RICE UNIVERSITY

New Approaches to Routing for Large-Scale Data Networks

by

Johnny Chen

A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE

Doctor of Philosophy

APPROVED, THESIS COMMITTEE:

Dr. Peter Druschel, Chair
Assistant Professor in Computer Science

Dr. Devika Subramanian
Associate Professor in Computer Science

Dr. Edward Knightly
Assistant Professor in Electrical Engineering

Houston, Texas

June, 1999

New Approaches to Routing for Large-Scale Data Networks

Johnny Chen

Abstract

This thesis develops new routing methods for large-scale, packet-switched data networks such as the Internet. The methods developed increase network performance by considering routing approaches that take advantage of more available network resources than do current methods. Two approaches are explored: dynamic metric and multipath routing. Dynamic metric routing provides paths that change dynamically in response to network traffic and congestion, thereby increasing network performance because data travel less congested paths. The second approach, multipath routing, provides multiple paths between nodes and allows nodes to use these paths to best increase their network performance. Nodes in this environment achieve increased performance through aggregating the resources of multiple paths.

This thesis implements and analyzes algorithms for these two routing approaches. The first approach develops hybrid-Scout, a dynamic metric routing algorithm that calculates *independent* and *selective* dynamic metric paths. These two calculation properties are key to reducing routing costs and avoiding routing instabilities, two difficulties commonly experienced in traditional dynamic metric routing. For the second approach, multipath routing, this thesis develops a complete multipath network that includes the following components: routing algorithms that compute multiple paths, a multipath forwarding method to ensure that data travel their specified paths, and an end-host protocol that effectively uses multiple paths.

Simulations of these two routing approaches and their components demonstrate signif-

icant improvement over traditional routing strategies. The hybrid-Scout algorithm requires 3-4 times to 1-2 orders of magnitude less routing cost compared to traditional dynamic metric routing algorithms while delivering comparable network performance. For multipath routing, nodes using the multipath protocol fully exploit the offered paths and increase performance linearly in the additional resources provided by the multipath network. The performance improvements validate the multipath routing algorithms and the effectiveness of the proposed end-host protocol. Furthermore, this new multipath forwarding method allows multipath networks to be supported at low routing costs. This thesis demonstrates that the proposed methods to implement dynamic metric and multipath routing are efficient and deliver significant performance improvements.

Acknowledgments

This thesis is the culmination of many years of academic and personal education. I would like to take this opportunity to thank the many people who have made significant contributions to my educational process.

First and foremost, I am indebted to my parents. I am especially grateful to my mother, who made countless sacrifices to ensure that her children receive the best opportunity to live full and joyful lives. My brother Steve and sisters Aileen and Jennifer have also given me an immeasurable amount of fun, love, and advice. My family's constant support provided a level of personal and educational development that I would otherwise never have received. This thesis is a tribute to the positive influences they have made in my life.

I also thank my advisors Peter Druschel and Devika Subramanian. Their high standards of research sharpened not only my problem solving skills, but also my ability to effectively convey technical and complex ideas. In addition, they gave me complete freedom to pursue my research interests and were open to projects outside their core research areas. Of course, this thesis would not have been possible without my advisors' help and guidance. I also thank my third committee member, Ed Knightly, for his assistance in my research.

My high school teachers Mrs. Cynthia Erdei, Mr. Lynn Rosier, and Mrs. Ann Koenig played a significant role in my decision to pursue higher education. Their dedication to teaching and challenging young minds opened doors for me to attend college. I wish for their continuing success in teaching and positively shaping young lives.

I am grateful for the friends who have enriched my life: friends Bo and Jerry, and college friends Judd, Cash, Jeff, Eddie, and Deanna. These friendships have become stronger over the years and will hopefully continue to develop. I have also made many life-long friends while at Rice, including lunch buddies Zoran and Dejan; officemates Zhenghua, Mohit, and Henrik; and good friends Cristiana and Phil. Others friends include Anita, Gaurav, Honghui, Weimin, Bo, Karthick, Ram, Mike, Collin, James, Shriram, Robby, John, Matthew, Cathy, Fay, Darren, and Mary. Each one of these friends has made my life in Houston very enjoyable.

Last but not least, I thank my writing instructor Dr. Jan Hewitt for significantly improving my writing skills and her tireless evaluations of my thesis.

Contents

Illustrations

Tables

Chapter 1

Introduction

Large-scale, wide area data networks are a part of today's global communication infrastructure. Networks such as the Internet have become an integral medium of information transfer, ranging from personal communication to electronic commerce and entertainment. The importance of such networks will only increase as the electronic world becomes more prevalent.

The basic function of a data network is very simple: delivering data from one network node to another. Achieving this goal requires many network components, including physical computers and links, signaling protocols between computers, and data packaging protocols. This thesis addresses one such component, routing, the process that logically connects network nodes by calculating paths between nodes so that data sent by one node traverses the calculated path to its destination.

Although many algorithms in graph and operational research literature calculate paths between nodes, the challenge in developing network routing algorithms is in dealing with the scale and distribution of the physical network. Because typical wide area networks have nodes on the order of tens of thousands, routing algorithms must be scalable. In addition, routing algorithms must be able to calculate paths in a distributed manner due to the global and distributive nature of physical networks. Moreover, because of the actual physical network, routing algorithms need to cope with events such as physical component failures and recalculate paths whenever such events occur. Finally, routing algorithms need to calculate paths to allow nodes to achieve high network performance.

In general, routing algorithms view a network as a weighted graph, where network links are represented as graph edges and network routers as graph vertices. Network routers are network nodes that execute routing algorithms and ensure that data travel the calculated paths. In the weighted graph, the assignment of edge weights depends on the specific routing algorithm; typically, the assignment reflects the latency and bandwidth of the link [94]. After a routing algorithm makes these link cost assignments, it then computes paths between nodes. Thus, the specific routing algorithm that routers execute determines the paths that data will travel in the network.

Routing algorithms in today's Internet base their implementations on the *static metric single shortest path* routing model. Single shortest path means that routing algorithms provide, at any given time, the least-cost path between nodes. Static metric refers to link cost assignments which are based on static properties of a link, such as its bandwidth or latency. As shown later, the main drawback of this model is that static metric shortest paths do not always provide good network performance.

