DYNAMIC ROUTING USING AN OVERLAY NETWORK OF RELAYS

A thesis presented to

the faculty of

the Russ College of Engineering and Technology of Ohio University

In partial fulfillment

of the requirements for the degree

Master of Science

Philip M. Prudich

August 2005
This thesis entitled

DYNAMIC ROUTING USING AN OVERLAY NETWORK OF RELAYS

by

Philip M. Prudich

has been approved for

the School of Electrical Engineering and Computer Science

and the Russ College of Engineering and Technology by

Shawn D. Ostermann

Associate Professor of Electrical Engineering and Computer Science

Dennis Irwin

Dean, Russ College of Engineering and Technology
The goal of resource management in a distributed system is to appropriate the available resources in such a way as to meet the requirements of the individual applications and system as a whole. In addition to commonly managed resources such as CPU cycles or memory usage, network resources are a critical part of this goal. Current attempts at network resource management do not provide direct control of the route to the resource manager. The solution that we propose is an overlay network of relays. Traffic may be redirected among these relays in order to provide a network resource manager the means to control routing. Two types of relays are presented. The first operates in kernel-space, providing a more efficient relay but requiring higher system privileges. The second operates in user-space, allowing for deployment on systems without administrative access. This user-space relay, however, has a greater impact on performance. We discuss the design, implementation, and performance analysis of the system, as well as the costs associated with using it.

Approved:

Shawn D. Ostermann

Associate Professor of Electrical Engineering and Computer Science
Acknowledgments

I would first like to thank Dr. Shawn Ostermann for the many years of advising that he has provided. His lessons regarding the general topic of computer science have been thought provoking and his excitement for the field contagious.

I would also like to thank Dr. Carl Bruggeman for his invaluable guidance during the course of my studies. The work presented here stems from discussions with him and his role as a co-advisor has been greatly appreciated.

Finally, I would like to thank all the members of the Internetworking Research Group at Ohio University for their encouragement and support, especially Josh Blanton, who has proven himself a wealth of information and a helpful friend.
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>3</td>
</tr>
<tr>
<td>List of Tables</td>
<td>8</td>
</tr>
<tr>
<td>List of Figures</td>
<td>9</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>11</td>
</tr>
<tr>
<td>2 Background</td>
<td>16</td>
</tr>
<tr>
<td>2.1 Networking Background</td>
<td>16</td>
</tr>
<tr>
<td>2.1.1 Internet Protocol</td>
<td>16</td>
</tr>
<tr>
<td>2.1.2 User Datagram Protocol</td>
<td>17</td>
</tr>
<tr>
<td>2.1.3 Transmission Control Protocol</td>
<td>17</td>
</tr>
<tr>
<td>2.1.4 Network Address Translation</td>
<td>18</td>
</tr>
<tr>
<td>2.1.5 Quality of Service</td>
<td>19</td>
</tr>
<tr>
<td>2.1.6 Overlay Networks</td>
<td>19</td>
</tr>
<tr>
<td>2.1.7 Grid Computing and Resource Management</td>
<td>19</td>
</tr>
<tr>
<td>2.2 Related Work and Technology</td>
<td>22</td>
</tr>
<tr>
<td>2.2.1 IP Source Routing</td>
<td>22</td>
</tr>
<tr>
<td>2.2.2 Multiprotocol Label Switching</td>
<td>23</td>
</tr>
<tr>
<td>2.2.3 IP Traffic Engineering</td>
<td>24</td>
</tr>
<tr>
<td>2.2.4 Intserv and Diffserv</td>
<td>24</td>
</tr>
<tr>
<td>2.2.5 Onion Routing</td>
<td>25</td>
</tr>
<tr>
<td>2.2.6 Detour</td>
<td>25</td>
</tr>
<tr>
<td>2.2.7 Resilient Overlay Networks</td>
<td>26</td>
</tr>
<tr>
<td>3 Dynamic Routing</td>
<td>27</td>
</tr>
</tbody>
</table>
### 3.1 Our Approach
- **3.1.1 Detailed Explanation of Requirements**

### 3.2 General Architecture
- **3.2.1 Component Descriptions**
- **3.2.2 Walk-Through of Usage**
- **3.2.3 Additional Design Decisions**

### 3.3 End Hosts and Applications

### 3.4 User-Space Solution
- **3.4.1 Why Tunnel?**
- **3.4.2 Tunnel Agent Operation**
- **3.4.3 Tunnel Implementation**
- **3.4.4 User-Space Relay Agent Operation**
- **3.4.5 User-Space Relay Implementation**

### 3.5 Kernel-Space Solution
- **3.5.1 Kernel-Space Relay Implementation**

### 3.6 Command Module
- **3.6.1 Route Maintenance with User-Space Relays**
- **3.6.2 Route Maintenance with Kernel-space Relays**
- **3.6.3 Command Module Design Decisions**

### 3.7 Analysis
- **3.7.1 Design**
- **3.7.2 Implementation**
- **3.7.3 Tunnels and the Internet Control Message Protocol (ICMP)**
- **3.7.4 Some Notes on Security Implications**

### 4 Experimental Setup and Results
- **4.1 Testbed Setup**
- **4.2 Benchmarks and Performance Analysis**
- **4.2.1 Throughput**
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2.2</td>
<td>CPU Load</td>
<td>56</td>
</tr>
<tr>
<td>4.2.3</td>
<td>Analysis</td>
<td>57</td>
</tr>
<tr>
<td>4.3</td>
<td>Per-relay Penalties</td>
<td>62</td>
</tr>
<tr>
<td>4.3.1</td>
<td>User-space Relay Penalties</td>
<td>63</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Kernel-space Relay Penalties</td>
<td>64</td>
</tr>
<tr>
<td>4.4</td>
<td>Delay Effects</td>
<td>64</td>
</tr>
<tr>
<td>4.5</td>
<td>Effects of Dynamic Routing</td>
<td>65</td>
</tr>
<tr>
<td>4.5.1</td>
<td>A Second Testbed</td>
<td>66</td>
</tr>
<tr>
<td>4.5.2</td>
<td>Changing the Dynamic Path</td>
<td>67</td>
</tr>
<tr>
<td>4.5.3</td>
<td>Round Trip Time</td>
<td>69</td>
</tr>
<tr>
<td>5</td>
<td>Conclusions and Future Work</td>
<td>75</td>
</tr>
<tr>
<td>5.1</td>
<td>Conclusions</td>
<td>75</td>
</tr>
<tr>
<td>5.2</td>
<td>Future Work</td>
<td>76</td>
</tr>
<tr>
<td>Bibliography</td>
<td></td>
<td>78</td>
</tr>
<tr>
<td>Appendix: Implementation Figures</td>
<td></td>
<td>82</td>
</tr>
</tbody>
</table>
List of Tables

Table | Page
--- | ---
4.1 Maximum and Average Throughputs for the Studied Routing Methods | 55
4.2 Encapsulation and Fragmentation Overhead | 60
4.3 Comparison of CPU Usage Between Fragmenting and Non-Fragmenting Cases | 61
4.4 Co-located User-space Relays and Performance | 62
4.5 Effect of the Number of User-space Relays and Host CPU Load on Throughput | 63
4.6 Effect of the Number of Kernel-space Relays and Host CPU Load on Throughput | 64
4.7 Effect of Round Trip Time on Performance | 65
4.8 Round Trip Times for Varying Relayed Paths | 70
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 General Usage Example</td>
<td>34</td>
</tr>
<tr>
<td>3.2 Tunnel Operation</td>
<td>40</td>
</tr>
<tr>
<td>3.3 User-space Relay Operation</td>
<td>43</td>
</tr>
<tr>
<td>3.4 Kernel-space Relay Operation</td>
<td>46</td>
</tr>
<tr>
<td>4.1 General Testbed</td>
<td>55</td>
</tr>
<tr>
<td>4.2 Packet Encapsulation</td>
<td>58</td>
</tr>
<tr>
<td>4.3 Packet Fragmentation</td>
<td>58</td>
</tr>
<tr>
<td>4.4 Testbed Allowing for Change in Path</td>
<td>66</td>
</tr>
<tr>
<td>4.5 RTT Change on Interception</td>
<td>68</td>
</tr>
<tr>
<td>4.6 Throughput Change when ‘Freeing’ a Connection</td>
<td>69</td>
</tr>
<tr>
<td>4.7 Throughput Response to a New Path with Longer RTT</td>
<td>71</td>
</tr>
<tr>
<td>4.8 Throughput Response to a New Path with Shorter RTT</td>
<td>72</td>
</tr>
<tr>
<td>4.9 Time-Sequence Graph of a Switch to a Shorter Path</td>
<td>74</td>
</tr>
<tr>
<td>A.1 User-space Relay and Tunnel Flowchart</td>
<td>83</td>
</tr>
</tbody>
</table>
A.2 User-space Relay and Tunnel Flowchart: Process Command Branch . . . 84
A.3 Kernel-space Relay Flowchart . . . . . . . . . . . . . . . . . . . . . . 85
A.4 Kernel-space Relay Flowchart: Add Route Command . . . . . . . . . 86
A.5 Kernel-space Relay Flowchart: Delete Route Command . . . . . . . . 87
A.6 Command Module Flowchart: Add Route Command . . . . . . . . . . . 88
A.7 Command Module Flowchart: Delete Route Command . . . . . . . . . 89
A.8 Command Module Kernel-space Relay Route Setup . . . . . . . . . . . 90
1. Introduction

In distributed systems, the distribution of workload and system usage is monitored through a Resource Manager (RM). Many times, the resources most closely watched are those that are integral parts of a single node [34]. For example, an RM may attempt to distribute workload in order to spread out the operational costs in terms of processor load, memory usage, and hard drive capacity. However, these are not the only attributes that one might wish to manage. In many distributed systems, it may be critical to manage the network resources as well as the host resources. Network resources include not only bandwidth, but also other Quality of Service (QoS) related measurements such as jitter, latency, and reliability of the link. Network resource management is necessary because some applications have requirements for communications other than simple throughput or connectivity. Because network conditions are constantly in a state of flux, communications must be adaptable so that network resources can be managed and performance requirements can be met. If a currently in-use path is determined to be undesirable for any reason (e.g. overload, disconnection, or other low QoS metric), it would be beneficial to be able to bypass the problem area.

Grid computing is a type of distributed system that emphasizes the possibility of geographic distance between cooperating hosts. In these systems, the network is also a resource, and may need to be managed. Network requirements and rules may also affect grid communication. Administrators may prefer that grid communication be routed along slower or more expensive links (in the routing sense), so that the best performance can be left for ordinary users. Dynamic routing would allow this to hap-
pen by shifting the traffic’s path away from default routes. Additionally, because grids are global by nature, they will most likely span multiple administrative domains. A system that works across those domains without requiring any additional cooperation would be most easily deployed. Network-wide traffic management approaches may not be feasible due to differences in network administration policies. Finally, in a grid situation, specialized grid software is already present to enable communication. This software may provide locations through which rerouting may occur without installing extra software on every router in the network.

There are a number of other situations in which a dynamic rerouting scheme would prove useful. One is in a traffic engineering scenario. For example, load balancing could be achieved by dynamically rerouting traffic over a set of links. Connection traffic could be routed away from overloaded portions of the network in order to alleviate strain.

Application- and connection-specific communication requirements could be met readily. For example, streaming video applications may tolerate a slower rate of end-to-end throughput if, in turn, the communication travels across links which demonstrate low jitter. In this manner, the periodic video updates may be more likely to be on time and thus provide a smoother frame rate. Interactive traffic could be placed onto links with lower delay while non-interactive traffic could be routed elsewhere. In a small enough system, traffic may be able to be routed through trusted routers, as opposed to default routes which may traverse more open links. In this manner, an additional layer of security may be provided.

In many distributed systems, a common solution to network management is the co-allocation of processes and bandwidth [5]. Not only must a host with adequate CPU or memory availability be found, but there must also be enough unreserved bandwidth on the network before the task is allocated a space. Another solution could be to migrate applications from host to host should a network problem arise. If there is not enough capacity available at the current host, the application would
be moved to a host with an adequate supply. This may work in some cases, but not all. Simply moving applications does not necessarily guarantee a substantial change in route.

It is not possible to migrate all applications. Not all hosts are capable of running all applications. There could be requirements on operating system, memory, or CPU availability which prevent the application from residing on other hosts. Physical location may be another problem. Any system involving sensors or actuators has software resident with those agents which cannot be moved. It is not wise, for example, to move a mine shaft air quality sensor out of the mine shaft in order to guarantee bandwidth availability to any air quality update messages. In this case it may be possible, however, to move enough other traffic off links used by the sensor communication so that sufficient bandwidth could be guaranteed.

