each fetch, as well as the time needed to establish the connection (the time between sending a `SYN` and receiving a `SYN ACK`) and the total number of bytes retrieved. One server was offline during the entire testing period, so requests involving that server were removed before analysis. We also threw out any requests where each link retrieved a differently-sized object; since each link returned a different number of bytes, the download times could not be compared. Although this was a rare occurrence, several fetches over Roofnet timed out and returned HTTP Request Timeouts, which our Perl script could not otherwise distinguish from successful completion.

While all of our data sets fetched all the objects over every link, there were some slight differences in testing methodology based on the type of the links being tested. These differences are outlined below:

1. *Wired+Wired and Wired+Wireless:* Tests involving at least one wired link naturally had two independent paths leaving the test node. In these cases (the MIT+DSL and DSL+CMU wireless tests), we fetched objects via both links simultaneously. This approach assumed that fetching the objects concurrently would not cause a bottleneck on the server's access link.

2. *Wireless+Wireless (Roofnet)* The tests performed on Roofnet presented an interesting challenge. Since all transmissions share a common first wireless hop, fetching objects simultaneously via both links would create artificial interference that could affect our results. To deal with this situation, we fetched the object first via one link, then via the other. The first fetch was timed out after three minutes and the second fetch started after that, in order to minimize the chances that wireless link conditions had changed. We also randomly selected which gateway was used first, which prevented us from giving an unfair advantage to any one gateway.

4.1.1 Metric analysis

Once we collected the download and connection time data, we used it to analyze the performance of various prediction metrics. These metrics take as input the connection times for each link and attempt to predict which link will fetch each object the fastest. The evaluated metrics are summarized below; the first three apply to all network configurations, while the last two are specific to multi-hop wireless networks.

1. *MONET (race)*: Fetches via whichever link connected first. This is slightly different than the waypoint selection scheme presented in [7], in that it does not prune the list of possible paths, instead trying all paths. MONET is described in more detail in Chapter 2.

2. *AvMet (history)*: Keeps a historical average of connection times. Averaged over a long enough timespan, AvMet is able to determine which link is better. AvMet is described in more detail in Section 4.4.

3. *Ideal*: The overall shortest download time for each request. The Ideal metric provides an upper bound on possible performance gains. Because the wireless+wireless tests fetched objects via different gateways sequentially rather than concurrently, it may not be possible to achieve ideal performance in wireless networks, even with a perfect predictor.

4. *ETT (Roofnet)*: ETT calculates the expected time required to transmit a 1500-byte packet to a node. Roofnet fetches via the gateway with the lowest ETT metric. ETT is described in more detail in Chapter 2.

5. *Static Best Gateway*: The single overall best gateway. This metric does not make any predictions on a connection-by-connection basis, but mimics what would happen if a user manually configured his gateway.

4.2 MONET predictions in wired+wired networks

The MONET proxies [7] use a simple race-based prediction scheme to select between links. This scheme assumes that connection times and download times are correlated; in other words, the link with the fastest connection time will also have the fastest download time.

To corroborate the MONET findings, we collected data for 45,636 requests, fetching each object concurrently over MIT's connection to the Internet as well as through a supplemental DSL line. Figure 4-2 is a CDF of the fraction of objects fetched versus total time. Lines are shown for each individual link (MIT and DSL), Ideal, and for the MONET predictor scheme. Data points where neither link could connect to the server were excluded.

We found that while a race-based prediction scheme does not achieve optimal performance, it still provides a benefit over using a single link. The MIT link was

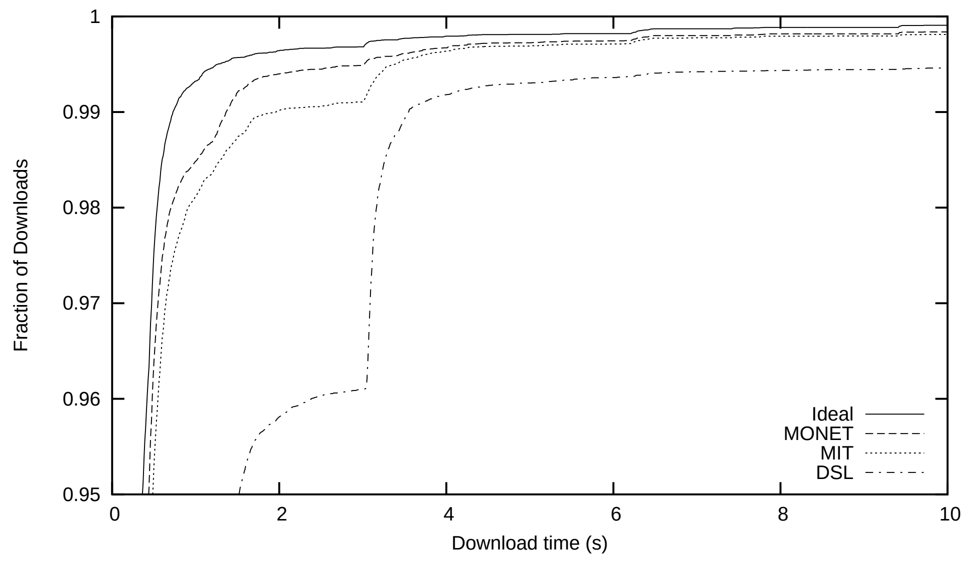


Figure 4-2: Upper 5% of a CDF of download times, comparing MIT's connection to the Internet and a secondary DSL line. The data shows that by using the MONET predictor, a cheap DSL line can successfully augment a much higher-bandwidth link.

able to fetch 99.0% of the objects in under 2 seconds. By using the both the MIT and DSL links together, the MONET predictor was able to improve this figure to 99.4%. The Ideal predictor, in comparison, was able to fetch 99.6% of the objects in under two seconds, meaning that the MONET predictor realized over 50% of the possible performance gain.