Although Internet routing algorithms use static metrics, this does not imply that the paths themselves are static. On the contrary, current Internet routing algorithms are *adaptive*, meaning they are able to recompute paths and reroute packets when network components (nodes or links) fail or recover. Therefore, even if routers or links fail, as long as a path exists between a node pair, Internet routing algorithms ensure that these two nodes can communicate with each other. Figure 1.1 shows the conceptual Internet routing model.

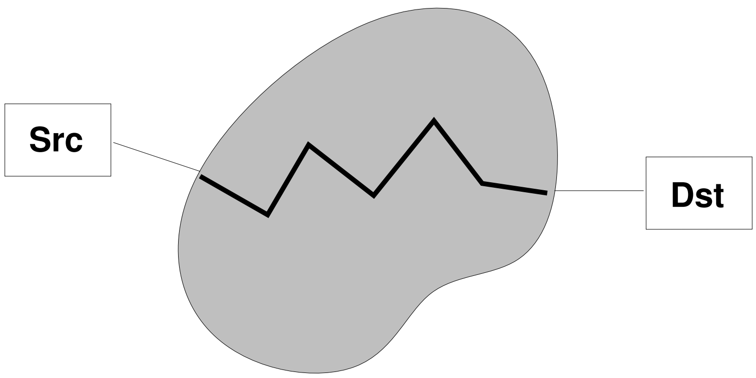


Figure 1.1 : The static metric single shortest path routing model. The solid line denotes the single path provided between Src and Dst.

The solid line in Figure 1.1 shows the shortest path calculated between Src and Dst. The shaded area denotes the network resources (network links and routers) that are not allocated for this communication channel. In this figure, messages sent by Src to Dst travel the solid path. Since the shortest path is calculated between nodes, packets travel to their destinations using paths that minimize the statically assigned cost. If link costs are uniform, then the shortest path minimizes the number of links and routers traversed. In general, a link's cost describes some notion of performance, usually given as a combination of the link's delay and bandwidth. For example, in the Internet, a link has a lower cost compared to the cost of another link with higher delay and lower bandwidth. Here, a link's cost reflects a notion of performance in terms of the amount of time expected for packets to traverse the link. With this cost assignment (expressed in terms of path delay and bandwidth), the shortest path minimizes the "expected time" for packets to reach their destinations.

However, performance measures based on the time a packet reaches its destination depend not only on the static properties of the links the packet traverses, but also on those links' utilization levels. For example, it takes less time for a packet to traverse a path when the path's links are idle than when the links are *congested*[*]. In fact, work by Khanna and Zinky [94] showed that computing paths that factor the dynamics of link utilization has a large impact on network performance.

To provide better network performance, methods developed in this thesis implement two routing models that deviate from the static metric single shortest path routing model. These two models are *dynamic metric single path routing* and *static metric multipath routing*. The methods developed improve upon previous methods that implement the two routing models. Conceptually, the two routing models increase network performance by effectively utilizing currently unallocated network resources (links and routers). These un-

[*]A link is in a congested state when too many messages are contending for the link's resources (e.g. link bandwidth and allocated router buffer space for the link). In this scenario, the router buffering this link's out-going data may be forced to drop messages out-going on the link.

allocated resources are represented by the shaded region in Figure 1.1.

The first model, dynamic metric single path routing, provides one path between node pairs, where the path computed considers network traffic and congestion. Dynamic metrics are defined as link cost metrics that change dynamically. In the context of this thesis, a link's dynamic cost is based on the link's utilization. With this type of metric, a link's cost is higher when it is experiencing congestion than when it is idle. In the dynamic metric routing model, routers recompute least-cost paths between nodes in response to dynamic changes in link costs. Because dynamically least-cost paths consider network traffic, dynamic metric routing offers the ability to provide nodes with higher performance paths (in terms of lower delay and/or higher throughput). Figure 1.2 shows a conceptual diagram of the dynamic metric single path routing model.

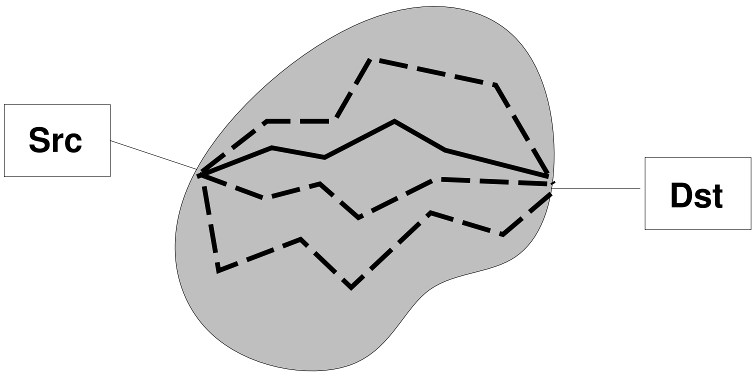


Figure 1.2 : A conceptual diagram of the dynamic metric routing model. The solid line denotes the current least-cost path between Src and Dst, and the dotted lines represent potential least-cost paths depending on dynamic link costs.

In this figure, the solid line represents the current path a dynamic metric routing algorithm provides between Src and Dst, and the three dotted lines denote potential paths Src has to Dst, depending on different traffic conditions. In a dynamic metric single path routing model, Src has available, at any one time, only one path to Dst. This path is chosen by the routing algorithm to carry data between Src and Dst. Because the routing algorithm continually recomputes the least-cost dynamic metric paths, the chosen path between nodes reflects the current least-cost path. Notice that the path Src has to Dst depends on network traffic and the routing algorithm's recomputation speed.

The second routing model considered in this thesis, the static metric multipath routing model, deviates from static metric single shortest path by providing multiple paths between nodes. Routing algorithms in this model provide potentially multiple paths between nodes concurrently, thereby increasing a node's available resources and allowing the node to use the multiple paths in ways that best increase its performance. Figure 1.3 shows the conceptual diagram of this routing model.

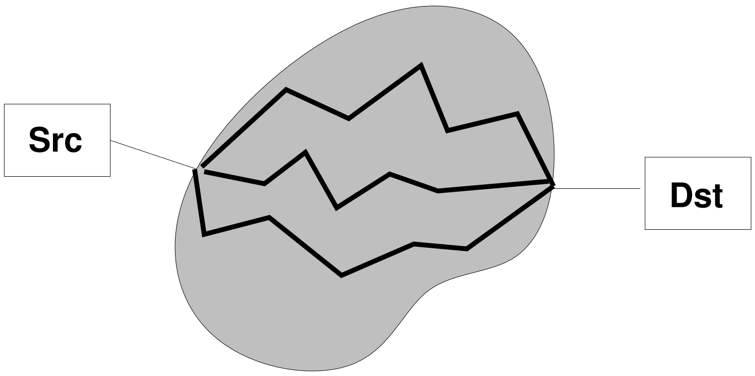


Figure 1.3 : A conceptual diagram of the static metric multiple path routing model. The three solid lines denote the paths that a multipath network provides between Src and Dst. Here Src can send data to Dst using any of the three paths.