If the goal is to accommodate communication requirements of applications in a distributed system, process migration can sometimes be an incomplete or inefficient solution to traffic management. It does not always provide the degree of control that may be required for a complex system. Migration may not be possible in all cases. For all of these reasons, another solution to traffic management must be found.

The straightforward solution is to move the traffic itself, not the applications. Applications can be resident at whichever host can best meet their requirements. The traffic can then be routed in such a way as to meet individual application, as well as system, constraints. When applications are added or removed from the system, and as network conditions change over type, routes can be changed and traffic patterns can be adjusted in order to continue meeting system requirements. Dynamic traffic rerouting would provide another tool for finding an optimal configuration in a distributed system.

Our main goal is to create a system for routing traffic dynamically in order to aid an RM in the management of network resources. To this end, we will de-couple the final destination of a packet from the routing decisions made for it, so that constraint-
based routing may be used. We must also allow for traffic redirection to occur at any point during a connection’s lifespan, with minimal impact on the communicating applications. It should be possible to direct separate connections between a pair of hosts along different paths. In addition, we will require minimal modifications to the network as a whole, and will not alter current protocols nor require installation of software onto standard routers. Also, the system should maintain fairness in the bandwidth-sharing sense at least as well as the original connection would have. The solution we propose will be transparent to the end applications, and will be able to be transparent to the end hosts, if required.

The system should be such that it can be used across the Internet without modification, so that several “islands” of a managed network could be managed as one. Finally, requirements on the extent of administrative control over the network should be kept to a minimum, to facilitate distributed systems physically located among multiple administrative domains.

The solution proposed here involves an overlay network of relays. This is an investigation into two different approaches to this system. Through working implementations, the two approaches are compared, contrasted, and quantified in their costs and benefits. The first approach uses user-space relay agents to provide dynamic rerouting capabilities. The second approach replaces the user-space relays with kernel-level relays, utilizing already-existing Network Address Translation (NAT) procedures. Finally, recommendations will be made regarding the proposed solutions, the costs associated with each, and the benefits achieved.

The implementation of the system was successful in providing the desired redirecting abilities. The kernel-space relay was found to have performance characteristics comparable to a default IP route when following the same path. The user-space relay, however, showed performance degradation in terms of attainable throughput, due in a large part to the overhead caused by the encapsulation and fragmentation necessary for it to work. The result is an average 5.5% reduction in throughput. In addition,
the CPU load introduced by the user-space relays can be significant, ranging as high as 38%\(^1\).

The layout of this thesis is as follows: Chapter 2 details the networking background necessary for understanding this system, as well as related work and approaches to similar problems. Chapter 3 describes the design and implementation of the dynamic routing solution which we propose. The results of benchmarking, performance analysis, and experimental use of the system are presented in Chapter 4. Conclusions and recommendations for future work are discussed in Chapter 5.

\(^{1}\)The hosts used in experimentation operate with AMD Athlon\textsuperscript{TM} processors running at 1.53 GHz, with 512 Mb of RAM.
2. Background

In order to understand the overlay system proposed, as well as the costs associated with using it, some background knowledge in networking is required. Section 2.1 begins with some basic networking background information, and ends with an introduction to distributed and grid computing, which are motivating cases for the work presented here. Section 2.2 discusses technologies used with similar problems, as well as the reasons why they are not appropriate solutions to the problem at hand.

2.1 Networking Background

2.1.1 Internet Protocol

The Internet Protocol (IP) [32] is a system designed to deliver datagrams between hosts on a network. IP does not provide any notion of a connection between hosts. IP is important for our purposes because it is the underlying protocol being used by both the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP).

One of IP’s most important functions is its role in routing. The IP header includes a number of different fields in order to aid in routing effectively. Included are the source and destination IP addresses, as well as precedence, or Type of Service (ToS), bits. In addition, there are optional fields that may be present. In particular, there is a Source Routing option that may affect the routing of the packet; this will be discussed further in Section 2.2.1.

IP routing is usually performed using routing tables, also called ‘next hop routing.’ [13] In this method, a router will examine a packet, and use the destination address as a lookup into a table. The table will list the next hop by destination.
Routing tables can be populated in a number of different ways [7]. Some examples are distance-vector, link-state, and shortest path first protocols. All of these types of protocols try to optimize the route a packet takes by using the same metric for all packets.

Constraint-based routing is the practice of making routing decisions based upon metrics other than distance (or number of hops) to the destination. While this can be one of the metrics, other factors influencing a routing decision may be Quality of Service requirements (e.g. available capacity, low delay, etc), or network policies (e.g. rate limiting and load balancing). The Type of Service field in the IP header attempts to provide a method for packets to request Quality of Service levels in terms of delay, throughput, reliability, and cost.

2.1.2 User Datagram Protocol

The User Datagram Protocol (UDP) [30] builds slightly upon IP. An important addition is that of port numbers. A port number is a way to identify the process on a remote host to which you are sending. The source and destination port numbers in the UDP header, along with the source and destination IP addresses, can identify a connection. UDP connections are simply a question followed by a response. UDP makes no guarantees about delivery; this includes whether or not the datagrams make it to the destination, as well as the order in which they arrive. In addition, UDP provides no method of congestion control. A UDP sender will not pace itself in an attempt to limit network congestion. These features allow UDP to have minimal overhead. However, this unreliable service may not be sufficient for some applications.

2.1.3 Transmission Control Protocol

The Transmission Control Protocol (TCP) [13, 33] also operates at a layer above IP. TCP, however, provides a reliable, connection-oriented packet delivery system. In addition to port numbers, TCP also provides sequence numbers, which allow for in-order delivery as well as prevent duplications at the destination host. These se-
quence numbers are also critical for use in providing reliability to the communication. Reliability is achieved through the acknowledgment of data received by the destination. Also, modern TCP self-regulates its sending speed via congestion control and feedback mechanisms. Instead of quickly introducing a maximum amount of traffic to a network, a sender will incrementally increase its sending rate until losses are detected. At that point, the sender will back off, preventing the high levels of traffic which could lead to congestion collapse [13].

The larger set of features creates a higher overhead for TCP. Depending upon the application, this overhead may be unnecessary.

2.1.4 Network Address Translation

Network Address Translation (NAT) [13, 16] was first introduced as a short-term solution to the problems of a diminishing IP address space. With NAT, a network can use private, re-usable addresses as long as all traffic is routed through a NAT box. The NAT box will translate the private addresses into a public, global address. All returning traffic comes back through the same NAT box, and the address is translated back to the original value. In this manner, many private addresses can share one public, globally routeable address. While this solution is useful in many situations, there are a number of shortcomings associated with it.

Some applications pass IP addresses and port numbers between hosts. If this is the case, these applications may fail once they get routed through a NAT box. This is because the NAT box can change the outer IP address, but cannot be expected to understand every application-level protocol. The results will be a packet which has the source address $s1$, but inside, the applications may believe that the source address is supposed to be $s2$. This inconsistency may cause problems for some applications. In addition, NAT may cause problems with security protocols, specifically IPsec [1]. IPsec is designed to authenticate and/or encrypt the payload of an IP datagram [23, 24]. In the case of encryption, host port numbers are not available for the NAT box.
to use. In the case of authentication, changes made via NAT to the IP address fields will be indistinguishable from those made via a hostile attack.

2.1.5 Quality of Service

Quality of Service (QoS) [14] refers to the level of performance a certain application or connection is seeing. With regard to networking, QoS concerns could be link speed, data loss, queuing delay, or jitter (delay variance).

Applications could request which type of service they would like to receive. For example, a multimedia application might tolerate a lower speed on the wire, as long as the jitter seen is relatively low. Another metric which could be considered is that of security. It may be important to an application that it only communicates with others via a trusted line. We aim to provide a system which could be used to satisfy widely varying QoS requirements.

2.1.6 Overlay Networks

An overlay network is a set of hosts or routers spread across some section of the Internet. Each node acts as a router, relay, or agent in the overlay system, and can have specialized software as needed. The paths between nodes can be abstracted as a simple link. One of the benefits associated with overlay networks is that they provide a means for rapid deployment of new technology. For example, experimental network services can be encapsulated in IP headers and sent between overlay nodes which understand those services. There is no requirement, then, that all Internet routers have the new service enabled.

2.1.7 Grid Computing and Resource Management

A major research area for which dynamic routing is useful is resource management in distributed systems [19, 35]. In these systems, a Resource Manager (RM) will attempt to distribute work in such a way that the system is most efficiently utilized while meeting individual application and system requirements. For example, applications may be run on several machines in order to share the CPU load, so that no
single machine is overloaded. In this way, certain requirements an application might have on the amount of available resources may be met. Resource management can apply to many different resources. Two obvious examples are processor time and available memory.

Other resources which are helpful to have managed are network resources. The RM would like to be able to control how much traffic is on certain links. Reasons for this might be specific latency or throughput demands of applications.

Research has already been performed in the area of migrating applications from one host to the next [11, 26, 38, 39]. Migrating the applications may alleviate problems with network availability, in that different links may be used from a different host. However, it may not always be possible to move the application. For example, any system using sensors and actuators will need to have software located with those sensors and actuators. It would not be possible to simply move, say, the air quality sensor off of the host in the mine shaft, since that’s precisely the air we care about monitoring. In addition, it may be the case that there are a limited number of hosts which meet other requirements of the application (i.e. a host may need some amount of available bandwidth as well as some amount of memory guaranteed, or there may even be hardware license restrictions). If there is only one host with the necessary properties, moving the application is not an option.

For all of these situations, the Resource Manager may not be able to move the application itself, but there may be a solution present if the RM could move the traffic. Perhaps a route other than the default would yield results acceptable for the system. Perhaps routing other, less important traffic away from a bottleneck link would allow the requirements to be met. For these situations, standard IP routing may not be sufficient.

Grid Computing [18, 20] is another area for which dynamic routing policies are needed. “Grid Computing” refers to a distributed, heterogeneous system of computers
whose goal is to share resources. These resources may be computation resources, data storage resources, software availability, or network resources.

Grids may be set up to utilize idle CPU power, to provide an arena for parallel algorithms, to collaborate amongst organizations, to provide reliability, or to balance resource usage throughout an organization. Grids range in size and makeup from a small lab full of identical hosts to a global grid of varying types of machines, connected through different means.

There are many problems which must be overcome to create a good grid architecture. The communications system can be complex, as the distribution of the hosts could be global. The grid software should be able to support prioritization of communication, as well as reservation of available bandwidth. The QoS issues mentioned previously all apply to a grid architecture.

Dynamic routing of grid communications would be useful in a number of ways. First, network connectivity is a resource itself. A grid-enabled application may need a route that has certain properties, and the default route may not be sufficient. Through dynamic routing, an acceptable route may be found, utilizing links that other grid machines are connected to. In addition, inter-grid communication must be managed. For example, if hosts A and B are working on a common problem, there may be a high volume of traffic between them. Being able to route around that volume of traffic (or being able to route that traffic elsewhere) may be crucial in maximizing the performance of the grid. Batch jobs are very often good candidates for a grid system. This is because a job may be given to a remote host to be completed and returned at a later time. If the inter-grid communication requirements for a job are low (as many batch jobs may be), the traffic generated by them may respond better to re-direction than, for example, a more interactive application would.

Finally, a grid computing system would be an ideal candidate for the solution presented here due to the distribution of the system, as well as the need for communications management throughout the grid. Grid systems must be well-thought-out
implementations, which lead to a thorough knowledge of the communication properties amongst the individual hosts. In addition, specialized grid management software is already present in many places, which may provide relatively easy locations to install the solution presented here. Finally, the distributed aspect of grid computing often requires the grid to span varying administrative domains, so a network-wide traffic management approach may not be feasible. All of these attributes make a grid computing system an ideal scenario for the system we present.

2.2 Related Work and Technology

Some pre-existing technologies provide partial solutions to this problem. In certain circumstances, these technologies might prove to be adequate. However, they do not provide complete solutions, and for that reason a new system is required.

2.2.1 IP Source Routing

IP Source Routing [32, 12] is an option that allows the sender of a packet to dictate the path used for a connection. Using the appropriate IP option, the sender creates an ordered list of network routers (hops) and inserts this list into the packet header. This list of hops will be followed, and the last hop will be the destination host. The return path is simply the original path in reverse order. In “loose” source routing, the list of hops is just a starting point; other routers may be used, but those in the list must still be visited in that order. In “strict” source routing, only those routers in the hop list are permitted to be visited. If the next router in a list is unreachable, nothing can be done by an intermediate router to salvage the connection.