A benefit of using MONET is the improvement it is able to provide for these shorter transfers. Figure 4-2 shows that almost 99% of transfers complete in under 2 seconds. On the other hand, when a TCP SYN packet is lost, TCP waits for a full three seconds before retransmitting. Therefore, when a packet is lost during connection establishment, it is almost always worth fetching the object over the secondary connection. Even though the supplemental line has less capacity, it can often completely fetch the object before TCP re-attempts connection establishment on the primary link, as shown by the higher performance of the MONET predictor between 1 and 3 seconds. From the user's perspective, MONET is able to reduce long delays that interrupt normal browsing.

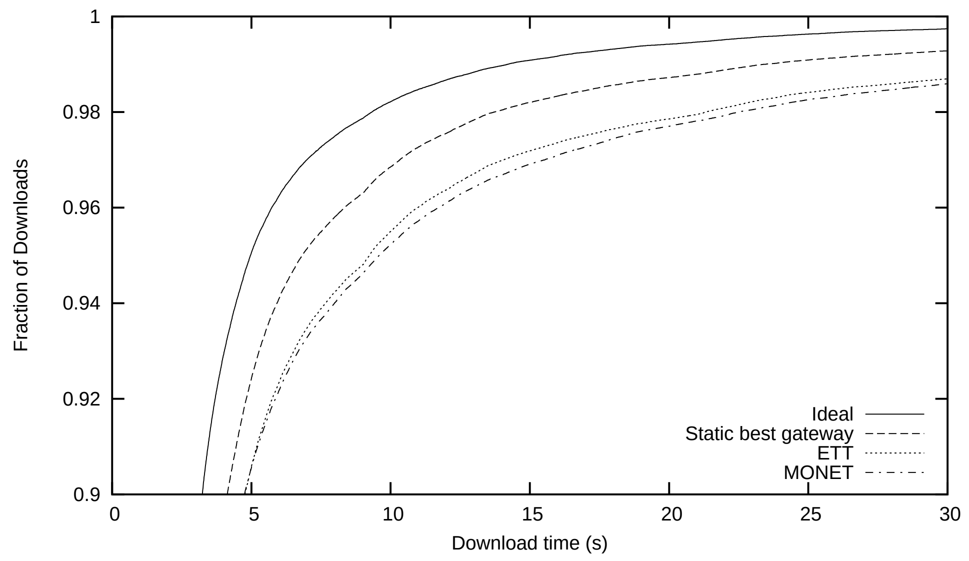


Figure 4-3: Upper 10% of a CDF of download times observed on Roofnet, comparing the performance of an MIT gateway and a DSL gateway. Note the difference in timescales between this and Figure 4-2, indicative of the much poorer overall performance of this wireless network. Disappointingly, neither the ETT or the MONET predictors were able to match the performance of the static best gateway.

4.3 Mispredictions in wireless+wireless networks

While the MONET prediction scheme is able to improve the performance and resilience of wired networks, its performance is disappointing when it is applied to wireless+wireless configurations. The bursty nature of wireless losses is primarily to blame; it is difficult to tell whether a slow connection time is caused by a generally poor wireless link or momentary interference in an otherwise high-performing link. Wireless prediction schemes such as ETT suffer from a different set of problems. Since they were designed to find high-throughput paths in multi-hop wireless networks, they do not take into account the potential differences in the wired path outside the wireless network. They may therefore find a very fast wireless path that leads to a very slow wired Internet connection.

4.3.1 Race mispredictions

Figure 4-3 shows the aggregate results from tests performed on 10 Roofnet nodes. On each node, our testing script fetched up to 10,000 objects through two gateways, as described in Section 4.1. Some tests were truncated by node reboots, and therefore fetched fewer than 10,000 objects. One of the test gateways was located on the MIT campus and used MIT's main connection to the Internet. The other was located at a Roofnet user's home and had access to a DSL line. As in Section 4.2, we discarded data points where neither link could connect to the destination server. Lines are drawn for the MONET predictor, Roofnet's ETT predictor, the static best gateway, and an upper bound on possible performance gains. This graph shows that MONET-style race predictions do not work well in multi-hop wireless networks; in fact, in this scenario they perform slightly worse than the existing Roofnet metric.

We found that the static best gateway is the best performing of the three metrics. The better gateway was able to fetch 92.4% of the objects in under 5 seconds, compared to only 90.6% for both the ETT and MONET predictors. When given 10 seconds, the static best gateway fetched 96.8% of the objects, compared to 95.5% for the ETT predictor and 95.2% for MONET.

This is in stark comparison to the wired+wired performance examined in Section 4.2, where the MONET predictor was able to improve performance significantly. This discrepancy suggests that some property of wireless networks is causing connection times to provide us with misleading information. To confirm that the mispredictions seen here were indeed affected by the wireless portions of the path, we used `tcpdump` to log all SYN packets going through the two gateway nodes. This allowed us to calculate round-trip times from the gateway node to the host server, giving us an idea of how well the wired portions of the path perform.