In Figure 1.3, the solid lines represent the available paths Src has to Dst. In this example, a multipath routing algorithm provides three paths between Src and Dst. Additionally, the model allows Src to decide how to send data on these paths. For example, Src can send data to Dst on all three paths simultaneously, one path at a time, or any combination it chooses. This model allows higher performance because Src can dynamically detect and use path(s) that best maximize Src's performance.

The ability to choose which path to use differentiates the solid lines in multipath routing and the dotted lines in dynamic metric routing (for convenience, we use "dynamic metric routing" instead of "dynamic metric single path routing" and "multipath routing" instead of “static metric multipath routing”, unless otherwise specified). In multipath routing, the decision of which path to use is delegated to the sending nodes (end-hosts or the applications running on the hosts). That is, multipath routing algorithms calculate multiple paths between nodes, and the end-hosts choose which path(s) to use. In contrast, dynamic metric routing algorithms provide, at any give time, the only one path between node pairs. Thus, end-hosts in this environment do not have the option to choose among the paths the routing algorithm might calculate.

Although the general multipath routing model can also use dynamic metrics (the dynamic metric multipath model provides multiple paths between nodes that consider traffic patterns), this thesis primarily focuses on the static metric multipath routing model. The reason is that if a static metric multipath routing algorithm calculates the appropriate paths *and* if end-hosts manage these paths effectively, then the situations where dynamic metrics can benefit are significantly reduced. For example, assume that in Figure 1.2, a multipath routing algorithm provides all four paths (both dotted and solid paths) between Src and Dst. In this case, even if the multipath routing algorithm uses dynamic metrics, the opportunity to recalculate paths are much reduced because the possible dynamic metric candidate paths are already provided. Moreover, if end-hosts effectively use multiple paths to avoid congestion, then link congestion is less likely to occur which, in turn, reduces path recomputations. This issue is discussed in detail in Section 4.2.5.

Notice that the two routing models described are still adaptive to network changes. That is, both dynamic metric and multipath routing models recalculate paths upon detection of network component failures or recoveries. Therefore, methods proposed in this thesis are adaptive in that they offer the same connectivity guarantees as the current Internet routing model, but they promise higher achievable network performance.

1.1 Contributions

The contributions of this thesis stem from the development of methods that implement the dynamic metric and multipath routing models. For dynamic metric routing, a novel algorithm is developed which outperforms previous dynamic metric algorithms and requires lower routing costs. For multipath routing, the major contributions are an efficient path forwarding method and a multipath transport protocol that maximizes end-to-end throughput. The contributions of this thesis are summarized below.

Contribution 1: A novel dynamic metric routing algorithm that outperforms previous dynamic metric routing methods.

Dynamic metric routing has been implemented in the past; however, previous implementations suffer two problems: routing instability and high routing overheads [94]. Routing instability occurs when paths are constantly being recomputed and do not stabilize. This instability degrades network performance and increases the probability of network congestion and failures. The second problem is high routing overheads. The overheads are in terms of the CPU cycles and messages required for path recomputation. Because routing overheads in traditional dynamic metric algorithms depend on network traffic patterns, mechanisms such as thresholding and hold-downs [74] are needed to limit the amount of routing overhead; however, the effects on route quality of these two mechanisms are uncertain because routers often compute paths based on out-dated information of the network state.

The new dynamic metric routing algorithm developed in this thesis reduces the above problems. The algorithm, hybrid-Scout, reduces routing instabilities by using two techniques: time staggered and selective dynamic metric path computation. Simulations show that hybrid-Scout reduces the amount of routing cost by as much as an order of magnitude while maintaining performance comparable to traditional dynamic metric routing algorithms. This new algorithm promises to provide the benefits of dynamic metric routing without the undesirable side-effects.

Contribution 2: An efficient multipath forwarding method.

One of the main challenges of multipath routing is the forwarding of data on their intended paths. Multipath forwarding is difficult because each node has potentially multiple paths to a destination; therefore nodes have to label data packets to indicate which path a packet should travel. The *path forwarding problem* is defined as how to specify a packet's path and forward the packet along the specified path.

This thesis presents a novel multipath forwarding method to solve the path forwarding problem by using small, fixed-length path identifiers (IDs). This method guarantees path forwarding by chaining path ID agreements along paths [3, 85]. To do this with minimal computation and message overhead, the method requires that the calculated path set satisfy a specific property, called the *suffix matched* property. For path sets that do not satisfy this property, efficient closure operators are developed to convert them into suffix matched sets.

The novel forwarding method uses a fixed-length per packet path ID and requires additional router storage proportional to the number of paths calculated. This is a significant improvement over the traditional source routing technique that requires variable length per packet path IDs, and routing storage overhead proportional to the number of paths calculated and their path length. The proposed forwarding method is a key component for efficient implementations of multipath routing.

Contribution 3: MPTCP, a multipath transport protocol that effectively utilizes available bandwidth on multiple paths.

The performance gains of using multiple paths depend on how effective end-hosts use the paths to increase their performance. The appropriate usage of these paths varies as applications vary. For example, an FTP application needs to use multiple paths to increase throughput, whereas a telnet application needs to use paths to decrease delay. The final contribution of this thesis is the development of a transport protocol, called MPTCP, which uses multiple paths to increase end-to-end throughput.

MPTCP stands for multipath TCP and is derived from the single path Transfer Control Protocol (TCP) [83]. MPTCP provides a reliable bit stream service and relieves applications of the burden of managing multiple paths. MPTCP operates by opening TCP connections on different paths and multiplexing data among them. The receiving MPTCP protocol coalesces data from the different connections to restore the original message stream. MPTCP performs congestion and flow control, both critical for MPTCP's high performance.

Simulation results show that MPTCP is able to effectively use the additional resources offered by multiple paths, even under heavy network utilization levels. This protocol demonstrates that immediate end-to-end performance gains can be obtained from multi-path networks.