IP Source Routing has a number of down-sides. The first is that the return path is determined simply by reversing the forward path. This limits control over the routes, should an asymmetrical path be needed. The next is a limit on the number of intermediate hops. Due to IP option size limitations, there can be no more than nine hops specified in the route data portion of the source route option. Depending on network setup, this may not be sufficient.
Source routing would also have to be permitted along the entire path. Source routing has some security implications [6] that have caused it to be largely disabled across much of the Internet. An evildoer could create a packet that lists his own machine as one of the series of intermediate hops, and spoofs a trusted host as the source. The destination host may decide to accept the connection based upon the trusted source host. This is a security problem because the return path will be the forward path reversed, allowing the true (evil) source to intercept the return traffic.

### 2.2.2 Multiprotocol Label Switching

Multiprotocol Label Switching (MPLS) [36] is a technology that allows routing to be performed in a number of ways. The original motivation behind MPLS was to allow this forwarding to be done by routers which could not otherwise perform well (or quickly) enough. Routing lookups are done via a label inserted into the packet instead of the traditional connection identifying information. Label assignments can be made based on any information the routers have about a packet, not merely where it’s going. This allows for constraint-based routing. MPLS has also been shown to be useful in a number of other ways. Traffic engineering [4] and Quality of Service are both possible through MPLS. MPLS can also provide fault tolerance in the form of backup paths.

Upon entry into an MPLS network, a packet is assigned to a Forwarding Equivalence Class (FEC). Routers in the MPLS network (called Label Switching Routers, or LSPs) then come to agreement on a set of labels for this FEC. This can be done on a hop-by-hop basis, or the labels can be determined upon the initial entry of the packet. In the latter case, an explicit route can be set up for the packet to follow. The labels are distributed via a Label Distribution Protocol (LDP). The ingress router then places a label with the packet, and sends it to the next hop LSR. The next LSR will use the label as a lookup into its table of next-hops. Once a packet is assigned to an FEC, LSRs downstream will not need to investigate the header, but need only to look at the label to determine any forwarding action.
MPLS also allows “stacking” of labels, to allow tunneling to arbitrary levels. Furthermore, MPLS is scalable enough to be used in major service provider networks. However, it should be noted that in many cases the MPLS labels are implemented as encapsulation headers on the original packets. More importantly, in order to join two MPLS “islands”, another layer of encapsulation is needed to cross the public Internet [40]. This is because MPLS requires administrative control over the routers in the network. This is precisely one requirement which this project hopes to overcome. Installing and enabling MPLS or any other router-based solution may not be an option under all circumstances.

2.2.3 IP Traffic Engineering

IP traffic engineering using the Resource ReSerVation Protocol (RSVP) [10] also attempts to provide QoS guarantees. This approach shares many of the shortcomings of MPLS, as RSVP is used in MPLS to help set up label switched paths. While QoS guarantees may be sufficient for many problems, RSVP does not provide control over the route itself. This could be a problem when redirection is desired for reasons which may not be negotiated along a path (e.g. perceived security of a link), or when redirection is determined necessary due to special needs not widely implemented.

2.2.4 Intserv and Diffserv

Integrated Services, or Intserv [9], also uses RSVP in order to reserve resources along a communication path. Intserv aims to provide guaranteed service as well as predicted service through resource reservation and admission controls. Routers along the path keep track of link and flow statistics in order to regulate traffic. Applications, upon beginning communication, send out RSVP messages in order to reserve the path they need. Intserv’s major drawback, however, is the amount of state kept for each connection. The state required in the routers is prohibitive to the wide-scale deployment of Intserv.
Differentiated Services, or Diffserv [8], provides network-wide QoS routing through the use of the Type of Service field in the IP header. The granularity of QoS routing is coarser in Diffserv than in Intserv, allowing it to scale much more effectively to Internet-wide use. Traffic is labeled in the ToS field with the type of service that it requires. Routing is then performed based, in part, on the ToS field. In this manner, no state is required for an individual flow.

Both Intserv and Diffserv, while addressing the problem of QoS routing, still fail to provide a mechanism for route control. While it is conceivable that these technologies could have a role in between the relay agents of our system, they are not sufficient as independent entities.

2.2.5 Onion Routing

Onion Routing [15] is a project whose goal is to provide anonymity to Internet users. A distributed overlay network of “Onion Routers” is created. These Onion Routers are user-level processes. A user who wishes to use the Internet anonymously accesses this network through an “Onion Proxy” on his own machine. Through this proxy, a sequence of hops is chosen and a series of encrypted tunnels created in a telescoping approach. Communication is performed through fixed-size cells. At the proxy, data is wrapped in layers of encryption, one layer per Onion Router to be visited. At each Onion Router, a layer is stripped off, and the next hop is discovered. In this manner, each Onion Router knows only the previous and next hop of each packet. This approach uses many iterations of encryption and decryption, as well as the trip to user-space at each Onion Router. In addition, some endpoint software is required in the form of the Onion Proxy.

2.2.6 Detour

The goal of the Detour project [37] is to investigate alternate-path routing mechanisms. Detour routers are deployed throughout an area of the network and traffic between them is tunneled over the Internet. In this way, it is similar to the solution
we propose. The Detour routers, however, will make routing decisions based upon QoS guidelines. The routers exchange not only connectivity information, but also link data such as actual drop rates and latency.

The Detour project focuses mostly on the routing algorithms needed to take advantage of this network information. Some algorithms try to route around currently overloaded portions of the network, while others try to balance a single flow’s traffic over multiple paths.

While these routing algorithms are indeed interesting, they are beyond the scope of this project. The Detour project, while providing the information needed to make knowledgeable decisions about routing, does not provide an interface for route control itself, as would be needed by a resource managing application.

2.2.7 Resilient Overlay Networks

A Resilient Overlay Network (RON) [3] is an overlay system whose goal is to detect network problems and route around them. The system probes the network paths between its’ nodes to keep track of their current paths. Upon receiving a packet, the RON will determine if the default route is currently operational (or meets any performance constraints provided for the connection). If the default path is not functioning or is not acceptable, the RON intercepts the traffic and forwards it to another RON host, taking an alternate path if one is available.
3. Dynamic Routing

In this chapter, the dynamic routing system is described, in design and in implementation. After a brief introduction to our approach, the general architecture, usage, and design decisions are described in Section 3.2. The user-space based solution (Section 3.4) and the kernel-space based solution (Section 3.5) are then discussed, followed by a description of the Command Module (Section 3.6). Finally, an analysis of the design and implementation follows in Section 3.7.

3.1 Our Approach

Our main goal is to create a system for routing traffic dynamically in order to aid a Resource Manager in the management of network resources. To this end, we will de-couple the final destination of a packet with the routing decisions made for it, so that constraint-based routing may be used. We must also allow for traffic redirection to occur at any point during a connection’s lifespan, with minimal impact on the communicating applications. It should be possible to direct separate connections between a pair of hosts along different paths. In addition, we will require minimal modifications to the network as a whole, and will not alter current protocols nor require installation of software onto standard routers. Also, the system should maintain fairness in the bandwidth-sharing sense at least as well as the original connection would have. The solution we propose will be transparent to the end applications, and will be able to be transparent to the end hosts, if required.

The system should be such that it can be used across the Internet without modification, so that several “islands” of a managed network could be controlled as one. Finally, requirements on the extent of administrative control over the network should
be kept to a minimum, to facilitate distributed systems physically located among
multiple administrative domains.

3.1.1 Detailed Explanation of Requirements

- Routes should be dynamic. That is, given a particular connection, the path
decided upon for that traffic can be changed at any moment. This is crucial for
system-wide traffic engineering, so that the system can adapt as connections are
added, removed, or rerouted. In addition, the characteristics of the connections
themselves will change over time, and a set of connections which previously
shared a link may no longer be able to while still meeting QoS requirements.

- Dynamic routing should be done with connection-level granularity. Destination
based routing methods do not allow for this level of precision. Not all traffic on
a network may be managed, and because of this the system will need to know
exactly which connections to reroute. In addition, \( n \) connections between two
hosts may need to take as many as \( n \) separate routes at any one time. This level
of granularity will allow the resource manager to make constraint-based routing
decisions. Ideally, though, for scalability, a solution will have the capability of
rerouting entire ranges of traffic with a small set of rules.

- The system should be transparent to end hosts and the applications running on
them. One reason for this is ease of deployment. The end applications cannot be
expected to be rewritten in order to support dynamic rerouting capabilities. The
system should be transparent to end hosts so that no installations or updates
will need to be made on the communicating machines. The first reason for this
is that resource-constrained hosts may not be able to run a rerouting process.
In addition, there may be situations in which the server, or the client, reside
outside of the reach of the dynamic rerouting system. As this is a network
management tool, components of it should only be required in the parts of the
network to be managed.
• Only minimal modifications should need to be made to the existing network and routers. Primarily, administrative control cannot be guaranteed in all situations. Also, deployment is simplified by not requiring router modifications.

• As managed traffic may be sharing links with standard traffic, all efforts should be made to maintain fairness between the two. Managed traffic should not adversely affect non-managed traffic (unless permitted by local policies). In terms of fairness, managed traffic should not obtain any larger portion of the bandwidth as a result of the system created. In addition, the congestion control mechanisms in place should not be made ineffective by any solution to this problem.

• The system should be deployable not only on a local network, but also across the global Internet. This implies that the system will have to operate across networks of varying administrative domain. In addition, the system will have to work across networks which may not allow any dynamic rerouting agents whatsoever. Multiple, geographically separate networks should be able to be managed as one.

• Accommodations will have to be made to address the varying levels of administrative access which will be allowed. On some hosts, root access will be possible, while on other hosts, simple user access may be allowed. Both situations must be able to be utilized in this system. It can be a liability to use root access at every agent. In addition, network control is not guaranteed on every network, but the system should be able to cross these areas.

• Finally, it is important that control is provided over the route that the traffic takes. In many circumstances, it may be sufficient to provide QoS guarantees. However, for this problem, it cannot be assumed that all requirements can be met without actual control over the route. Alternatively, the knowledge that
the route will meet any requirements may only be a partial solution, if there is also a requirement that the traffic pass through a specific network or host. For example, a monitoring application in a distributed system may require that all traffic cross one of a set of hosts to assure that all packets are seen.

### 3.2 General Architecture

The general form of the solution presented here is an overlay network. The overlay network is comprised of many relay points, scattered in such a way as to provide many different options for paths amongst them. The method is to relay traffic between these points along a path chosen by an outside entity. The entity choosing paths could be a network resource managing application, or even a human operator. This entity can be referred to as the Network Resource Manager. The intelligence needed by the Network RM to determine acceptable paths is an interesting area of research, but is outside the scope of this project.

Traffic is admitted to the system based upon directives given by the Network RM. The Network RM specifies a connection identifier and a route, which is a series of relay points. A Command Module receives these routes and the connection identifier, and performs the task of alerting any affected relay points of the new routing rules. The Command Module is the interface into the dynamic rerouting system.

In order to operate within a range of administrative privileges, two types of relays have been implemented. The first operates in user-space, and requires a layer of encapsulation. This is done in order to redirect the traffic to a user-level application relay, as well as to preserve original data in the headers. Relaying is performed through changes to the encapsulation header. The second type of relay performs the forwarding operations in kernel-space via Network Address Translation. This type of relay does not require encapsulation. Finally, a “mixed system” is also possible, utilizing both user-space and kernel-space relays. Encapsulation is required in a mixed system in order to accommodate the user-space relays involved.
By intercepting traffic and redirecting it after it leaves the sender, we can provide transparency, allowing immediate use of this system without updating application code. Live updates from the command module to the relays allow traffic redirection to take place mid-connection. The construction of this system allows for traffic engineering without needing to have administrative control over the network itself, only a few gateways and applications.

Encapsulation will affect the maximum performance available to the communicating hosts. However, the goal is to quantify these costs and allow anyone implementing the system to decide if the benefits provided by rerouting traffic outweigh the costs associated with it.

3.2.1 Component Descriptions

There are three major portions to the general solution presented here: the command module, tunnel agents, and relay agents.

Command Module: The interface into the network resource manager or other application making routing decisions for the system. A simple module whose purpose is to turn high-level routing commands into specific commands for each relay and tunnel (if needed), and to send these commands to their respective agents.