Figure 4-4 shows gateway-server round-trip times and client-server connection times for objects fetched via both the MIT and DSL gateways. Both gateways show sharp TCP "knees" at 300 ms, which is similar to the wired performance observed by Andersen *et al.* [7]. The round-trip times observed by the Roofnet gateways are

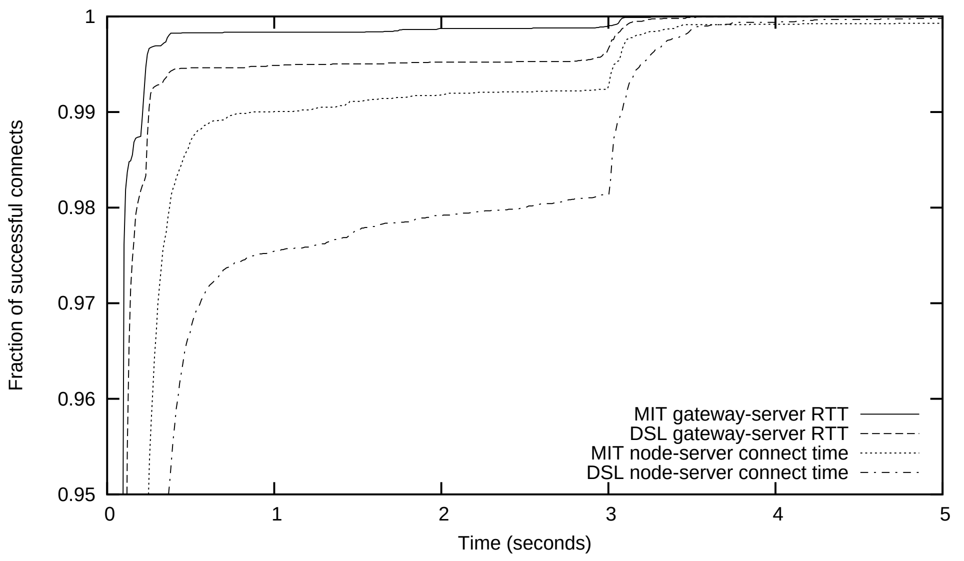
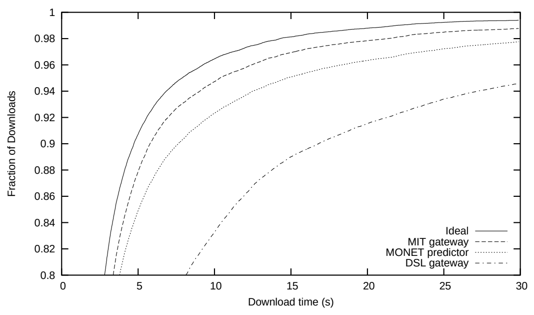


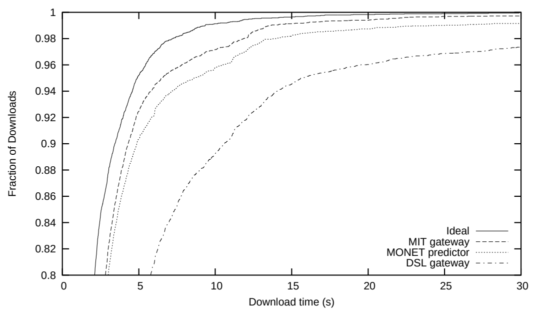
Figure 4-4: Comparison of connection times as seen by a client Roofnet node and as seen by a gateway node. The wired performance of the gateways in similar to the performance seen in our wired+wired tests, implying that the MONET mispredictions are being caused by some property of wireless networks.

also consistent with the MIT and DSL link performance observed in our wired+wired tests. Since performance in the wired portions of the path is on par with previous wired+wired measurements, we can infer that wireless losses are causing MONET to mispredict.

The unpredictability of wireless losses (and the resulting poor performance of TCP over wireless links) is a well-documented fact [16, 9, 12, 8]. Interference from outside sources is often bursty and short-lived, so it can be difficult to judge the quality of a link based on the round-trip time of a single packet. If an otherwise high-quality link experiences a momentary spike in packet loss due to interference, MONET will interpret a lost SYN packet as evidence that the link is unreliable. At the opposite extreme, a short-lived improvement in a low-quality link can convince MONET that the link is much better than it actually is. Further exacerbating the situation is the fact that SYN packets occupy only 40 bytes, while the average HTTP response packet is at least ten times larger. The larger data packets have higher loss rates, which implies that small SYN probes do not provide accurate data on wireless link



(a)



(b)

Figure 4-5: (a) Gateway performance when the MIT gateway had the lowest metric. (b) Gateway performance when the DSL gateway had the lowest metric.

quality. These findings indicate that an effective metric should average link quality data over time, rather than make decisions based solely on a single data point.

4.3.2 ETT mispredictions

Figure 4-3 also reveals that neither the race-based predictor nor Roofnet's ETT metric are able to come close to the performance of the single better gateway. One major limitation of the ETT metric is its focus on wireless link performance. ETT was designed to find the best path between two nodes in a multi-hop wireless network, and research has shown that it does this relatively well [10]. However, in our Roofnet test cases, a significant portion of the path between a node and a host server exists outside of the multi-hop wireless network, and ETT is unable to see any problems that may exist in this wired portion of the path.

Unfortunately, ignoring wired performance alone doesn't explain the poor performance of ETT. Figure 4-5 shows data collected on one Roofnet node, separating data points based on which gateway had a lower metric. We have already shown that the wired performance difference between the MIT gateway and the DSL gateway is small compared to the overall end-to-end difference, so one would expect the lower-metric DSL gateway to outperform the MIT gateway in Figure 4-5b. However, the data shows the MIT gateway to still be the better choice. In fact, overall gateway performance for this data set is *largely unaffected* by the relative ETT metrics of the two gateways.

To confirm this finding, we performed a series of downloads between Roofnet nodes, logging the current ETT metric and data transfer rate for each download. The test script fetched a 1 mbyte file, randomly generated to reduce the benefits of compression, from webservers hosted by each gateway node. The transfers were contained entirely within the Roofnet network, so they were unaffected by the wired performance differences of the gateways. Table 4.1 summarizes the results.

The first two entries show the average download speeds seen by a node that had roughly equal metrics to both test gateways. Even though the ETT metric estimated that the wireless links were identical, the test node was able to fetch objects three times faster from the MIT gateway. The metrics to each gateway also varied between 6000 μs and 8000 μs while the tests were performed, but transfer rates did not vary much from the averages shown. After speaking with the Roofnet developers [11], we learned that since the ETT metric uses only periodic broadcast probes to estimate link quality, it does not perform as well under heavy load. In this case, the wireless connections to the two gateways looked identical during periods of inactivity, but the link to the MIT gateway performed significantly better when loaded.