1.2 Dissertation Overview

The rest of the dissertation is organized as follows. Chapter 2 presents the routing background necessary for the remainder of the thesis and discusses the differences between the dynamic metric and multipath routing models. Chapter 3 describes the hybrid-Scout routing algorithm, presents simulation results, and specifies the conditions under which hybrid-Scout outperforms traditional dynamic metric algorithms.

Chapters 4 through 9 address issues in the multipath routing model. Chapter 4 provides a formal introduction to multipath routing and presents the cost and benefit tradeoffs of multipath networks. Chapters 5 – 7 develop the components necessary for multipath routing to succeed. Chapter 5 develops appropriate path calculation algorithms that calculate quality paths between nodes. These paths are supported by the multipath forwarding methods given in Chapter 6. Finally, Chapter 7 describes a transport protocol, MPTCP, that effectively uses the multiple paths to maximize throughput.

Chapter 8 describes the implementation of two multipath routing algorithms. These two algorithms are then tested in Chapter 9. Through simulation, Chapter 9 evaluates the entire multipath routing model, from the performance of user protocols to the costs incurred by the two routing algorithms. The results conclude that multipath routing can be implemented efficiently and sufficient benefits obtained.

Finally, Chapter 10 describes the related work, and Chapter 11 summarizes the research contributions of this thesis and discusses future research directions.

Chapter 2

Background

This chapter provides the necessary background for this thesis by describing the basic Internet routing model and its current routing algorithms. The deficiencies in Internet routing motivate the need for dynamic metric and multipath routing in large-scale data networks.

The chapter begins by describing the two predominant routing algorithms, the Distance Vector and Link State routing algorithms, which are the basis of many Internet routing protocols. Moreover, many of the routing algorithms developed in this thesis are based on these two algorithms. Sections 2.1 and 2.2 describe the Distance Vector and Link State algorithms respectively. To place these algorithms in context, a brief description of the Internet is given in Section 2.3.

Section 2.4 then compares the Internet's static metric single path routing model with the dynamic metric single path and static metric multipath routing models. This comparison highlights the advantages and disadvantages of each model and reveals performance critical issues in dynamic metric and multipath routing models.

2.1 The Distance Vector Routing Algorithm

The first routing algorithm described is the Distance Vector routing algorithm (DV), a distributed, adaptive routing algorithm that computes shortest paths between all node pairs. Based on a centralized method known as the Bellman-Ford algorithm [22, 62], DV is the basis of many practical routing algorithms currently in use [7, 111, 133].

Basic Distance Vector Algorithm

A DV router computes a forwarding table which is used to forward packets on the best know path to their destination. Each entry in a DV forwarding table contains three elements: the destination address, the next-hop neighbor on the known shortest path to that destination, and the cost of the known path. Each router's forwarding table is continually updated by the distributed DV algorithm to indicate, via the next-hop entry, the shortest path between nodes. Routers update their forwarding tables by exchanging path information with their neighbors via *distance vector packets* (DVP). A DVP carries the identity of the originating router and a list of distance vectors of the form (*dst_addr*, *cost*), which are taken from the router's forwarding table. An DVP entry (d, x) sent by a router R indicates that R can reach destination d, and the cost of the path is x.

The description of the DV computation uses the following notation. Let C_d^r be the cost of the known least-cost path to destination d in r's forwarding table, N_d^r the next-hop neighbor on the path to d recorded in r's forwarding table, and c_{rs} the cost of the link from r to s. DV assumes that the path cost is additive and the cost of each link is positive. That is, the cost C of a path $(x_1, \dots, x_n)$ is $C = c_{x_1x_2} + c_{x_2x_3} + \dots + c_{x_{n-1}x_n}$ and $c_{x_ix_{i+1}} > 0$, $\forall 1 \leq i < n$. The forwarding table in each router r is initialized as follows:

$$C_r^r = 0;\ \forall s : s \neq r, C_s^r = \infty.$$

When a router r receives a DVP $((d, C_d^s), \dots)$ from a neighbor s, r updates its forwarding table as follows: for all destinations d in s's DVP,

$$if\ (C_d^s + c_{rs} < C_d^r\ or\ N_d^r = s)\ then\ (C_d^r = C_d^s + c_{rs} \text{ and } N_d^r = s).$$

Notice that the cost added to C_d^s is c_{rs}, the cost from r to s. This addition correctly computes the path cost from r to d through s. Router r locally computes c_{rs} by observing its link status to neighbor s.

Routers exchange DVPs with their neighbors periodically or in response to link/node failures or recoveries. It can be proved that after a bounded number of DVP exchanges following a topology or link cost change, all routers' forwarding tables will contain values reflecting the shortest paths to all destinations [22, 157].

After DV forwarding tables converge, routers will then forward packets to their destinations on the calculated shortest paths. Because DV computes all pairs single shortest paths, tagging a packet with its destination address uniquely identifies the packet's path through the network. Data packets in DV are delivered to their destinations as follows: whenever a router receives a packet, it finds the forwarding table entry for the packet's destination and forwards the packet to the next-hop router listed in the table entry. This process of looking up the next-hop address and forwarding a packet to the proper neighbor continues until the packet reaches its destination.

Dynamic Metric Distance Vector

To calculate dynamic metric paths, a DV router continually updates the cost of each of its out-going links. When a router observes that the difference between a link's current cost and its last advertised cost exceeds a certain threshold, the router initiates (or triggers) a Distance Vector route computation. Because link costs may change very frequently, thresholding is used to dampen the number of path recomputations. Another mechanism often used to further dampen the rate of recomputations is to use hold-downs [74]. A hold-down limits the number of route computations a router can trigger within a certain time interval.

Given that the cost change in link l causes router R to initiate a route computation, the steps of recomputation are as follows:

1. In R's forwarding table, for every path p with l as its out-going link, R updates p's cost to reflect l's new link cost.

2. For each path updated, R sends a DVP containing these paths (and their new costs) to all out-going links but l.

3. Upon receiving a DVP update, router P performs the usual path updates. After updating its forwarding table, P sends DVPs containing path p if 1) p is newly acquired (as in the static metric case) or 2) the new cost of p exceeds p's old cost by a threshold.

After the DVP exchanges, the paths calculated between nodes will reflect l's new cost. Furthermore, the amount of dynamic metric computation is reduced by the use of different thresholds in the computation.