Tunnel Agents: Ingress and egress points for the dynamic rerouting system, needed when using user-space relays. The ingress tunnel agent needs to be a gateway capable of intercepting packets as directed by the command module and forwarding them along the chosen path. Upon reaching the end of the dynamic path, the egress tunnel agent will decapsulate the packet and send the original packet using standard routing procedures.

Relay Agents: Agents which may (or may not, in the case of a user-space relay system) be visited between the ingress and egress points of the system. These agents
offer points of redirection in the relay system. There are two methods discussed here for implementing the relay agents, dependent upon administrative access available on the nodes hosting the agents. The first method allows for wider usage, relaying traffic through user-space. The second method relays the traffic via NAT rules in iptables, requiring higher system privileges, as well as more requirements on kernel modules and patches.

It is important to note that the system has been designed without placing requirements on the operation of end hosts or applications. In this manner, transparency to the end hosts can be maintained.

3.2.2 Walk-Through of Usage

In the general case, the system will act as follows: The Command Module will receive an order from the Network RM for a given route to be created. This order will be of the form `<Connection ID | Hop 1 | ... | Hop n>`. Having received the order, the Command Module will check the route for validity, and then will send out route update commands to any affected tunnels and relays. The precise algorithm used for route updates is discussed in Section 3.6. The controller should maintain a database of available agents in order to verify the routes. In addition, the Command Module may keep statistics for each of the agents, such as current load, performance, and usage information. A master routing table will be kept in the Command Module, as the relay agents will only know their specific parts of each route.

A route is considered valid if it contains two agents. Single-agent routes are not considered useful because the agent will need to be located on the default path of the flow in order to intercept it, and will proceed to re-inject it at the same point. For routes visiting at least one user-space relay, there is an additional requirement: the first and last rerouting agents visited must be tunnel agents. Any number of relay agents, both user- and kernel-space, may be in between the ingress and egress tunnel agents. In routes visiting only kernel-space relay agents, there is no requirement of
tunnel usage, although there are certain circumstances which may demand it (e.g. rerouting of IPsec or other NAT-sensitive protocols).

Figure 3.1 demonstrates the usage of the overlay system. The communication being managed flows between hosts H1 and H2. By default, the traffic visits the hosts marked Tunnel A / Relay 1, R1, R2, and Tunnel B / Relay 4 in order from H1 before reaching H2. The hosts labeled Tunnel A/ Relay 1 and Tunnel B/ Relay 2 are gateway hosts on which we have installed our relay and tunnel software. The tunneling hosts will not intercept nor redirect traffic unless instructed to do so by the Command Module, who in turn receives commands from a network resource manager (not pictured). For example, when the Command Module receives a command to redirect the traffic between H1 to H2 via user-space Relay 3, command messages will be sent to Tunnel A, Relay 3, and Tunnel B. The tunnels will start intercepting any traffic between H1 and H2, and will forward it to Relay 3, who will in turn forward it to the opposite tunnel point. Similarly, if the Command Module receives a command to redirect the traffic via kernel-space Relay 2, messages will be sent to Relay 1, Relay 2, and Relay 4(all kernel space relays), and redirecting will begin. In this manner, traffic can be rerouted dynamically with minimal requirements on the network infrastructure.

3.2.3 Additional Design Decisions

3.2.3.1 Use Existing Technology

Many of the design decisions made for this system were based upon a desire to utilize existing technologies to solve the problem. There are many reasons for this desire. First, it simplifies deployment in that minimal updates will be required on any hosts or routers used. In addition, the performance characteristics of existing technology are already known. In the general case we already know whether or not current technology will work on and across today’s Internet. This allows distributed systems whose hosts are scattered further across the Internet to use this solution
without enabling a new protocol over the entire network. Fewer allowances may be needed in some firewalls as there are no new protocol numbers to check for.

3.2.3.2 Including Routes in Packets

The route itself was intentionally omitted from the packets. When compared to IP source routing, this allows an arbitrary-length route. Source routing, as mentioned previously, has a route limited to nine designated intermediate hops due to space constraints on packets. Encapsulating the packets increases their size by a known length, whereas route inclusion would grow the packets by an arbitrary amount. Since much of the overhead associated with the user-space relay is due to encapsulation and fragmentation overhead (see Section 4.2.3), the per-packet overhead should be kept to a minimum.

Finally, routing entries at the relay agents provide another place to provide security. Requiring authentication from the command console when inserting routing rules is another step against using this system as a tool for spoofing packets.
3.2.3.3 Command Protocol Considerations

The command sockets on each agent should provide reliability as well as ordering of the command messages. Reliability is needed to ensure that the entire dynamic path has been configured properly, while ordering is required to maintain consistency between the rule sets of the command module and the relay agents. In addition, a production implementation must support authentication of command messages in order to provide security for the system as a whole.

TCP was chosen as the command protocol, as it meets the requirements for reliability and maintaining order. It should be noted that in this implementation, a complete connection is used for each message exchange (e.g., an Add Rule command message followed by the corresponding Acknowledgment). This decision was made to make implementing a multiple-instance command module more straightforward, should the need arise. Limiting each communication set to a series of single connections limits the number of intermediate states in which the agents could be. There is the added overhead of connection setup and tear-down for each message, though, and this cost will need to be investigated for scalability in large systems with complex and frequent rule message exchanges. Also, in the current implementation, no authentication takes place between the agents. Authentication needs to be present for a complete implementation.

3.2.3.4 Flexibility in the Type of Traffic Intercepted

In all instances (agents and command modules), care was taken to provide for adaptation in which transport protocols can be rerouted. Protocol is important, as, for example, port numbers are used to identify TCP connections. Knowledge of the transport protocol headers will be necessary in many cases to completely distinguish between flows to be rerouted and those to be left alone.

A general connection identifier, then, would be:

\[
< \text{ip.src} \mid \text{ip.dst} \mid \text{protocol} \mid \text{protocol data} >
\]
For TCP, this becomes:

\[
\langle \text{ip.src} \mid \text{ip.dst} \mid \text{IPPROTO_TCP} \mid \text{port.src} \mid \text{port.dst} >
\]

3.3 End Hosts and Applications

In order for this system to be easily deployed, the fewest possible requirements should be made on the end systems involved. It cannot be expected that all applications will be re-written to take advantage of the new rerouting abilities, nor can it be expected that administrative permission will be given on all hosts to install any extra software. For these reasons, this system makes no requirements on the end hosts or applications. Transparency is maintained by way of tunneling. It is possible, however, for a single host to act as an application endpoint, tunnel agent, and relay agent simultaneously.

It is assumed that the application developers have already made design decisions providing whatever properties are required in the communications of the applications. In order to be rerouted, the tunneling endpoints need only to be able to intercept whatever protocols the end applications are using. The most studied case in this solution is that of TCP, however other protocols would be intercepted in an identical manner.

3.4 User-Space Solution

The user-space relay based solution to our problem involves tunneling traffic in between any number of relay agents operating in user-space. Tunneling is needed in order to allow the relay agents to listen on a standard socket, without any special modifications or kernel module installations.

The user-space relay is designed to operate with minimal administrative requirements. This is meant to accommodate situations in which the relay agents are not located on hosts over which we have administrative control. The user-space relay may also be used any time that altering the NAT table isn’t possible or acceptable for any other reason.
3.4.1 Why Tunnel?

A tunneling mechanism was chosen as a means to relay the traffic. This approach allows the use of user-space relays as it provides a simple method for delivering traffic to user-space while requiring only basic socket opening privileges.

Encapsulating the original packets also maintains the original information present in the packet. Maintaining the original headers allows their use as connection identifying information. Altering their information through NAT, for example, requires another method for identifying connections, as source and destination addresses would be limited to the set of relay agent host addresses.

A final reason for tunnel usage is to allow for a less intrusive implementation. In some instances, it may not be desirable to alter the original IP fields any more than would be done normally by routers. Encapsulating the packet would again allow the original packets to traverse the dynamic rerouting system without being altered, in a manner which presumably cooperates better with stricter protocols. This final point also has the benefit of better maintaining the current separation of network layers.

Extra bytes required for encapsulation are a per-packet penalty, and will have some cost in terms of bandwidth usage. However, the nature of the rerouting problem and its solution involves tradeoffs: throughput for QoS, or slightly lesser performance for an ability to control routing. The increased overhead due to tunneling is a part of the performance tradeoffs associated with the use of this solution, and is taken into account in the results contained here.

UDP has been chosen as the encapsulating protocol. There are many encapsulation methods in today’s networks. IP-in-IP [28], Minimal IP-in-IP [29], GRE [17], or even a solution-specific encapsulation could be used. However, in order to have agents available on the widest array of systems, UDP encapsulation is used. If a relay is running in a low-privilege user mode, then reading packets directly off the wire will not be possible. Opening a UDP socket will allow the required packets to be delivered
to the agent directly, with the performance advantage of avoiding a search through all incoming packets.

UDP provides precisely what this solution needs, and nothing that it does not. Any communication requirements such as reliability and ordering have presumably been addressed by the end application developers. Network management issues such as congestion control have also been addressed in an end-to-end manner. In most cases, then, it would be at best redundant, and at worst counter to the needs of the application for a tunneling protocol to provide these features. It should be noted, however, that standard congestion control thinking may not apply to a rerouting situation. For example, the congestion avoidance mechanism of slow-start may not be available for determining a sending rate on the new path. Finally, UDP provides port numbers, which are required for the relay agents’ listening processes.

The use of UDP may cause some problems, though, when taking into account firewalls. Long periods of inactivity between relays may cause firewalls to drop subsequent UDP packets between them, depending on the rule-set’s actions on established and non-established connections. This problem could be solved with heartbeats along any paths set up in the system, but this solution would introduce extra load on the network regardless of actual communication.

3.4.2 Tunnel Agent Operation

The tunnel agent has a number of requirements, and may not be deployable on every host. The first, and most critical, requirement is also the most obvious: the tunnel must be located in such a place as to intercept the original traffic flow. Relays may be placed anywhere, egress tunnels may be placed anywhere, but ingress tunnel points must be along the original path.

The tunnel application will need the ability to insert rules through iptables in order to divert traffic which is to be tunneled. Also, a UDP socket will be needed to send the packets into the rerouting system (in the case of an ingress tunnel agent), or to receive the packets from the system (in the case of an egress tunnel agent). Both
types of tunnels will also need a command socket for rule updates. Finally, an egress tunnel agent will also need permission to write to a raw socket for decapsulation purposes.

The tunnel agent operates via three channels (see Figure 3.2). The first channel is a command channel and listens for any communication from the command module. The commands are agent-specific, meaning that tunnels receive individually tailored commands. The command syntax is of the form `<Connection Identifier | Forwarding Rule >`. The commands most commonly used are those to add and remove forwarding rules. Upon receiving the commands, the tunnel agent will make the necessary changes to its own forwarding table.

The second channel is a tunnel ingress channel. This channel watches all traffic coming through the tunnel host and intercepts any traffic which needs to be tunneled. If any traffic is found, it is brought to user-space for encapsulation and resending. This channel operates via a special hook in iptables, the specifics of which are discussed in Section 3.4.3. The final channel is the tunnel egress channel. This channel listens for encapsulated data. In the current case, this means that the tunnel opens a UDP port for listening. Upon receiving tunneled data on this port, the tunnel agent will do a lookup to determine whether this packet should be decapsulated. If so, the outer headers will be stripped, and the original IP packet will be re-injected through a raw socket onto the network for delivery.

When designing the tunnels, it makes sense to have them play a secondary role as user-space relay agents. Egress tunnel agents already have a UDP port for listening in the same manner as a relay, and ingress tunnel agents already have a UDP port for sending in the same manner as a relay. A routing table is already present in each as well. While using the tunnel agents as relays is not a requirement, the option is easily created.
3.4.3 Tunnel Implementation

3.4.3.1 Command Channel

The general properties of the command socket are as listed in Section 3.2.3.3. Both tunnel agents and user-space relays should support at least the following transactions via the command socket:

- **Add Rule** – Issued by the command module and received by the user-space relay, this message contains a Connection Identifier as well as next relay information.

- **Delete Rule** – Also issued by the command module, a Connection Identifier whose entry is to be deleted from the relay’s rule list.

- **Query Send Port** – A command module request for the relay’s transmission port. This is currently used for interoperability with kernel-space relays.

- **Acknowledgments** – Replies to the above messages, to supply the command module with information about the success(or failure) of the issued commands.
While the following transactions may also prove useful:

- **Heartbeat** – Heartbeat messages back to the command module may be useful for network management, as they could provide a feedback mechanism through the control module to the network resource manager.