The last two entries in Table 4.1 illustrate another limitation of the ETT metric. We performed two sets of downloads between two test nodes and the MIT gateway. Both test nodes had average ETT metrics of 2000 μs, but saw widely disparate transfer speeds. Further investigation revealed that the problem was most likely a software misconfiguration on the test nodes. ETT has no way to include software misconfigurations in its estimations, and therefore was not able to capitalize on the higher transfer speeds of node 5.22.131.90.

These findings imply that neither the ETT metric nor connection latency can be relied upon as the sole source of data when trying to predict download times. The data collected on Roofnet shows that wireless losses dictate which connection is better, and the link with less wireless loss should be used in almost all cases, failing over to the second link only in extreme cases.

Test Node	Gateway	Avg. ETT Metric	Avg. Transfer Rate
5.5.92.185	MIT	6000-8000 μs	140 kbyte/s
5.5.92.185	DSL	6000-8000 μs	45 kbyte/s
5.22.131.90	MIT	2000 μs	500 kbyte/s
5.4.102.95	MIT	2000 μs	150 kbyte/s

Table 4.1: Average ETT metrics and download speeds (in kilobytes/sec) as measured between test nodes and Roofnet gateways.

4.4 AvMet: A history-based prediction scheme

We set out to devise a prediction scheme that would take advantage of the end-to-end benefits of the MONET-style predictor but not be misled by the false information that connection times can sometimes offer. We noted that while connection times do not always provide useful information on a connection-by-connection basis, they still perform better than random. Therefore, connection times can positively identify the better of two gateways when averaged over a long enough time span.

We propose AvMet, a history-based scheme that is tuned to perform well in wireless networks. AvMet uses connection latency to identify the better gateway, but is not swayed by transient changes in wireless quality. AvMet requires only a single piece of state: a floating point variable *curbest*, which stores the exponential weighted moving average (EWMA) of which connection finished fastest. This variable is calculated as follows:

$$curbest_{new} = \alpha * curbest_{old} + (1 - \alpha) * link$$

where α is the exponential coefficient and *link* is either 0 or 1, depending on which link connected fastest. If $curbest < 0.5$, AvMet believes that link 0 is the better link; if $curbest > 0.5$, link 1 is the better link.

When AvMet receives a `SYN ACK` packet, it updates *curbest*, then takes appropriate action based on which link received the `SYN ACK`. If the fastest connection time registered on the current best link, AvMet completes the connection on that link and fetches the object. If the other link connected fastest, AvMet delays processing the `SYN ACK`, giving the connection a chance to complete on the better link.

4.4.1 Customization

AvMet can be customized to specify its responsiveness to network condition changes. The EWMA coefficient, α, determines how many samples are included in the *curbest* average. Using a smaller coefficient allows AvMet to respond faster to changes in link performance, but also increases the likelihood that AvMet will choose the poorer gateway. In scenarios where connection times poorly predict download times, a large coefficient should be used: the results calculated for Roofnet used a large coefficient of 0.999 (1,000 samples).

It can be helpful to view this coefficient in terms of time, rather than as a number of samples. Our testing script fetched 10,000 objects in three hours, an average of about one object per second. Therefore, a coefficient α of 0.999 will keep a running average of about 20 minutes of link performance, while a coefficient α of 0.99 will look back 1 or 2 minutes.

The amount of time delayed must also be tuned to match the network setting. Most connections in wired networks complete within a few hundred milliseconds [7], and from Figure 4-2 we see that 99% of downloads complete within 3 seconds. A connection time of more than a few hundred milliseconds indicates that the `SYN` (or `SYN ACK`) was most likely lost, and therefore the object would be best fetched through the other link. In this scenario, delaying for more than the few hundred milliseconds does not do any good, since if the `SYN ACK` has not returned by then, it very likely will not return until after TCP resends the `SYN` 3 seconds later.

However, because wireless losses are not predictable, a longer timeout is needed in wireless networks. We found through experimentation that a delay of 4.5 seconds gives the best overall results. In other words, a total of two SYN packets must be lost on the primary link before it becomes worthwhile to switch to the backup.

4.4.2 Performance gains

Figure 4-6 shows the results of 54,775 fetches on Roofnet. The results indicate that AvMet performs noticeably better than the Roofnet ETT predictor. More significant

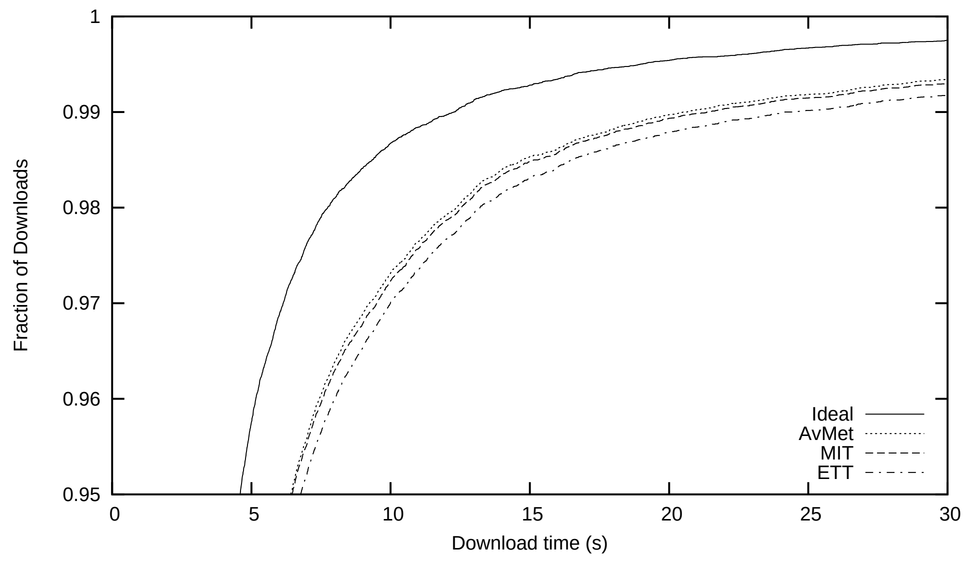


Figure 4-6: Linear graph of data collected on Roofnet. The AvMet predictor has better performance than the static best gateway (in this case, MIT), showing that it is not swayed by incorrect race predictions.

is the fact that, unlike the MONET predictor, AvMet is not affected by misleading connection times. AvMet is able to glean useful information from historical trends and use it to improve performance beyond that of the static best gateway.