2.2 The Link State Routing Algorithm

The Link State (LS) algorithm is another routing algorithm on which many protocols are based. Example of these protocols are found in the Internet and in ATM networks [3, 110, 114].

The Basic Link State Algorithm

In the LS routing algorithm, every router periodically broadcasts (via flooding) its local connectivity. This information is flooded in a Link State packet (LSP) which consists of the router's ID, a list of its neighbors' IDs, and the cost of each connecting link. After all routers broadcast their LSP, every router knows the entire network topology. A router then performs a shortest-path spanning tree computation rooted at itself [54]. From the computation of this tree, the shortest paths to all destinations in the network are known. This information is then encoded in the router's forwarding table. A LS forwarding table is a list of tuples of the form (*dst_addr*, *next-hop*), where *dst_addr* is the address of the destination, and *next-hop* is the neighboring router on the shortest path to that destination.

Since the LS algorithm also computes single shortest paths between nodes, the usage of LS forwarding tables to deliver packets to their destinations is the same as DV.

Dynamic Metric Link State

Like DV, path recomputations in dynamic metric LS are also triggered by link cost changes. Dynamic metric LS routers continually update the link costs of their outgoing links and trigger path recomputation when the difference between the current and previously advertised link costs exceeds a threshold. Again, similar to DV, cost thresholding and hold-downs are used to limit the number LS path computations.

Upon detecting such link cost changes, an LS router broadcasts an LSP that contains the new link cost. When a router receives an LSP, it performs its usual shortest-path spanning tree computation that incorporates the new LSP information. Thus, after every router receives a dynamically triggered LSP and recomputes shortest paths, the paths provided by a dynamic metric LS routing algorithm will reflect the new cost change.

2.3 The Internet

The Internet is a global data network, and as of 1999, it consists of approximately 40 million hosts and 20,000 routers [136, 142]. The Internet is a best effort datagram network: units of data transmission are packetized, and each packet is individually and independently delivered (hop-by-hop) to its destination without any guarantees of success. That is, packets can be dropped in transit. In this section, we describe the current Internet routing architecture and show how LS and DV routing algorithms operate in and scale to large data networks.

In order for a node to send packets to another node, the sending node must specify the destination node. In the Internet today, every node is uniquely identified by an 32 bit IP address*. Thus for node A to send a packet to node B, A must designate the IP address of B as the packet's destination. This universal naming allows every node in the Internet to uniquely identify every other node.

*In the next version of IP, called IPv6, nodes will be identified by an 128 bit address [26].

Physically, the Internet is a collection of smaller inter-networks called *routing domains* (or *autonomous systems*). Each routing domain is responsible for routing packets within itself. That is, packets destined to hosts in a routing domain are routed by the domain's routing algorithm. An intra-domain routing algorithm refers to a routing algorithm that routes packets within a domain, and an inter-domain routing algorithm is responsible for routing packets between domains. Using this organization, packets sent from one domain to another are first routed out of the sending domain, then to the destination domain, and finally to the destination host by the destination domain's routing algorithm. Thus, logical Internet connectivity is ensured by the cooperation of inter- and intra-domain routing algorithms.

Because of the size of the Internet, two methods are used to make the network scalable: subnetting and hierarchical routing domains. Subnetting uses the IP addressing structure so that routers can route packets based on a set of hosts instead of individual hosts, and hierarchical routing domains reduce the size of routing domains. Both of these methods are described below.

Subnetting

Internet addresses are divided into *subnets*. A subnet is identified by some IP address prefix. Whenever an organization wants to connect to the Internet, it needs to obtain a unique subnet address or a set of unique subnet addresses. Furthermore, every host owned by the organization that connects to the Internet needs to have an IP address such that the IP prefix of the host address is the same as one of the subnet addresses obtained by the organization. The number of bits in the IP prefix is variable length [65, 129, 133], and the exact number of bits is not directly relevant to our discussion.

With subnet addressing, routers outside a subnet need to know and maintain only a single route (i.e. state) for all the hosts in the subnet. That is, routers outside a particular subnet do not need to maintain a route for every hosts in that subnet, but rather, these routers only need to maintain one route to the subnet; this route is used to forward packets to every host in that subnet. Because several thousand hosts can have the same subnet address, subnetting allows substantial space savings in routers, making routing over the Internet more scalable.

However, subnetting alone is not enough to provide scalability because there are hundreds of thousands of subnet addresses in the Internet. To further reduce the amount of information that routers have to process and store, hierarchical routing domains are used.

Hierarchical Routing

Hierarchical routing allows a routing domain to contain subnet addresses as well as other routing domains (represented by a set of subnet addresses). Conceptually, routing is hierarchically structured such that at the lowest level, a routing algorithm routes packets to hosts with the same subnet address. At the second level, a routing algorithm routes packets among second-level routing domains using the subnet addresses the routing domains encompass. Similarly, at level i, a routing algorithm routes packets to the appropriate level i routing domain that contains the packet's destination subnet address. Because of the inclusive property of routing domains, a packet is routed to level $i + 1$ if the packet is destined for a subnet address that is not in level i. Eventually, routers in the highest level of the hierarchy, say level L, know about every subnet address and the level L routing domain that contains the address.

Thus a packet destined for a different subnet first travels up the domain hierarchy, until it reaches the routing domain that is the first common ancestor of both the source and destination routing domains. From this ancestor domain, the packet then travels down the hierarchy until it reaches the destination domain. The destination domain's routing algorithm then ensures the packet reaches its destination.

This hierarchical organization achieves scalability because a routing algorithm operating in a level i routing domain only needs to compute paths to nodes at that level. A node at level i could be a host (if $i = 1$) or a level $i - 1$ routing domain. If a node is a routing domain, then the node advertises the subnet addresses that it encompasses. With this hierarchical structure, a routing algorithm at a particular level only has to compute paths to nodes at that level, thereby reducing the size of path recomputations that routers need to perform.

Conceptually, the Internet routing hierarchy can have many levels. However, in practice, the Internet routing is divided into only two levels, intra-domain (lower level) and inter-domain (higher level) routing levels.

Although the size of host addresses is different from domain addresses, a router's basic mechanisms for computing paths and forwarding packets are the same. Thus, the routing algorithms described in this thesis are applicable for all the levels of the routing hierarchy.

More information on Internet routing is available in standard network text books such as [93, 129, 154].