- **Query** – A control module-initiated request for specific host information. For example, rule hit statistics, host load information, or other useful QoS metrics may be exchanged for the purpose of better total system management.

### 3.4.3.2 Ingress Channel

At the tunnel ingress, packets are intercepted through the QUEUE target in iptables. As commands for tunnel creation are received, iptables rules are generated to match the connection. Packets caught by the iptables rule are queued for user-space access through the QUEUE target. The QUEUE target represents a Netlink socket through which packets are passed from kernel-space to user-space. In the tunnel application, this queue is represented by a file descriptor, and messages are retrieved one by one. The entire IP datagram is passed to user-space, at which point it is sent back out via UDP to the next hop along the dynamic path. In order for the tunnel to see all packets which go through it, a new chain is inserted into the POSTROUTING hook of the mangle table. In this manner, not only will traffic being forwarded through the host be caught, but also any locally-generated matching traffic will be as well. This allows applications residing on the tunnel agent to be immediately entered into the dynamic routing system, while not requiring a tunnel agent on each host wanting to participate in the system.

### 3.4.3.3 Egress Channel

Tunnel egress begins in the same manner as user-space relaying. The original IP datagram is received and decapsulated through a UDP socket as normal. A forwarding lookup is then done on the connection identification. The procedure changes only when the resulting forwarding entry is flagged with a decapsulation value. At this
point, instead of being sent out via UDP, the packet is prepared for writing to a raw socket. The “Time to Live” (TTL) field of the IP header [32] is checked and decremented. The original packet is then written to the raw socket and default IP routing takes over, delivering the packet to the original destination.

If an alternative encapsulation is used, making it inconvenient to receive the packet at user-space for decapsulation, the QUEUE target used for tunnel ingress could also be used at the endpoint to pull the packets to user-space.

When using a series of tunnels, it is important to update and check the inner IP header’s TTL field. Otherwise, a loop in the tunneled route may lead to large amounts of traffic surviving indefinitely in the network. A new TTL is given to a tunneled packet every time it is re-sent, so the outer TTL only prevents routing loops between relays, not throughout the system.

Solving this problem, though, is straightforward. At each relay agent, decrement and check the inner header’s TTL field as a standard router would. If the TTL is zero, simply drop the packet. When the packet exits the dynamic rerouting system, the TTL will have been decremented once per relay.

### 3.4.4 User-Space Relay Agent Operation

The user-space relay is needed for situations in which we cannot or will not make intrusive changes in the packet filtering and NAT modules on the relay host. Primarily, it is envisioned to allow use on machines for which we cannot gain administrative privileges necessary for such changes.

The user-space solution is provided as a lowest-denominator relay. For this solution, the system need only be allowed to open two channels: a command connection and a forwarding connection (see Figure 3.3). The command connection is a TCP socket listening for messages from the command module. Commands should be authenticated before any changes are made to the local forwarding rules. The forwarding connection is a simple UDP socket. When a packet is forwarded to the relay agent, the data delivered through the socket will be the original IP datagram that was en-
capsulated by the ingress tunnel. From connection information present in this packet, the relay agent will do a lookup into the forwarding rules table and determine the next hop in the overlay network. With a UDP send call, the datagram is then forwarded to the next relay agent or tunnel endpoint.

Many of the features that this version of the solution provide can be useful. The operation of this relay is less intrusive upon the kernel operations of the host. No changes are required to the firewall. No special permission is needed to set the interfaces to promiscuous mode, nor for raw socket writing.

![Diagram of User-space Relay Operation]

Figure 3.3. User-space Relay Operation

### 3.4.5 User-Space Relay Implementation

#### 3.4.5.1 Command Channel

The implementation of the user-space relay’s command channel follows that of the tunnel agent. The exception is that user-space relays cannot process any add or
delete route commands which are flagged as tunnel starting or ending commands. See Section 3.4.3.1 for more information.

3.4.5.2 Forwarding Channel

The forwarding socket is implemented based upon the encapsulation protocol chosen. For this system, we chose UDP, and thus the forwarding socket is a standard UDP socket. Packets have their encapsulation stripped off of them as they are delivered through the network stack. Once the packet arrives in user-space, the payload received will be the entire original IP datagram intercepted by the tunnel ingress agent. From this datagram, the relay can parse out connection information, and perform a lookup on the local rule list. Once a corresponding forwarding rule is found, the data is resent to the next relay via a standard UDP send, getting re-encapsulated as it is passed down through the stack.

Early versions of the system included a user interface at each relay. While this is a useful tool for debugging, rule addition and deletion from the agent’s console are not necessary, and have ramifications concerning system consistency. Errors may occur if single rules are inserted or deleted outside of the command module’s control. Any implementation allowing this interface for rule addition and deletion must be aware of these consequences.

The user-space relay must keep track of its own rule set. A simple implementation for this is a linked list of forwarding entries, of the form <Connection ID | Forwarding Information>. The Forwarding Information for a next hop need only be the IP address of the next agent and the port which it is listening on.

There are a number of costs associated with using user-space relays. The first is the need for encapsulation. This brings with it extra overhead in terms of the encapsulation headers needed, as well as the computation time for switching them to user-space. Once in user-space, there is the added time needed to perform routing lookups. An analysis of these costs can be found in Sections 4.2.2 and 4.2.3.2.
In addition to these costs come a number of benefits. The structure of the user-space relay approach allows for deployment on a potentially wider range of hosts, as it includes those hosts for which we may not have administrative control. Also, in contrast to the kernel-space relay approach, there is a lower demand on the number of port numbers needed.

Implementing the user-space relay is straightforward. The ‘AddRoute’ command acts as a replace, in that if an old entry with an identical connection identifier is found, it will be deleted after inserting the new forwarding entry. Rule deletion is a simple list delete. Upon receiving a Query Send Port message, the reply port will always be the same, i.e. the UDP send port. Relay and command receipt operation is carried out with a select call, listening to the UDP port and to the listen port for TCP command connections.

For a more detailed view of control flow in the user-space relay, please refer to the appendix.

3.5 Kernel-Space Solution

The kernel-space solution is meant to be an optimization on the user-space solution. This optimization, though, comes at a price: higher permissions are needed. This is because the kernel-space solution utilizes pre-existing Network Address Translation (NAT) (See Section 2.1.4) technology. Relaying is achieved through a set of NAT rules based upon messages from the Command Module.

There is still a user-space portion to this relay, as there is a command socket opened to receive forwarding updates (see Figure 3.4). Upon receipt of these messages, the relay adds, modifies, or removes NAT rules through system calls to iptables. These NAT rules will effectively redirect the traffic to the next relay in the sequence by rewriting the IP addresses and port numbers in the packet headers. At each relay in the path, these addresses and ports will be altered so that all requested relays are visited in turn. The NAT rules inserted at the last relay agent will restore the original
IP addresses and port numbers, allowing default IP routing to take over and deliver the datagram to its final destination. A valid path for a kernel-space system does not need tunnel agents at the endpoints. The first relay in the path must be along the default route.

The immediate benefit provided by kernel-space relays is the lack of tunneling required in the basic case. This lowers the per-packet overhead. Also, avoiding a trip to user-space for forwarding lookups and packet sending is more efficient. Avoiding encapsulation also allows for other functions of IP to work effectively, such as accurate TTL tracking. Tunneling may be used, however, to allow a path consisting of a mix of user- and kernel-space relays, as well as to allow use of NAT-sensitive end-to-end protocols such as IPsec.

Drawbacks to a kernel-space solution include the access level needed to insert, change, and delete NAT rules. Also, in order to change the routing of a pre-existing connection, control must be allowed over the connection tracking table. In the case of
the testbeds presented here, this requires kernel patching, which may not be possible in all cases. Finally, also due to connection tracking, paths through kernel-space relays need to have symmetry immediately around the relay agents. This is because a rule to translate the addresses of certain packets in the forward direction will automatically be triggered for the reverse direction as well.

### 3.5.1 Kernel-Space Relay Implementation

Due to the header translations performed by iptables, the kernel-space relays do not have the original connection identifiers available to use as a key into a route lookup table. The source and destination IP addresses will belong to other relays (with the excepted case being that of the relay at the beginning of the dynamic path). The source and destination ports, then, become the identifying parts of the header. For this reason, a new port is allocated on each relay for every flow that is to be rerouted through that relay. This limits the scalability of the kernel-space relay based system when compared with the user-space relay based system. A single port number can be used to identify a maximum of 65,536 different flows. However, in many systems this limit will not be reached. A limiting factor on the number of independent flows through a host such as those used in the experimentation here is the size of the connection tracking table. The default size of this table for each host in the testbed is 30,712 connections.

For a more detailed view of control flow in the kernel-space relay, please refer to the appendix.

### 3.6 Command Module

The Command Module serves as an interface between the overlay network of relays and the system making route decisions. The requirements for the command module have been kept to a minimum to allow for easier integration with other systems.

In the general case, the Command Module will likely be the active arm of a network resource manager. This Network RM will make the routing decisions, and
will delegate the maintenance of the routes to the Command Module. For security, commands should be authenticated, and the Command Module could be designed in a decentralized manner.

The command module keeps track of tunnel and relay points, as well as the routes currently being used in the system. When new routes are specified to the command module, it checks them for correctness and transmits commands to the affected tunnel and relay agents to create the route.

### 3.6.1 Route Maintenance with User-Space Relays

#### 3.6.1.1 Adding Routes

The route creation algorithm is simple when using user-space relays. The command module builds these routes by reversing the path specification and transmitting “next hop” messages to the necessary relays. These messages consist of the IP address and UDP port of the next relay. Setting the routes up in reverse order allows the entire path to be ready when tunneling begins.

As an example, if the command module received a route add command for a connection with id \textit{ConnID}, with a route of $T1 \rightarrow R2 \rightarrow T3$ (signifying an order of hops along the path Tunnel 1, Relay 2, Tunnel 3), it would build a path as follows:

- Send Tunnel 3: $\langle \text{Add} \mid \text{ConnID} \mid \text{Tunnel~End} \rangle$

- Send Relay 2: $\langle \text{Add} \mid \text{ConnID} \mid \text{Forward~to}: \text{Tunnel}~3 \rangle$

- Send Tunnel 1: $\langle \text{Add} \mid \text{ConnID} \mid \text{Tunnel~Start} \mid \text{Forward~to}: \text{Relay}~2 \rangle$

#### 3.6.1.2 Deleting Routes

In order to delete a user-space route, the Command Module transmits messages to each tunnel and relay in forward order. These messages contain the connection identifier of the route to be deleted. To delete the route added in the previous section, the message $\langle \text{Delete} \mid \text{ConnID} \rangle$ will be sent to Tunnel 1, Relay 2, and Tunnel 3.
3.6.2 Route Maintenance with Kernel-space Relays

3.6.2.1 Adding Routes

Route setup for kernel-space relays is slightly more complicated than in the user-space case. This is because of the connection tracking required when using NAT. Route setup in this case is a four step process:

1. Get a port number from each relay host. Since there is no encapsulation header, the original connection identifying data is no longer present. In order to differentiate connections, we must utilize unique port numbers on the relay agents.

2. Create the NAT rules. Inform each relay of the incoming connection information to look for, as well as the outgoing information to translate to.

3. Activate the route. Delete the connection tracking entries for the connection at each relay, so that the new NAT rules will be used.

4. Add route to route table. Insert the new route into the global routing table.

3.6.2.2 Deleting Routes

Again, route maintenance is more complicated in the kernel-space relay case. Simply changing the NAT rules does not necessarily change the actions taken on a connection; deleting the connection tracking entry is necessary. Also, the ports used for connection identification must be closed.

1. Delete NAT rules on each relay host. Remove the NAT rules so that no newly matching connections will be rerouted via the old path.

2. Deactivate the route. Delete the connection tracking entries for any matching connections so that new routing decisions will be made for them.

3. Close ports. Close the ports used for this route on each relay, as they are no longer needed. This is required as a separate step from the previous two in
order to mitigate race conditions. If the ports are closed too early, rerouted traffic already present in the network may cause TCP resets, which could be propagated back to an end host. Closing the ports late in the delete procedure prevents this from happening.

4. Remove route from route table. Finally, remove the old route from the global routing table.

For a more detailed view of control flow and route setup in the Command Module, please refer to the appendix.