The delay that AvMet imposes before switching to the poorer link is the main factor that sets its performance apart from MONET's. With a 4.5 second delay, AvMet will not fetch via the second link until at least two SYNs have been lost on the primary link. The loss of two SYN packets is indicative of a serious problem; two situations in which this could occur are extremely poor wireless conditions from the node to the gateway and a failure in the wired portion of the path from that gateway to the host server. In any case, the loss of two or more SYNs on the primary link is enough evidence to infer that there is a problem with that link, and in these cases, AvMet fetches on the secondary link.

Figure 4-7 focuses on metric performance for longer transfers. AvMet converges on static best gateway performance for transfers lasting longer than 30 seconds, but it is still able to provide higher resilience than either MONET or ETT. Overall, the ETT metric was able to fetch 99.69% of the objects in under three minutes. MONET

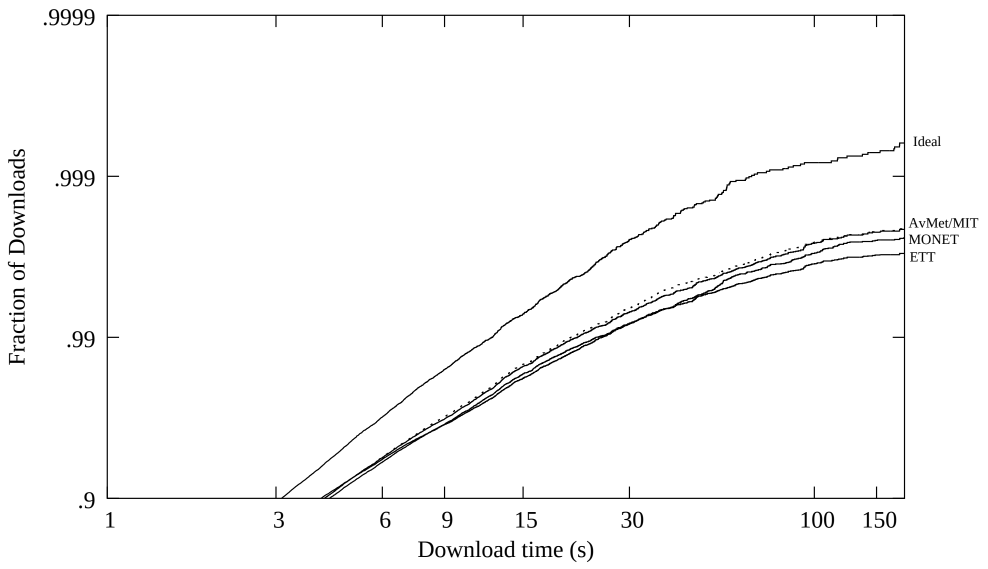


Figure 4-7: Logscale graph of data collected on Roofnet. The AvMet predictor is able to improve on the ETT metric in terms of availability.

was able to do slightly better, successfully fetching 99.75% of the objects. Static best gateway and AvMet were able to improve availability even further, fetching 99.78% and 99.79% of the objects, respectively.

Though the performance gains over the single better link are meager, the main benefit of AvMet is its ability to provide the performance of the static best gateway without knowing in advance which gateway that is. AvMet provides a configuration-less solution to the gateway selection problem; one only needs to start it up and the algorithm will automatically select and use the better gateway.

4.4.3 Performance in wired networks

We also evaluated the performance of AvMet on the wired+wired data from Section 4.2. Yip found in his Masters thesis that it is often only worth fetching an object through an alternate path if the connection time on the alternate path was more than 20 ms faster [35], a finding that fits in well with the AvMet framework. We analyzed the data using AvMet, though trial-and-error analysis found that delays of 50-100 ms provide the best performance. Figure 4-8 shows the effect of using AvMet with 100 samples and a 100 ms delay.

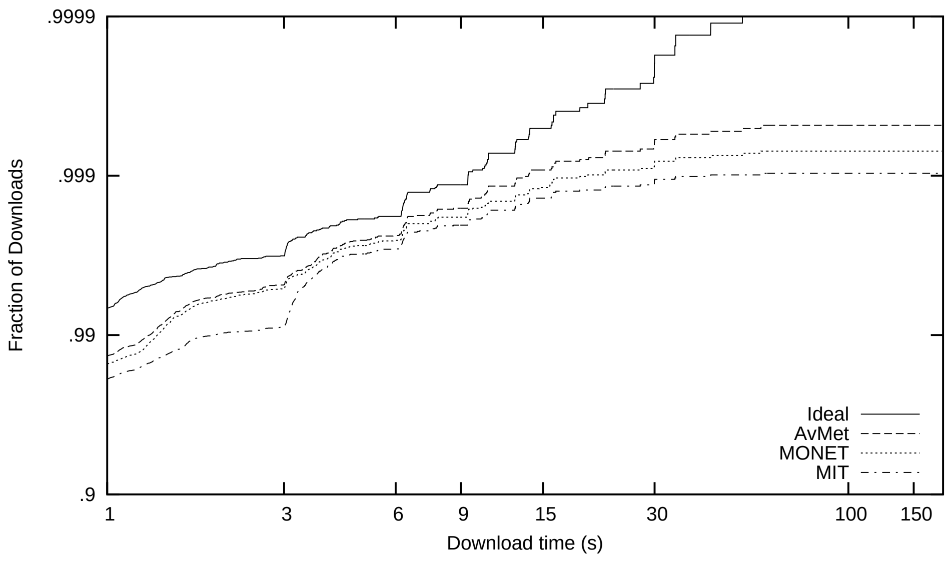


Figure 4-8: Logscale graph of wired+wired data collected at MIT. AvMet is able to improve server availability over single-link performance by half an order of magnitude.