2.4 Comparison of Routing Models

As stated in Chapter 1, dynamic metric single path and static metric multipath routing models can offer higher network performance compared to current Internet routing algorithms that implement the static metric single path routing model. This section compares the three routing models and examines how their differences influence network performance.

Our comparison begins by examining routing algorithms under dynamic environments. This discussion applies to both single path and multipath routing models. Next, Sections 2.4.2 and 2.4.3 compare the performance differences among the three different routing models: Section 2.4.2 presents how network performance is influenced by each model's resource usage, and Section 2.4.3 compares the operation granularity of multipath and dynamic metric routing models. Operation granularity refers to the scale in which a routing algorithm computes its paths.

2.4.1 Static and Dynamic Metrics

A distributed routing algorithm (both single path and multipath) needs time t in order to converge on the computation of its paths. This convergence time depends on factors such as the actual routing algorithm, network topology, link transmission speed, and router computation speed. For example, in the LS algorithm, convergence time depends on the time needed to detect topology changes, to broadcast topology information, and to recompute shortest paths.

When the network state changes, one expects that after time t, a routing algorithm will converge and calculate the appropriate paths based on the new network state. For example, assume that the convergence time for an LS algorithm is 5 seconds. Then 5 seconds after a link failure, the LS algorithm will recompute shortest paths such that the new paths do not use the failed link. In this case, the LS algorithm is correct because it calculates the expected paths (the shortest path between nodes) and does so within 5 seconds. However, the notion of correctness and convergence is less clear in scenarios where the network state changes faster than a routing algorithm's convergence time. This type of network scenario typically occurs in dynamic metric routing because link costs can change much more frequently than a routing algorithm's convergence time.[†]

In the previous example, assume that the network topology (or link costs) changes every second, then the LS algorithm can never catch up to the network changes and can never compute paths that reflect the actual shortest path in the current network. Rather, the paths that the LS algorithm computes will reflect the shortest paths in an out-dated network state, where the degree of out-datedness depends on the algorithm's convergence time. In other words, as the network changes, LS *tracks* the changes and computes paths according to its most up-to-date view of the network, which may be the actual network state sometime in the past. Note that system A tracks system B if whenever B changes to B', A also changes (say to A') such that the difference between the output of A' and B' is less than or equal to the difference between the output of A and B' [31]. Thus, a routing algorithm that tracks network changes will recompute its paths such that the newly computed paths better reflect the current shortest paths, compared to the previously computed paths.

[†]Although frequent network changes can also occur in static metric networks, these are usually networks under special conditions such as highly mobile, wireless networks [87, 127].

In network scenarios where a routing algorithm does not have an up-to-date view of the current network state[‡], one can still reason about the correctness of a routing algorithm. Recall in networks where rates of change are slower than convergence time, a routing algorithm is correct if it computes the expected path within its convergence time. Since a routing algorithm computes paths based on its current view of the network, such algorithms are correct if 1) their view of the network is *consistent* with the actual network and 2) the expected paths are computed based on that view. A router has a consistent view of the network if the router's knowledge of the current network state is the same as the actual state of the physical network.

In situations where a routing algorithm's network view is not consistent with the current network, a correctness criterion must ensure that the routing algorithm does not diverge in its shortest path computation. That is, the paths that a routing algorithm *actually* calculates should track the paths that the routing algorithm *would* calculate given it has the instantaneous network state. This property can be ensured by two criteria: 1) the algorithm's view of the network must track the actual state of the network. That is, the difference in the view of the network at time X and $X + \delta$ should reflect the actual network changes observed in the elapsed time δ. Of course, the amount of network updates during the elapsed time depends on the actual parameters of the routing algorithm (e.g. threshold and hold-down values). 2) Correct paths are calculated based on the algorithm's view of the network state. In the LS example, this means that given LS's view of the network, the shortest paths are calculated between nodes.

Assuming that dynamic metric versions of DV and LS are not guaranteed to have an up-to-date view of the network state, both algorithms are still correct according to the criteria given above. In both algorithms, shortest paths are computed based on each algorithm's

[‡]A routing algorithm might not have an up-to-date network view either because the network state changes too frequently or because the algorithm does not attempt to obtain the exact, moment-to-moment state of the network due to efficiency issues. Mechanisms such as thresholding and hold-downs allow routing algorithms to reduce routing cost at the expense of less accurate network view.

current view of the network, and network views are continuely updated via triggered path recomputations. In LS, the network view is updated by topology broadcasts, and each broadcast partially updates the network view. Similarly in DV, the network view is updated by messages received from neighboring routers that contain path information that use current network link costs.

In practice, this implies that the accuracy with which a dynamic metric routing algorithm tracks physical link costs directly affects the quality of the paths calculated. The closer the tracking, the closer the calculated paths reflect the current state of the network; however the cost of tracking the network state comes at a cost. Chapter 3 addresses this issue and shows that achieving good network performance involves a tradeoff between the cost of tracking the physical network and the quality of the paths calculated.

2.4.2 Resource Usage

This section compares the three routing models in terms of network resource consumption and shows how their differences affect network performance. To understand these issues, one needs to determine 1) what network resources are, 2) how these resources are consumed, and 3) how network performance is defined. These terms are defined below.

First, there are two types of network resources, router and link resources: router resources refer to router CPU cycles and memory, and link resources refer to link bandwidth. Second, these network resources are consumed by two types of messages: data messages and routing messages. Routing messages are sent by routers to compute and provide paths between nodes, and data messages are sent by applications to other applications. Both types of messages consume link resources as they traverse the network and router resources because routers receiving these messages have to process and/or forward them. Third, network performance is determined by application network demands. Examples of different network demands are high throughput, low delay, and high probability of satisfying QoS requests[§]. Although network performance is application specific, in general, performance

[§]Quality of Service (*QoS*) specifies a level of network service, typically expressed as a combination of can be characterized by the speed in which data messages are delivered to their destinations: the faster data messages are delivered from their sources to their destinations, the higher the network performance. We shall use this general measure of network performance for the remainder of the discussion.

Given these definitions, the following subsections analyze how the three routing models trade off between network performance and resource consumption of data and routing messages.

Data Messages

This subsection examines how data messages use network resources as they travel to their destinations. In static metric single path routing, the network resources used by data messages between a node pair are fixed given a particular topology: the paths provided are based on static link costs and do not change with traffic conditions. The drawback of this routing model is that network performance is dependent on traffic; therefore depending on traffic patterns, the static metric shortest path might not provide good network performance.