3.6.3 Command Module Design Decisions

The Command Module was designed in a centralized manner. In order to achieve (overlay-) network-wide route control, all network information will have to be available in one place. In the simplest case, a single application will have the task of managing routing decisions. Decentralized routing decisions may work well in traffic engineering systems which focus on achieving certain QoS levels (e.g. RSVP or MPLS). However, as discussed previously, we wish to provide control over the route itself, which requires more network knowledge and control at the decision-making application. A decentralized framework for a command module would also raise problems with consistency in forwarding tables, as well as race conditions in forwarding updates (i.e. if many command modules try to update the same connection’s path at the same time, in different orders). In order to avoid single-point-of-failure type problems, the current system is capable of allowing multiple command modules to issue commands, should the need for redundancy arise.

3.7 Analysis

3.7.1 Design

Architecturally, this system meets the goals and requirements set forth from the beginning. It is possible to dynamically reroute traffic at any time during the connection. The Command Module’s design allows for an interface with a resource
manager, allowing the RM to use constraint-based routing. Routes can be set up on
a per-connection basis. The system successfully maintains transparency to the end
hosts and applications.

Modifications to the network are kept to a minimum. The installation of relays is
the biggest change in the network. Routers remain functioning as normal. Operating
across the Internet is possible to the extent that UDP traffic is allowed. Administra-
tive control requirements are kept low, although in the kernel-space relay solution,
kernel patching may be required.

One performance-related issue that this overlay system’s design introduces is the
appearance of ‘doglegs’ in the paths. Relays are placed on hosts which are not core
routers. These relays may be connected to the Internet at a single point. If this is
the case, it means that traffic will have to traverse a single link twice (once to the
relay and once from it). This doubles the load on the routers moving the traffic, and
can halve the possible end-to-end throughput.

A solution to the dogleg problem is to multihome the relays on multiple networks
wherever possible. This option will not always be available, and may not even solve
the problem if it is. In general, the closer the relays are located to the core routers,
the shorter these dogleg paths will be.

3.7.2 Implementation

The implementation of this system also follows the goals outlined previously.
When possible, the system uses the complete connection information as an identi-
fier, allowing differentiation of multiple flows between two hosts. The use of target
port numbers to identify flows in a kernel-space relay path limits the scalability of
this approach. Many times, however, the limits on the size of the connection tracking
tables will be met before this is a problem.

UDP has been used as the encapsulating protocol without modification. In addi-
tion, only two major changes have been made to agent hosts: the inclusion of the
 QUEUE target needed for tunnel ingress agents, and the nfnetlink_conntrack patches required for the connection tracking entry deletion needed for kernel-space relays.

A shortcoming of the current implementation is the lack of authentication between the Command Module and the various agents. A second potential inefficiency is the use of an entire TCP connection per message exchange. Finally, support for path MTU discovery could be supported in order to mitigate the problems associated with fragmentation.

3.7.3 Tunnels and the Internet Control Message Protocol (ICMP)

An important function of tunneling that must be addressed here is the relaying of certain control messages back to the connection’s endpoints. The Internet Control Message Protocol (ICMP) [31] is used to convey administrative messages that are not part of an actual connection. These messages are related to the connection, however, and in some cases are critical to the success of the communication.

For this reason, the tunnels must address the forwarding of these ICMP messages. The matter of which ICMP messages must be forwarded has been addressed previously [28], and must be part of a full implementation of the dynamic routing system.

3.7.4 Some Notes on Security Implications

Security is a concern in any computer system. IP source routing was deemed an unacceptable solution due in part to security concerns. In order to use the same attacks on this system as used on IP source routing, the evildoer would need to make sure that there was a path back through his host. In order to do this, he would either have to fool the Command Module into believing that he was the network resource manager, or fool relays into thinking that he is the next hop. In order to do this, the evildoer would need to know connection identifying information, as well as the current route (or dynamic route) that it is taking. In short, these requirements amount to the same requirements needed to successfully perform a man-in-the-middle
attack. Authentication of all command messages in the system should prevent the system itself from being compromised, although any attacks possible on a default routed connection are still possible.

Another problem may come from the decapsulation of previously tunneled traffic onto a network. TUNneled traffic may have passed through firewalls which it would not have been able to had it been visible. This means that some harmful traffic may not be filtered. Also, the source address on the packets may not be valid on the current network, in the case where non-routeable addresses are used.

Finally, as mentioned previously, some care must be taken that protocols which do not coexist with NAT are tunneled before using any kernel-space relays. IPsec is the example most concerned with the security implications of this system. Tunneling only partially solves the problems associated with IPsec, however, since port numbers of the original connection may be encrypted. In this case, the granularity of which traffic can be intercepted is much coarser, as only source and destination addresses can be used.
4. Experimental Setup and Results

The previous chapter discussed the design and implementation of a system capable of providing dynamic routing abilities to a network resource manager. In this chapter, the system’s performance is measured and analyzed, beginning with throughput and CPU load affects in Section 4.2. Any per-relay penalties are investigated in Section 4.3, followed by a discussion of the effects that network delay may have on the system in Section 4.4. Finally, the effects of dynamically changing a connection’s path are investigated in Section 4.5.

4.1 Testbed Setup

The testbed network is made up of five identical machines. The configuration of these machines changes with the experiments, but the basic setup is a linear network. Each machine has an AMD Athlon™ processor running at 1.53 GHz, with 512 Megabytes of RAM. The machines are connected with crossover cables and link speed is 100 Megabits per second. The hosts are running the Fedora Core 1 Linux release, kernel 2.4.22-1.2199. The kernel-space relay requires the nfnetlink_conntrack iptables patch, in this case, version 0.13 was used.

Attached to this network are two communicating hosts. These hosts run on an Intel Celeron™ processor running at 566 MHz, with 128 Megabytes of RAM. The operating system on the end hosts is Fedora Core 2, kernel version 2.6.8-1.521.

Figure 4.1 shows the basic testbed used for benchmarking and performance analysis. Table1 through Table5 are the hosts capable of running tunnel and relay agents. Host1 and Host2 are communicating endpoints only. While this setup is simple, it
Figure 4.1. General Testbed. The simple testbed used for benchmarking and performance analysis

Table 4.1 Maximum and Average Throughputs for the Studied Routing Methods

<table>
<thead>
<tr>
<th>Route</th>
<th>Average (Max) Throughput, Mbps</th>
<th>Performance Penalty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>89.640 (89.644)</td>
<td>-</td>
</tr>
<tr>
<td>Kernel-space Relay</td>
<td>89.640 (89.645)</td>
<td>0.00%</td>
</tr>
<tr>
<td>User-space Relay</td>
<td>84.780 (84.787)</td>
<td>5.42%</td>
</tr>
</tbody>
</table>

serves its purpose well. The experiments which use this setup will not be changing paths, and so only need a linear network. More complex experiments will use a separate configuration.

4.2 Benchmarks and Performance Analysis

Throughput and CPU load are two areas of major impact for this dynamic routing system. In this section, the performance of the system is compared and contrasted with that of default IP routing.

4.2.1 Throughput

During steady-state, a change in attainable throughput is the most noticeable cost of the dynamic routing system. This may only be seen, however, when using a path through user-space relays. As shown in Table 4.1, the cost of a kernel-space relay in terms of throughput is often undetectable. A path using user-space relays and the
tunnels required for them, however, may see a performance penalty of greater than 5%.

For the purposes of Table 4.1, “Max Throughput” refers to the maximum throughput seen for a set of ten-second windows during the course of a connection. “Average throughput” is the throughput over the course of the entire connection. All results were the averages of ten connections. The kernel-space relay path was comprised of kernel relays located on hosts Table1, Table4, and Table5. The user-space relay path consisted of tunnel agents on hosts Table1 and Table5, and a user-space relay on host Table4.

### 4.2.2 CPU Load

Another cost associated with the use of the dynamic routing system is the CPU load introduced on the relay hosts themselves. The relay agents represent potential bottleneck points along a communication path, and the resources they require should be well known and planned for.

CPU usage statistics were monitored in order to determine the amount of CPU load that was caused by the dynamic routing system. The monitoring was performed by using vmstat at two-second intervals during the course of ten connections, and averaging the results. For these experiments, CPU utilization never reached 100%. An investigation into how pre-existing CPU load affects relay performance can be found in Section 4.3.

Low CPU utilization is another benefit of the kernel-space relay. The increased load on a host due to the NAT processing required for the kernel-space relays averages to 1.02%. This is a negligible amount, especially when considering that in some cases the redirecting host had lower CPU load than the same host when it was not redirecting.

The tunnel agents and user-space relays, however, have a much more dramatic impact on a host’s CPU utilization. When testing the tunnel agents only, the increase in CPU load on the tunneling hosts averages to 31.90% on the forward-path ingress
agent, and 26.20% on the forward-path egress agent. The discrepancy could lie in the different methods of receiving the majority of packets, as the ingress agent receives them through the QUEUE target in iptables, while the egress agent receives them through a UDP socket. In addition, the ingress agent must perform fragmentation, while the egress agent performs fragment re-assembly.

We must also consider the impact that tunneling has on the rest of the network. For example, when tunneling between Table1 and Table5, without relaying on any host in between them, the CPU load of the intermediate hops is 2.59% higher than when default routing is used. This is due to the increased number of datagrams caused by IP fragmentation.

We also find that the CPU utilization on an intermediate hop rises 34.11% when that hop acts as a user-space relay. In short, on a user-space relay based path traveling from Tunnel_1 to Relay_2 to Tunnel_3, Tunnel_1 will see an increase in CPU utilization of nearly 32%, Relay_2 can expect a rise of 34%, and Tunnel_3 may well see a cost of 26%. While the magnitude of these values will vary with different systems, the relative impact of the dynamic routing system will remain.

4.2.3 Analysis

4.2.3.1 Encapsulation and Fragmentation Overhead

Tunneling is performed by adding extra headers, and hence, extra bytes, to a packet in order to transmit the packet via a (potentially) different protocol. The tunneling required for the user-space relays is no exception to this rule. In addition to the overhead of these bytes alone, their addition will likely also cause fragmentation on any packets already close to the Maximum Transmission Unit (MTU).

The UDP/IP encapsulation chosen adds 28 bytes of headers to the datagram (8 bytes for the UDP header and 20 bytes for the IP header, see Figure 4.2). If this addition requires fragmentation, an extra 20 bytes will be needed in the form of the new fragmentation header (see Figure 4.3). Finally, an extra 38 bytes, comprised of
an Ethernet header and the inter-packet gap, will be needed for the new fragment. This total of 86 bytes is the header and transmission overhead introduced by encapsulation and fragmentation.

<table>
<thead>
<tr>
<th>IP</th>
<th>UDP</th>
<th>IP</th>
<th>TCP</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 Bytes</td>
<td>8 Bytes</td>
<td>20 Bytes</td>
<td>32 Bytes</td>
<td>up to 1448 Bytes</td>
</tr>
</tbody>
</table>

Figure 4.2. Packet Encapsulation

Fragmentation has long been known to affect throughput [22]. One reason is the added header overhead needed for transmitting the second segment. Another reason is the extra computational overhead needed for splitting packets, forwarding extra packets, and reassembling packets at the other end.

The communicating applications, however, are not the only things impacted by this overhead. The increased number of bytes and packets on the network affects all traffic, as network use is increased.

<table>
<thead>
<tr>
<th>IPfrag</th>
<th>UDP</th>
<th>IP</th>
<th>TCP</th>
<th>Data1</th>
<th>Ethernet</th>
<th>IPfrag</th>
<th>Data2</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 Bytes</td>
<td>8 Bytes</td>
<td>20 Bytes</td>
<td>32 Bytes</td>
<td>1420 Bytes</td>
<td>38 Bytes</td>
<td>20 Bytes</td>
<td>28 Bytes</td>
</tr>
</tbody>
</table>

Figure 4.3. Packet Fragmentation

The goal of this experiment is to quantify how much encapsulation and fragmentation affect the performance of the user-space relay, and to determine how much of the effect is due to header and transmission overhead, as well as how much can be attributed to processing overhead.
To test this experimentally, a linear network testbed was used. In the case without fragmentation, the MTU of the start and end links (the links from Host1 to Table1 and from Table5 to Host2) were set to 1400B. The MTU of the remaining links were set to 1500B. In this way, enough space was guaranteed along the tunneled path to accommodate the encapsulated traffic. For the case showing fragmentation, the every link along the path had its MTU set to 1400B. Tunneling agents operated on Table1 and Table5, and user-space relays operated on Table2, Table3, and Table4.