The smaller number of samples used here allows AvMet to deviate more from the static best gateway. In this case, since connection times do perform well as download time predictors, the difference is for the better. AvMet is able to fetch 99.43% of the objects in under 2 seconds, compared with only 99.39% for MONET.

By attempting to connect on both links simultaneously, MONET is able to increase availability. Figure 4-8 shows that the MIT link was able to completely fetch 99.90% of the objects in under three minutes, and MONET was able to improve this to 99.93%. These numbers corroborate our earlier experience with MONET [7]; we found that the MIT link achieved approximately 99.8% availability and exploiting local access multi-homing increased availability to 99.93%.

Because AvMet biases slightly towards the best average link, it is able to improve wired availability even further. AvMet fetches over 99.95% of the objects that it is possible to fetch, providing half an order of magnitude improvement over single-link performance.

4.5 Anecdotal Scenarios

The wireless+wireless results presented so far were derived from combining all the data that we collected on Roofnet. There is much more variation on a case-by-case basis. This section presents some best-case scenarios for the AvMet and MONET predictors, illustrating pathological failures which these predictors are able to overcome.

4.5.1 End-to-end benefits of MONET and AvMet

As mentioned in Section 4.3.2, one of the limitations of ETT is its exclusion of wired performance data. This limitation became apparent in tests that included a faulty gateway. This gateway, located on the MIT campus, was the lowest metric gateway for a number of nodes. However, due to a combination of factors, this gateway had poor end-to-end Internet performance. The antenna for this node was, in the words of one Roofnet developer, “flung out of a window” [11]. This deployment caused its wireless link quality to fluctuate wildly at times. The node also experienced intermittent periods of extremely high wired loss. In fact, this particular gateway behaved so erratically that it was subsequently removed from the Roofnet network.

Figure 4-9 shows the results from a series of downloads comparing this faulty gateway and a DSL gateway. The faulty MIT gateway had a lower ETT metric than the DSL gateway for 66% of the requests, causing ETT to try and fetch over the faulty gateway the majority of the time.

This scenario is the poster child for AvMet’s end-to-end checks. Because the ETT metric only looked at wireless link performance, it was unable to account for the losses in the wired path from the gateway to the destination server, a shortcoming that is displayed in ETT’s poor performance. The MONET predictor was able to derive some information on these wired losses and perform better than the ETT metric, but it still could not match the performance of the static best gateway. AvMet, on the other hand, was able to match static best gateway performance for shorter transfers and exceed it for longer transfers.

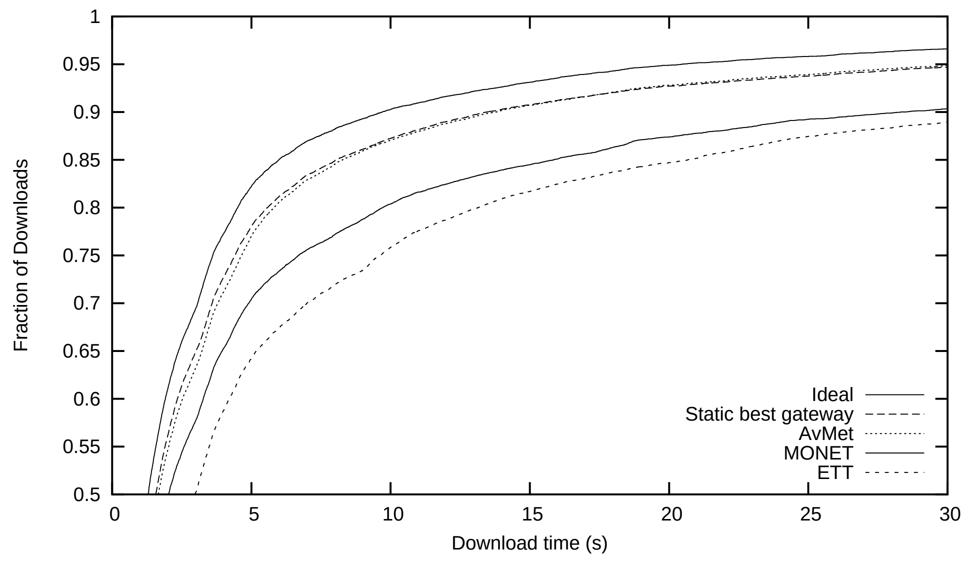


Figure 4-9: The MIT gateway tested in this data set was faulty and had extremely high loss rates. ETT did not pick up on these losses, so it performed poorly. AvMet correctly identified the better gateway, giving it the best overall performance.

Although this anecdote may seem like a contrived scenario, it is important to realize that networks like Roofnet rely almost exclusively on their communities to provide gateways to the Internet. It is possible for a Roofnet gateway node to be online while its connection to the Internet is down. This node would still continue to advertise itself as a gateway, so without the end-to-end checks of AvMet and MONET, nearby nodes could potentially send their data into oblivion.