In dynamic metric routing, the cost of a link considers the traffic on the link. In this model, a link's cost changes dynamically – a link's cost increases if the amount of traffic on the link increases. Thus, in dynamic metric routing, the network resources used between a node pair change (via path recomputation) in order to minimize dynamic path costs and to increase performance. In essence, with dynamic metric routing, data messages travel different paths (i.e. use different network resources) depending on network traffic patterns.

In the multipath routing model, the routing algorithm provides multiple paths between nodes and allows the sending nodes to choose how to send data using the provided paths.

---

delay bounds and capacity requirements [9, 43, 60, 71, 95]. For example, a video conferencing application that transmit real-time video and audio may need paths with 1Mb/s bandwidth and a delay bound of 10ms. In this scenario, the video conferencing application needs to reserve this QoS specification on the path to its destination. Notice that this request may be rejected by the network due to resource availability. However, once reserved, the network is expected to provide the requested path performance to the application.

With respect to network resource usage, the resources a message uses to its destination depend on the paths calculated by the routing algorithm and on the end-host sending the message. The paths that data messages *can* travel depend on the paths calculated by the multipath routing algorithm. However, the path that data packets actual travel depends on the end-host sending the packets. Thus, in multipath routing, end-hosts can increase their performance given that they are able to gauge the performance of their available paths and send data on paths that best maximize their performance.

In summary, both dynamic metric and multipath models offer higher performance than static metric shortest routing by delivering data on paths which consider observed network traffic. In dynamic metric routing, the routing algorithm observes network traffic and recomputes its paths to optimize a given performance metric. The data messages then automatically traverse the recomputed paths. With multipath routing, the multipath routing algorithm simply computes multiple paths between nodes. It is the responsibility of the end-hosts to observe the performance of the computed paths and send data on paths that best increases their performance. This difference in data resource usage affect the two models' cost-performance tradeoff. The tradeoff difference is discussed further in subsequent sections.

Routing Messages

Routing messages are the other type of message that consume network resources. This section considers the tradeoff between their resource consumption and network performance.

With respect to this tradeoff, the static metric single shortest path routing is at one end of the spectrum that minimizes routing resources. This model uses the least amount of resources among the three models because it computes only one path between a node pair, and these paths do not change unless the network topology changes. In contrast, dynamic metric and multipath models use more routing costs in the attempt to provide higher network performance.

In the dynamic metric routing model, routing messages are used to track the network state (e.g. link costs). Thus, if an algorithm uses more routing messages, it has a more accurate view of the physical network, resulting in higher path quality because the computed least-cost paths more closely reflect the actual least-cost paths in the physical network. This higher path quality, in turn, increases performance because better paths increase the speed that data messages reach their destinations (given that link metrics reflect packet transmission time). However, because routing messages also consume network resources, which affects network performance, dynamic metric algorithms must balance between the accuracy of their network state information and the potential performance degradation of obtaining the information.

Multipath routing algorithms also incur extra routing overhead in exchange for potentially higher network performance. Extra overhead is incurred through calculating and maintaining multiple paths between nodes. Thus, multipath routing algorithms must balance between the resources needed to compute and maintain paths versus their potential performance gains.

To summarize, compared to static metric single shortest path routing, both dynamic metric and multipath routing require more routing message resources. These extra resources allow both models to obtain higher network performance. However, there is a fundamental difference between dynamic metric and multipath cost-performance tradeoff. In dynamic metric routing, the current least-cost paths between nodes are determined by the routing algorithm. Therefore, changing paths between two nodes requires routing intervention, which entails routing costs. In multipath routing, however, the cost of calculating multiple paths is incurred *once* for a particular network topology, and subsequent path changes to avoid congestion are done by end-hosts and thus do not require routing intervention or routing overhead. For this reason, the multipath routing model is inherently more efficient than dynamic metric routing at making the routing cost versus network performance tradeoff. We address this issue in more detail next.

2.4.3 Granularity

As shown in the previous section, both multipath and dynamic metric routing increase network performance by using network resources that consider network traffic. However, the difference between the two models is that end-hosts in multipath routing choose which paths to use, whereas nodes in dynamic metric routing do not. This section explains how this difference serves to distinguish the two routing models.

At a very abstract level, one can imagine that a dynamic metric routing algorithm can "simulate" a multipath routing algorithm by allowing nodes to specify the desired path before sending a packet and then compute the path fast enough that the routing algorithm provides an illusion that there are multiple paths between nodes. This section shows that this cannot be realized in practice. The reason points to the fundamental difference between the two models – the granularity in which the two models operate. The difference can be categorized into *time* and *path* granularities.

Time granularity refers to the time scale in which nodes can send data on different paths. In a multipath model, nodes have the freedom to send data on any of their available paths. For example, a node can send consecutive messages on different paths in a round-robin fashion. That is, the time granularity of path switching is on the message level (or on the order of microseconds). However, in dynamic metric routing, switching paths between nodes requires global path recomputation which is on a much larger time granularity, typically on the order of seconds. The reason for the larger time granularity is due to efficiency: computing paths on the microsecond granularity would consume a prohibitively high amount of network resources, which can significantly reduce network performance. Notice that in multipath routing, switching between paths does not incur any routing overhead.

Path granularity refers to the scale of switching paths. In a multipath setting, when a node decides to send data on different paths, the decision is made by the sending node alone and affects only that node's performance. That is, path switching is on an end-to-end granularity. In dynamic metric routing, however, path recalculation is done on a network granularity because path recomputation entails global calculation. This computation involves potentially every router in the network and potentially affects paths of other connections. Because of this global granularity, it is thus infeasible to allow nodes in a dynamic metric environment to arbitrarily switch paths and request global path computations. Notice that the end-to-end path granularity in the multipath model naturally allows nodes to freely switch paths.

Because multipath and dynamic metric models offer different operating granularities, in practice, the two models are very different and require different routing algorithms to implement each model.

2.4.4 Routing Model Summary

In summary, both dynamic metric single path and static metric multipath models trade off resource consumption for increased network performance by delivering data on paths that consider the dynamics of network traffic. In the dynamic metric shortest path model, resources are used to more accurately track the network state, resulting in better calculated paths. Multipath routing, on the other hand, uses additional resources to compute and maintain multiple paths, thereby allowing end-hosts to achieve higher network performance.