In addition to the above, the experiments have been run with 100 Mbps Ethernet, with frame sizes of 1438 Bytes (1400B MTU plus Ethernet headers and inter-packet gap). The user data in each packet is 1400B minus the IP headers (20 Bytes) and the TCP headers with timestamps option (32 Bytes), yielding 1348 Bytes. Taking all of this into consideration, we arrive at the maximum possible throughput of a default-routed flow by:

\[
\frac{100,000,000 \text{ bits}}{1 \text{ second}} \times \frac{1 \text{ packet}}{1438 \text{ Bytes}} \times \frac{1348 \text{ Bytes (of user data)}}{1 \text{ packet}} = 93,741,304 \text{ bits per second}
\]

This is the method used for the theoretical values listed in Table 4.2.

The theoretical maximum for TCP throughput in this experiment was 89.399 Mbps. Even default IP routing did not achieve this rate. However, theoretical values are used in order to compare expected and actual performances.

The fragmentation results initially seen were unexpected. For a path Host1 → Tunnel_1 → Relay_1 → Relay_2 → Relay_3 → Tunnel_2 → Host2, there are four instances of fragmentation/defragmentation: Tunnel_1 → Relay_1, Relay_1 → Relay_2, Relay_2 → Relay_3, and Relay_3 → Tunnel_2. For this reason, a performance penalty related to the number of relays used was expected. However, this is not the case. The major penalty measured was upon first use of the tunnel. After tunneling, any additional relays had only a minor affect on the throughput (see Section 4.3). This leads to the conclusion that the major impact of fragmentation, for this experiment, is not the CPU overhead associated with breaking up the datagrams and reassembling
Table 4.2 Encapsulation and Fragmentation Overhead. The effect of encapsulation alone is shown by the results of the non-fragmented tunnel. The fragmented tunnel shows both the effects of encapsulation and fragmentation.

<table>
<thead>
<tr>
<th>Tunnel Setup</th>
<th>Theoretical Maximum (Performance Penalty)</th>
<th>Experimental Maximum (Performance Penalty)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-tunneled (MTU: 1400B)</td>
<td>89.399 Mbps ( - )</td>
<td>89.246 Mbps ( - )</td>
</tr>
<tr>
<td>Non-fragmented Tunnel (End MTU: 1,400 B) (Tunnel MTU: 1,500 B)</td>
<td>87.691 Mbps ( 1.91% )</td>
<td>87.534 Mbps ( 1.92% )</td>
</tr>
<tr>
<td>Fragmented Tunnel (End MTU: 1,400 B) (Tunnel MTU: 1,400 B)</td>
<td>84.354 Mbps ( 5.64% )</td>
<td>84.086 Mbps ( 5.78% )</td>
</tr>
</tbody>
</table>

them. The major source of decreased performance is, instead, the increased amount of data and Ethernet overhead that is required as a result of the fragmentation.

Fragmentation requires that extra headers be sent for each packet, from the IP layer down. In addition to these headers, the Ethernet overhead also doubles as the number of packets. This is an additional 20 bytes minimum for the IP header, as well as a 38 byte overhead due to the Ethernet headers and inter-packet gap (assuming 100megabit Ethernet).

There are a number of reasons that the CPU overhead associated with fragmentation may not have been very detrimental to the throughput. The experiment was run on a silent network, a single flow with no competing traffic. The packets maintain their ordering. The first fragment will be followed by the second, minimizing any search time needed for re-assembly. In addition, the relay hosts are unloaded, except for the relaying itself. This makes an ideal environment for fragmentation and re-assembly to occur.
In order to test a more realistic CPU overhead due to fragmentation, we must load the network with IP fragments, and possibly also load the relay hosts with other processes. This might create a more visible connection between performance and the number of relays.

Table 4.3 demonstrates another effect of fragmentation on the relay system. Fragmentation increases the utilization of the agent’s CPU. The forward path ingress tunnel received MTU-sized packets and encapsulated them, requiring fragmentation upon sending. The egress tunnel received these fragments, defragmented them, and sent them without encapsulation or fragmentation. The egress tunnel also received the acknowledgments and encapsulated them, although due to their small size they did not need fragmenting. The user-space relay received, defragmented, fragmented, and sent the forward-traveling packets, while it received and sent the return path ACKs. The ingress tunnel also received the ACKs and decapsulated them, sending them without fragmentation. This processing accounts for the increased CPU load in the fragmenting case.

The difference in experimental and theoretical penalties in the fragmented tunnel case of Table 4.2 represents the effect of processing fragments during this experiment. From these results we see that an otherwise idle CPU will, in this case, prevent a major throughput penalty due to fragmentation processing. On a host with a higher
Table 4.4 Co-located User-space Relays and Performance. The number of user-space relays visited on a single host and the resulting connection throughput.

<table>
<thead>
<tr>
<th>Number of Relays</th>
<th>Throughput (kBps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10,850.7</td>
</tr>
<tr>
<td>1</td>
<td>10,849.5</td>
</tr>
<tr>
<td>2</td>
<td>10,845.3</td>
</tr>
<tr>
<td>3</td>
<td>8,439.4</td>
</tr>
<tr>
<td>4</td>
<td>7,052.4</td>
</tr>
</tbody>
</table>

background load, the increased computation required for fragmentation may produce noticeable performance penalties (see Section 4.3).

4.2.3.2 The Cost of a Trip to User-space

Repeated trips to user-space are one cause of increased delay and overhead for the user-space relays. In order to quantify this while minimizing the impact of propagation and transmission delays, a dynamic path was created with many relays on a single host. The tunnel endpoints remain on outside agents, but multiple relay agents existed on a single machine. Specifically, tunnel agents were located on Table 1 and Table 5, while up to four user-space relays were present on Table 2.

With regard to the impact of traveling to user-space, the results (Table 4.4) were meaningful only in the simplest cases. Once a number of relays were being used on a single host, the CPU load they caused became a hindrance. However, the results remain interesting due to the insight they provide into how well the user-space relay reacts to this type of CPU load, as the results differ from those seen later, in Table 4.5.

4.3 Per-relay Penalties

We have determined that there is a cost associated with using the relay system. Quantifying how costly each relay along a path is will help to better understand the
Table 4.5 Effect of the Number of User-space Relays and Host CPU Load on Throughput

<table>
<thead>
<tr>
<th>Route</th>
<th>Throughput (kBps)</th>
<th>Standard Deviation in the Loaded Case</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No Load</td>
<td>Load</td>
</tr>
<tr>
<td>t1t5</td>
<td>10,851.0</td>
<td>10,808.1</td>
</tr>
<tr>
<td>t1u4t5</td>
<td>10,850.9</td>
<td>10,126.2</td>
</tr>
<tr>
<td>t1u2u4t5</td>
<td>10,850.7</td>
<td>10,227.9</td>
</tr>
<tr>
<td>t1u2u3u4t5</td>
<td>10,847.1</td>
<td>10,588.8</td>
</tr>
</tbody>
</table>

Tradeoffs when using this system. Each user-space relay used will add a measurable amount of delay to the path, as well as introduce a potential bottleneck due to processing speed or buffer size. The following experiments attempt to show the difference between using a path which traverses a minimum number of relays and a path which uses a larger number of relays.

4.3.1 User-space Relay Penalties

In the case of the user-space relays, only small penalties are detectable when there is no load on the relaying hosts, as can be seen in Table 4.5. In order to further investigate a per-relay penalty, the CPU usage on the relaying hosts was raised through a series of file reads and writes.

The load caused by file I/O caused noticeable changes in throughput. These changes varied wildly. Causes for these variations could be losses at key moments during the connection’s lifespan, although it should be noted that the ten runs averaged in each case showed some consistency.

While these results are not a clear indication of a simple per-relay penalty, they do show that the performance of the relay system greatly depends on the performance of each relay. An increased number of relays, then, provides a larger number of potential bottleneck points along a dynamic path.
Table 4.6 Effect of the Number of Kernel-space Relays and Host CPU Load on Throughput

<table>
<thead>
<tr>
<th>Route</th>
<th>Throughput (kBps)</th>
<th>No Load</th>
<th>Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>k1k5</td>
<td>11,473.5</td>
<td>No change</td>
<td></td>
</tr>
<tr>
<td>k1k4k5</td>
<td>11,473.5</td>
<td>No change</td>
<td></td>
</tr>
<tr>
<td>k1k2k4k5</td>
<td>11,473.5</td>
<td>No change</td>
<td></td>
</tr>
<tr>
<td>k1k2k3k4k5</td>
<td>11,473.5</td>
<td>No change</td>
<td></td>
</tr>
</tbody>
</table>

### 4.3.2 Kernel-space Relay Penalties

Neither the number of kernel-space relays visited, nor the CPU load introduced in this experiment had any measured effect on the achievable throughput of a route through kernel-space relays (Table 4.6).

### 4.4 Delay Effects

An integral part of using user-space relays is that the packets must be taken into user-space at each relay, increasing the delay between the two endpoints (see Section 4.5.3). Most of the experiments presented here were conducted with a very low-delay network path. In this environment, the increased delay caused by the trip to user-space can be large in comparison to the end-to-end delay. The purpose of these experiments is to show that the results would hold for paths with longer round trip times (RTTs).

Using the Network Emulator (netem) [21] software, artificial delay was created along the communication path. A series of TCP transfers were then performed in order to determine the effect of delay on the dynamic routing system.

From Table 4.7, we see that the RTT value for the path affects the performance of the user and kernel-space relay systems in the same way it does for default routing. This is expected as long as there are large enough buffers available along the path.
<table>
<thead>
<tr>
<th>Delay (RTT)</th>
<th>Default Throughput</th>
<th>Kernel-space Relay Throughput</th>
<th>Efficiency</th>
<th>User-space Relay Throughput</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 ms</td>
<td>11,473.5</td>
<td>11,473.5</td>
<td>100.0000%</td>
<td>10,881.5</td>
<td>94.8403%</td>
</tr>
<tr>
<td>16 ms</td>
<td>11,452.4</td>
<td>11,452.3</td>
<td>99.9991%</td>
<td>10,862.8</td>
<td>94.8517%</td>
</tr>
<tr>
<td>32 ms</td>
<td>11,424.0</td>
<td>11,424.0</td>
<td>100.0000%</td>
<td>10,836.3</td>
<td>94.8560%</td>
</tr>
<tr>
<td>64 ms</td>
<td>11,360.7</td>
<td>11,360.6</td>
<td>99.9991%</td>
<td>10,777.9</td>
<td>94.8700%</td>
</tr>
</tbody>
</table>

However, performance may suffer depending on buffer size and delay. For example, to achieve the rates listed in Table 4.7 for a 16ms delay, the system would have to buffer $100 \text{Mbps} \times 0.016 \text{seconds} = 200,000 \text{Bytes}$ This buffer size would need to be available not only at the communicating endpoints, but also at each relay agent which receives the packets along the way (i.e. the user-space relays).

Providing this buffer space on all relay agents may prove problematic, for the very reasons that we wish to have a user-space relay. Maximum buffer sizes are administratively set, and may not be sufficient for connections with a high bandwidth-delay product.

As a brief note, the throughputs listed in Table 4.7 are the throughputs seen over the course of an entire connection, not the maximum throughput achievable. This is the cause for the gradual decrease in the results as the delay climbs. On links with higher delays, the process of slow start will take longer, bringing down the throughput when measured over the connection’s lifespan.

### 4.5 Effects of Dynamic Routing

The purpose of the dynamic routing system is to provide a means for changing the paths which communicating hosts use. The following experiments explore the behavior of redirected traffic as well as the costs and benefits derived.
4.5.1 A Second Testbed

In order to test the effect of dynamically changing the path a connection uses, a testbed with multiple paths is required. Figure 4.4 shows the second testbed. This testbed has two available routes. The default routing tables direct traffic from Host1 to Host2 via Table2. The path through Table3 and Table4, in addition to having an extra host, also has had delay introduced through the use of netem. Traffic leaving Table3 has a tunable delay applied, although for most experiments the delay was set to 32ms. One should note that this delay is for traffic leaving Table3 only. Switches have been added to the testbed. The switches are Cisco Catalyst 3000s, and operate at 10 Mbps, half-duplex.

![Diagram of testbed allowing for change in path]

Figure 4.4. Testbed Allowing for Change in Path. Delay has been introduced via netem on Table3

4.5.1.1 Additional Information on the Switches

In addition to the bottleneck speed of 10 Mbps, the switches introduce another problem to the testbed: half-duplex communication. This further limits the throughput, as the first testbed operated at 100 Mbps, full-duplex. The problem, though, is that the switches do not correctly operate at half-duplex (or full-duplex, for that matter). Packet collisions and framing errors appear with the use of the switches.
However, in order to simulate a real network with alternate paths, their inclusion is necessary.