4.5.2 Single-hop wireless performance

Figure 4-10 shows predictor performance in a wired+wireless (single-hop wireless) network configuration. We fetched 212,000 objects concurrently though a 6 mbit/s DSL connection to the Internet as well as a wireless connection to the CMU network. The results indicate that the MONET predictor works relatively well in this scenario; it approximately matches the performance of the better link (DSL). AvMet is able to improve performance even further, providing a hundred millisecond advantage on average over the DSL link alone. The benefits provided by AvMet and MONET are much lower here than in the wired+wired case. Much of this is because the wired

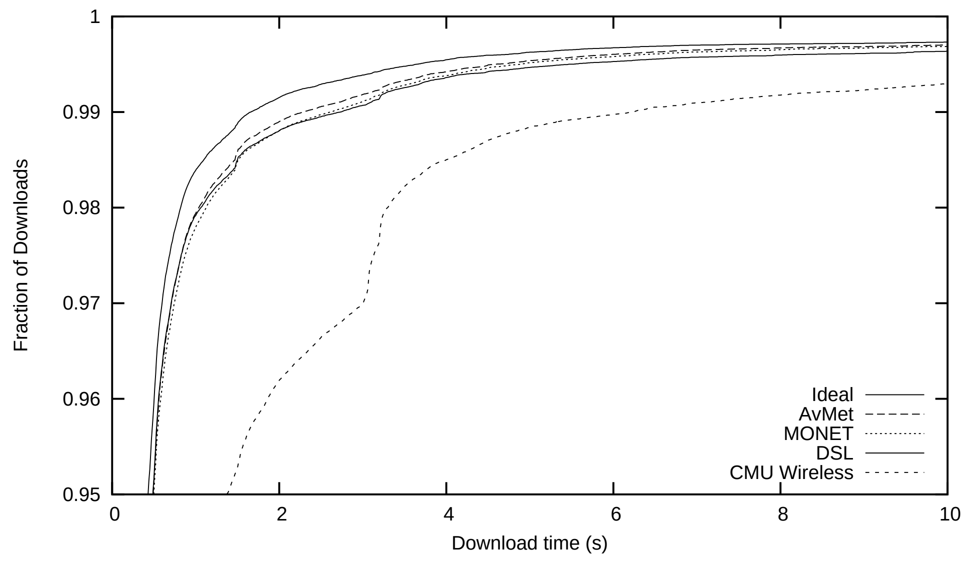


Figure 4-10: Upper 5% of a CDF showing predictor performance in a wired+wireless network. Because the wired link significantly outperforms the wireless link, the possible performance gains are much smaller in this network configuration. All predictors do equally well, roughly matching the performance of the better link.

DSL link is significantly better than the wireless CMU connection. Looking at the Ideal line, we can see that the possible performance gains are much smaller in this scenario as well. Overall, both the MONET and AvMet predictors work well in a wired+wireless network, but they do not provide much benefit over using the wired link alone (especially when using a high capacity DSL line).

4.5.3 Uploads

One of the areas in which Roofnet shines is transfers from Roofnet nodes to Internet hosts. The vast majority of residential ISPs cap upload speeds at around 48 kbyte/s (384 kbit/s). This upload bandwidth cap is the main bottleneck seen when uploading files to a remote host. On the other hand, the Cambridge Roofnet network contains several gateways located on the MIT campus, none of which have these upload restrictions. If a node is relatively close to an MIT gateway, it would most likely see faster transfer speeds by routing all upstream transfers through the MIT gateway, regardless of which gateway has the lowest ETT metric.

Recent studies on Roofnet have shown that 60% of node pairs on Roofnet are able to achieve throughputs higher than 400 kbit/sec [10]. Out of the 33 non-gateway nodes studied, 32 were 4 hops or closer to a gateway. The seven nodes that were exactly four hops away from a gateway saw an average throughput to that gateway of 379 kbit/sec. This suggests that if a node is 4 hops or closer to an MIT gateway, it should route all uploads through the MIT gateway, regardless of its ETT metric. This upload phenomenon has already been noticed and exploited by the Roofnet community; BitTorrent users account for 30% of all Roofnet traffic [10].

Chapter 5

Conclusion

This thesis described the MultiNAT application and AvMet link-selection scheme, which work together to provide better Internet performance and greater end-to-end Internet availability. MultiNAT differs from previous solutions in that it requires few configuration changes for most clients; they simply need to install MultiNAT on their primary gateway machine.

In order to effectively take advantage of client multi-homing, MultiNAT needs a prediction scheme that dictates which link to use at what times. The difficulty arises in finding a prediction scheme that works in all network configurations; wired and wireless networks suffer from extremely different modes of failure, and therefore an effective solution for wired networks may be ineffective in wireless networks.

We analyzed the performance of several existing prediction schemes and found that while they realized up to 50% of possible performance gains in wired networks, they did not translate well to the wireless landscape. The MONET scheme, which uses connection times as its main source of data, was often misled by the unpredictable nature of wireless loss. The ETT metric, which was designed to find high-throughput paths in multi-hop wireless networks, could not provide an effective end-to-end estimate of link quality.

We developed AvMet, a new link-selection scheme that tracks historical connection latency trends and uses them to provide better future predictions. We evaluated AvMet's performance over a wide variety of network configurations, including test nodes with two wired links, two wireless links, or one of each. We found that AvMet was able to outperform the existing MONET and ETT predictors in all network configurations. AvMet was able to take advantage of the end-to-end information provided by connection times, but it was not misled by transient fluctuations in the wireless landscape. AvMet was also able to increase the availability of Internet services: it provided up to half an order of magnitude improvement over existing predictors.

Our experiences with MultiNAT and AvMet have allowed us to make three key conclusions:

1. **Exploiting multi-homing does not require elaborate client-side configuration.** As shown in Chapter 3, our MultiNAT service is able to provide clients with the performance and availability benefits of multi-homing, but with no configuration changes for most clients.

2. **Existing prediction schemes are not effective in the face of unpredictable wireless losses.** As shown in Chapter 4, the unpredictability of wireless loss renders the MONET prediction scheme ineffective in wireless networks.