4.5.2 Changing the Dynamic Path

Before investigating the effects of dynamically routing a connection onto a different path, we should understand the effect of changing the connection’s path through the relay system. In order to determine this, we change a dynamic path such that the links and hosts traversed remain the same, but the relay agents that are visited change. For example, the communication’s path may always cross Table2, but Table2 may or may not be acting as a relay point for the connection.

4.5.2.1 Intercepting a Connection

Figure 4.5 shows the changes that can take place to the round trip time of a connection due to its interception by the dynamic routing system. Upon interception, the sending rate drops slightly as a result of the increased RTT, although it will rise if the window can grow. The acknowledgments take longer to arrive, and thus new packets are clocked out at a slower rate. The drop in sending rate can also be attributed to the added overhead of the relay system, discussed previously.

The impact on a connection being routed onto a kernel-space relay path is due to dropped packets. In order to assure that the routes are changed correctly, there is a temporary period during which the connection’s packets are dropped by the relays. While this technique is successful in its goal of preventing false resets being relayed to the end hosts, a side effect is up to a window’s worth of packet loss.

4.5.2.2 Changing a Connection’s Dynamic Route

Changing a dynamic route can have the same effect as interception, as the path characteristics are still likely to change. In the case of the user-space relayed route, the effect may be smaller, since no additional encapsulation or fragmentation will be needed. All extra byte-wise overhead is introduced with the tunneling, and not
affected by additional hops. Since the physical path remains unchanged, any change in these experiments is due to the additional processing needed at intermediate hops.

4.5.2.3 ‘Freeing’ a Connection

Removing a connection from the dynamic routing system allows default routing to take over. At this point, the connection should perform no different than it would have had it never been intercepted.

Figure 4.6 shows the effect of ‘freeing’ a connection from the dynamic routing system in terms of throughput. The dynamic route was released approximately ten
seconds into the connection. The sending rate increases from that point, due in the most part to the lack of encapsulation and fragmentation overhead.

### 4.5.3 Round Trip Time

The round trip time for a connection can have many implications for its performance. For TCP, a longer round trip time means a longer period of slow start, since each ACK takes longer to get back to the sender. One cause of the differences in throughputs seen in the experiments presented here is a longer RTT for packets relayed via user-space. The round trip times for connections relayed through user-space are also more variable due to operating system scheduling characteristics.
Table 4.8 Round Trip Times for Varying Relayed Paths

<table>
<thead>
<tr>
<th>Tunnels and User Relays</th>
<th>Average RTT</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.682 ms</td>
<td>0.034 ms</td>
</tr>
<tr>
<td>2</td>
<td>1.024 ms</td>
<td>0.052 ms</td>
</tr>
<tr>
<td>3</td>
<td>1.139 ms</td>
<td>0.103 ms</td>
</tr>
<tr>
<td>4</td>
<td>1.154 ms</td>
<td>0.439 ms</td>
</tr>
<tr>
<td>5</td>
<td>1.277 ms</td>
<td>0.478 ms</td>
</tr>
</tbody>
</table>

Table 4.8 shows some RTT values for a set of relayed paths. The values were found by relaying ping traffic through the dynamic routing system.

Changes to a connection’s path will also affect the communication in varying ways based upon the underlying protocol. For example, TCP uses a RTT measurement as part of it’s calculations for a retransmission timeout [2]. Because the round trip time of a path can have an effect on a connection’s performance, the following experiments explore the impact that significant changes in path characteristics might have.

4.5.3.1 New Paths with Longer RTTs

Rerouting is expected to have some impact on the end-to-end connection and performance of the communication. For example, a TCP connection rerouted onto a slower link may cause a retransmission timer to expire, causing a connection to go into loss recovery as a result of the sudden increase in RTT. These possibilities should be examined to ensure that the benefits of rerouting outweigh the costs.

For this experiment, a connection was started along the default path (although through the dynamic routing system). Approximately 60 seconds into the connection, the flow was redirected along the alternative path, whose one-way delay was set to 400 ms.

The connection, shown in Figure 4.7, experienced no retransmission timeout. Instead, after a slight delay, the connection continued at a slower rate. The delay is
caused in part by the loss of a group of packets. This loss occurs as part of the route change. Once the dynamic path is altered, any packets traversing the old path may be dropped. These losses require retransmission, causing an anomaly in the throughput graph (Figure 4.7, at time t=1:10).

Figure 4.7. Throughput Response to a New Path with Longer RTT. The connection is rerouted at time t=1 minute.

4.5.3.2 New Paths with Shorter RTTs

While rerouting onto a slower path may cause retransmission timers to expire, rerouting onto a faster path may cause packet reordering or buffer overflow. Traffic on the new, fast path might arrive at the destination before old traffic still traveling
the slow path. This reordering may trigger enough duplicate acknowledgments to cause an unnecessary fast retransmit on the part of the sender.

A second possible problem when moving to a faster path is that the traffic on the new path may arrive at the destination at the same time as previous packets sent on the slow path. In the best case, the sudden influx of traffic will be absorbed by the receiver’s buffers. However, the buffers may overflow and packets may be dropped.

Figure 4.8 show the throughput seen for a connection that begins using the 800ms RTT path. At approximately 60 seconds, the flow is then redirected over the shorter path. The performance improves markedly as a result.

Figure 4.8. Throughput Response to a New Path with Shorter RTT. The connection is rerouted at time t=1 minute.
Examining a time-sequence graph [27] (Figure 4.9) of this connection reveals some of the unintended effects of rerouting traffic. The data being sent is represented by the diagonal line starting near the x-axis at 63.50 seconds, and proceeding right and upwards. At this point, traffic redirection begins. The process starts by redirecting the return (ACK) path. This means that suddenly, ACKs are returned very quickly and TCP’s estimate of the appropriate round trip timeout decreases. The first ACKs shown in Figure 4.9 show this, as they appear at a time 63.67 seconds, an RTT much shorter than the 800ms seen on the long path. However, these ACKs cause the sender to believe that some earlier data packets were lost, triggering retransmission at 64.20 seconds. However, this is incorrect, since ACKs are received twice for these data ranges, evidenced by the vertical line at 64.38 seconds, as well as the DSACK blocks at 64.40 and 64.62 seconds. The act of changing paths, then, can potentially cause spurious retransmissions in TCP due to the sudden changes in the connection’s timing patterns.
Figure 4.9. Time-Sequence Graph of a Switch to Shorter Path. The switch takes place at time $t = 63.5$ seconds.
5. Conclusions and Future Work

5.1 Conclusions

Resource management is an important facet of distributed systems. There have been many projects aimed at providing control over host-based resources, as well as a number of projects whose goal is to provide QoS guarantees as a means of providing network resource management. This work, however, provides a resource manager with another method for managing network resources: control over the routes themselves. Constraint-based routing is possible based upon any metric the resource manager understands.

The overlay network based solution presented here allows for immediate use, free from any network upgrade requirements. In addition, the implementation of a kernel-space relay agent as well as a user-space relay agent provides a way to deploy the system over a broader range of administrative privileges.

We have shown that the implementation of this system provides the dynamic routing capabilities as intended. We have also shown that in the ideal case, a dynamic path will consist of kernel-space relay agents, as the overhead they introduce is minimal. Further, we have shown that the system still functions for the user-space relay based paths, although often with significant costs to the connection. These costs have been shown to be due, in large part, to the overhead associated with encapsulation and fragmentation. While these aspects are fundamental to the general design, we have quantified their impact so that educated decisions can be made regarding the use of this dynamic routing system.
This solution is not suitable for all distributed systems. However, for many cases this may be a viable solution to a network management problem. In particular, this system is best suited for systems using networks that are not managed in-house, but which can have many member hosts distributed across the network.

5.2 Future Work

There are a number of directions in which this work can be extended and expanded. The performance of user-space relay based paths could be greatly improved if the encapsulation and fragmentation overhead could be mitigated. Possible solutions to this problem may include a different encapsulation protocol, although this may prevent the user-space relay from easily receiving packets through a standard socket, as the current implementation allows. One possible alternative which would allow for the continued use of UDP is Minimal Encapsulation within IP. This encapsulation effectively replaces the outer, full IP header with an inner, 12 octet minimal header.

Support for Path MTU Discovery [25] may help to alleviate the problems which stem from fragmentation, at least in those cases where the communicating hosts use this process.

The kernel-space relay could be improved through a better interface to the operating system. The current implementation uses a relatively inefficient set of calls to the system() function in order to alter the NAT table. A relay with a better OS interface may help minimize the CPU load this system demands.

While the system presented here was designed to allow straightforward interfacing with a resource manager, little research was done into incorporating this system into an actual resource managing suite. This process may introduce additional features or requirements, such as feedback from the relay agents to the resource manager, or additional routing commands. Also, the scalability of the system will need to be examined in a production network. While global scalability is not the goal of this work, the limits of the system should be known.
Additionally, some security features such as command authentication are necessary and need to be added to the system for any production-type use. Other security features, however, such as encryption, may be useful as well. The costs of implementing these features must also be taken into account when judging the impact of the dynamic routing system.

Finally, research could be done into when a connection should be dynamically redirected, and the impact that this has for different types of applications. For example, perhaps there are two thresholds: one for signaling that the RM should start seeking a new path, and one for demanding a new path. The question of how to best use dynamic routing capabilities is an open and interesting one.
Bibliography


http://developer.osdl.org/shemminger/netem/.


Appendix: Implementation Figures

The following sections provide a more in-depth view of the operation of individual agents. Figures A.1 and A.2 detail the operation of both the tunnel agents and the user-space relays. The shaded portions of these two flowcharts indicated branches needed only for tunnel operation. Figures A.3, A.4, and A.5 represent the operation of the kernel-space relay. Portions of the Command Module’s operation are shown in Figures A.6 and A.7, demonstrating the route addition and route deletion processes.

Finally, Figure A.8 shows the communication which takes place among the Command Module and a set of kernel-space relay agents over the course of a route setup.
Figure A.1. User-space Relay and Tunnel Flowchart. Shaded portions correspond to sections present in tunnels only.
Figure A.2. User-space Relay and Tunnel Flowchart, Process Command Branch. Shaded portions correspond to sections present in tunnel only
Figure A.3. Kernel-space Relay Flowchart
START

connID in port db? no

connID in forward db? yes

connID in port db? yes

newentry = remove connID from port db

update newentry with rules

add NAT rules

oldentry? yes

delete old NAT rules

insert oldentry into close db

insert newentry into forward db

END

yes

sendACK(err)

no

oldentry = remove connID from forward db

Figure A.4. Kernel-space Relay Flowchart: Add Route Command
Figure A.5. Kernel-space Relay Flowchart: Delete Route Command

connID: Connection Identifier

START

connID in forward db?

connID in port db?

delete NAT Rules(rule)

remove from forward db

insert into close db

sendACK(ok)

sendACK(err)

remove from port db

insert into close db

sendACK(ok)

END
Figure A.6. Command Module Flowchart: Add Route Command
Figure A.7. Command Module Flowchart: Delete Route Command
### Figure A.8. Command Module Kernel-space Relay Route Setup

<table>
<thead>
<tr>
<th>Command Module</th>
<th>Relay B</th>
<th>Relay C</th>
<th>Relay D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get Command: Route A:1 F:2 -&gt; B, C, D send(D, port query)</td>
<td></td>
<td></td>
<td>create socket, bind a port (3) send(CM, 3)</td>
</tr>
<tr>
<td>set D.natport=3 send(C, port query)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>set C.natport=8 send(B, port query)</td>
<td></td>
<td>create socket, bind a port (8) send(CM, 8)</td>
<td></td>
</tr>
<tr>
<td>set B.natport=7 send(D, C:8</td>
<td>D:3 -&gt; A:1 F:2) create socket, bind a port (7) send(CM, 7)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>send(B, A:1 F:2 -&gt; B:7</td>
<td>C:8)</td>
<td>Add NAT Rules: C:8</td>
<td>B:7 -&gt; C:8</td>
</tr>
<tr>
<td>send(B, delete conntrack A:1 F:2)</td>
<td>ct_del(A:1 F:2) ct_del(F:2</td>
<td>A:1)</td>
<td></td>
</tr>
<tr>
<td>send(C, delete conntrack A:1 F:2)</td>
<td>ct_del(A:1 F:2) ct_del(F:2</td>
<td>A:1)</td>
<td></td>
</tr>
<tr>
<td>send(D, delete conntrack A:1 F:2)</td>
<td>ct_del(A:1 F:2) ct_del(F:2</td>
<td>A:1)</td>
<td></td>
</tr>
<tr>
<td>add route to database</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>