3. **Past history is a reliable indicator of future performance.** As shown in Section 4.4, our AvMet link-selection scheme is able to glean useful information from connection times in spite of unpredictable wireless loss. By looking at historical connection time trends, AvMet is able to filter out transient fluctuations in wireless link quality; AvMet can provide performance and availability improvements in *all* network configurations.

Bibliography

[1] Akamai: SureRoute for Failover. `http://www.akamai.com/en/html/services/sureroute_for_failover.html/`, June 2005.

[2] MIT Roofnet. `http://pdos.csail.mit.edu/roofnet/`, June 2005.

[3] Netfilter/Iptables Project Homepage. `http://www.netfilter.org`, June 2005.

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

[5] David G. Andersen. *Improving End-to-End Availability Using Overlay Networks.* PhD thesis, Massachusetts Institute of Technology, February 2005.

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

[7] David G. Andersen, Hari Balakrishnan, M. Frans Kaashoek, and Rohit Rao. Improving Web availability for clients with MONET. In *Proc. 2nd Symposium on Networked Systems Design and Implementation (NSDI)*, Boston, MA, May 2005.

[8] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R.H. Katz. A Comparison of Mechanisms for Improving TCP Performance over Wireless Links. *IEEE/ACM Transactions on Networking*, 5(6), December 1997.

[9] H. Balakrishnan, S. Seshan, E. Amir, and R.H. Katz. Improving TCP/IP Performance over Wireless Networks. In *Proc. 1st ACM Conf. on Mobile Computing and Networking*, November 1995.

[10] John Bicket, Daniel Aguayo, Sanjit Biswas, and Robert Morris. Architecture and evaluation of an 802.11b mesh network. In *Proceedings of the 11th ACM International Conference on Mobile Computing and Networking (MobiCom '05)*, Cologne, Germany, August 2005.

[11] Sanjit Biswas. Personal communication, June 2005. ETT metric failures.

[12] R. Caceres and L. Iftode. Improving the Performance of Reliable Transport Protocols in Mobile Computing Environments. *IEEE Journal on Selected Areas in Communications (J-SAC)*, 13(5), June 1995.

[13] Benjamin A. Chambers. The grid roofnet: a rooftop ad hoc wireless network. Master's thesis, MIT, May 2002.

[14] B. Chandra, M. Dahlin, L. Gao, and A. Nayate. End-to-end WAN Service Availability. In *Proc. 3nd USENIX Symposium on Internet Technologies and Systems (USITS)*, pages 97–108, San Francisco, CA, March 2001.

[15] Douglas S. J. De Couto, Daniel Aguayo, John Bicket, and Robert Morris. A high-throughput path metric for multi-hop wireless routing. In *Proceedings of the 9th ACM International Conference on Mobile Computing and Networking (MobiCom '03)*, San Diego, California, September 2003.

[16] A. DeSimone, M. C. Chuah, and O. C. Yue. Throughput Performance of Transport-Layer Protocols over Wireless LANs. In *Proc. IEEE Conference on Global Communications (GlobeCom)*, Houston, TX, December 1993.

[17] R. Droms. *Dynamic Host Configuration Protocol.* Internet Engineering Task Force, March 1997. RFC 2131.

[18] Krishna P. Gummadi, Harsha V. Madhyastha, Steven D. Gribble, Henry M. Levy, and David Wetherall. Improving the reliability of Internet paths with one-hop source routing. In *Proc. 6th USENIX OSDI*, San Francisco, CA, December 2004.

[19] M. Holdrege and P. Srisuresh. *Protocol Complications with the IP Network Address Translator.* Internet Engineering Task Force, January 2001. RFC 3027.

[20] Internet Engineering Task Force. *IP Network Address Translator (NAT) Terminology and Considerations*, August 1999. RFC 2663.

[21] David B. Johnson, David A. Maltz, and Yih-Chun Hu. Dynamic Source Routing protocol for mobile ad hoc networks (DSR). Internet draft (work in progress). `http://www.ietf.org/internet-drafts/draft-ietf-manet-dsr-10.txt`, July 2004.

[22] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, August 2000.

[23] D. Richard Kuhn. Sources of failure in the public switched telephone network. *IEEE Computer*, 30(4), 1997.

[24] C. Labovitz, A. Ahuja, and F. Jahanian. Experimental Study of Internet Stability and Wide-Area Backbone Failures. In *Proc. 29th International Symposium on Fault-Tolerant Computing*, June 1999.

[25] Craig Labovitz, Abha Ahuja, and F. Jahanian. Experimental study of Internet stability and wide-area network failures. In *Proc. FTCS*, Madison, WI, June 1999.

[26] Mike Muuss. TTCP. `http://ftp.arl.army.mil/ mike/ttcp.html`, July 2005.

[27] V. Paxson. End-to-End Routing Behavior in the Internet. In *Proc. ACM SIGCOMM*, pages 25–38, Stanford, CA, August 1996.

[28] V. Paxson. End-to-End Routing Behavior in the Internet. *IEEE/ACM Transactions on Networking*, 5(5):601–615, 1997.

[29] C. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. *ACM Computer Communications Review*, October 1994.

[30] J. B. Postel. *Internet Protocol.* Internet Engineering Task Force, Information Sciences Institute, Marina del Rey, CA, September 1981. RFC 791.

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

[32] Stefan Savage, Tom Anderson, et al. Detour: A Case for Informed Internet Routing and Transport. *"IEEE Micro"*, 19(1):50–59, January 1999.

[33] Niki Scevak, Joe Laszlo, and Andrew Peach. US Online Population Forecast, 2005 to 2010. `http://www.jupiterresearch.com/bin/item.pl/research:vision/75/id=96341`, May 2005.

[34] W. R. Stevens. *TCP/IP Illustrated, Volume 1.* Addison-Wesley, Reading, MA, 1994.

[35] Alexander Yip. NATRON: Overlay routing to oblivious destinations. Master's thesis, MIT, August 2002.