Analysis of this cost–performance tradeoff reveals that multipath routing is intrinsically more efficient than dynamic metric routing. This is derived from the observation that to provide increased end-to-end performance, a multipath routing algorithm needs to compute its paths only once (until the network topology changes), while a dynamic metric algorithm needs to continually recompute paths to maintain path quality.

Although dynamic metric algorithms make this tradeoff less efficiently, an advantage of this model is that it only requires changes in router software and minimal end-host changes. Since dynamic metric routing provides one path between a node pair, current end-host protocols and softwares need little change, if any, to obtain performance benefits. In contrast, multipath routing requires updates to both router and host software in order to reap multipath performance benefits.

As this section shows, the decision of which routing model to implement requires evaluating tradeoffs at various levels. Analysis of such tradeoffs continues throughout the remainder of this thesis. The following chapters carefully consider the cost–performance tradeoffs when developing algorithms that implement dynamic metric and multipath routing models.

Chapter 3

Dynamic Metric Routing

This chapter presents a novel routing algorithm that implements the dynamic metric shortest path routing model. Dynamic metric routing has been implemented on the ARPANET (the precursor of the Internet) and substantial improvements in network performance were observed: Khanna and Zinky [94] showed that dynamic metric routing has a significant impact on the ARPANET's performance, where performance is measured in packet drops. This dynamic cost metric, called the *revised ARPANET routing metric*, also reduced routing instabilities and oscillations compared to the previous dynamic metric based purely on link delays. Despite their positive results, dynamic metrics are not widely used in today's Internet. The technical reasons are that 1) the amount of routing updates with dynamic metrics are hard to control *a priori* because routing updates are dependent on network traffic, and 2) routing using dynamic metrics can cause routing oscillations, though Khanna and Zinky's dynamic metric lessens the problem.

The dynamic metric routing algorithm presented in this chapter to a large extent, overcomes the limitations cited above. The algorithm uses *Scout*, a destination-initiated shortest path computation technique. A destination node using the Scout algorithm initiates a path computation from every node in the network to itself at periodic intervals. The period between route recomputations is controlled by the initiating node. The routing overhead of the algorithm is a function of this period and is independent of the dynamic conditions in the network. Therefore, Scout has the property that its routing overhead is predictable and under the control of the initiating nodes.

Since each initiating node takes charge of its route recomputations, the updating of routes to each initiating node is uncorrelated. In contrast, traditional dynamic metric algorithms based on Link State (LS) and Distance Vector (DV) recompute paths between all nodes quasi-simultaneously. This quasi-simultaneous computation has the unfortunate tendency of shifting *congestion causing traffic* from a currently congested area to a currently uncongested one which then becomes the next congested area, triggering route recomputations over again [163]. Because route recomputations in Scout are independently controlled by the destinations, route updates to different destinations are uncorrelated and usually staggered in time, thus allowing congested traffic to be split, significantly reducing route oscillations caused by shifting all traffic in a congested zone at the same time.

However, this flexibility and autonomy in route recomputation comes at a price. Scout is not as efficient as algorithms such as DV in computing all-pairs shortest paths. This is because Scout destinations independently compute paths to themselves and do not share path computations with other nodes; thus, using Scout to compute dynamic metric paths to all nodes is inefficient.

To obtain the benefits of dynamic metric Scout while maintaining low routing costs, Scout is integrated with the traditional LS and DV algorithms. In this hybrid algorithm, the LS or DV component operates as usual and calculates static metric shortest paths. This provides good baseline paths between nodes. The Scout component then computes dynamic metric paths to a small, selected number of "hot" destinations. Analysis of Internet packet traces show that a high percentage of Internet traffic is destined to a small percentage of "hot" destination networks (see Figure 3.1). The main idea behind the hybrid is that if a small number of destinations receive the majority of the traffic, then making sure that their traffic do not experience congestion (by calculating dynamic metric paths to these destinations) will reduce the probability that the remaining traffic experiences congestion.

By using Scout with dynamic metrics to improve path quality to hot destinations and relying on traditional static metric algorithms to maintain paths to other nodes, the hybrid algorithm reaps the benefits of dynamic metric routing without paying a high, unpredictable routing overhead and without incurring significant routing instabilities. Moreover, since static metric DV and LS algorithms efficiently calculate all-pairs shortest paths and Scout

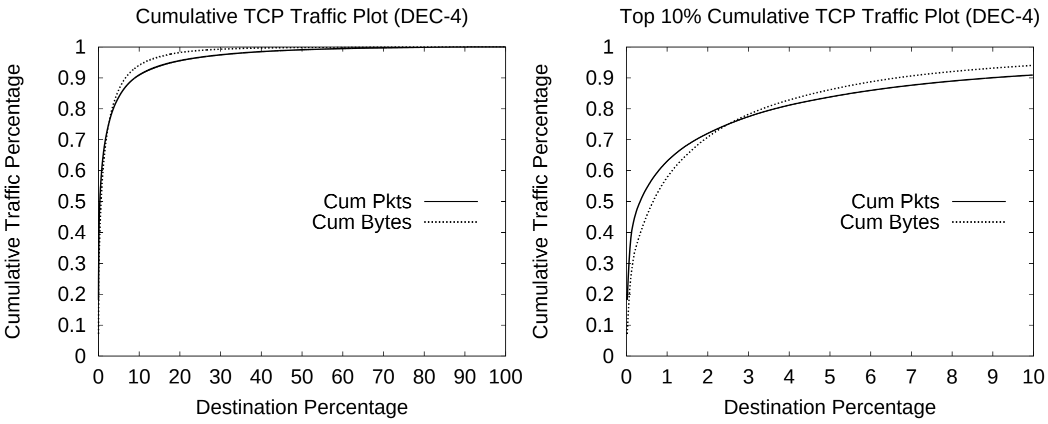


Figure 3.1 : Traffic locality in dec-pkt-4 TCP packet traces from *http://ita.ee.lbl.gov/html/contrib/*. The graph shows the cumulative distribution of network traffic to destinations. The trace contained an hour's worth of wide-area traffic from DEC's corporate intra-net to the rest of the Internet. The x-axis shows the destination percentage sorted by traffic frequency, and the y-axis shows the cumulative distribution. The left graph shows the cumulative distribution of traffic to all destinations, and the right graph shows distribution to the top 10% of destinations that receive traffic. The distributions of traffic to sources exhibit the same distributions. The locality shown in this trace is typical of other traces analyzed.

efficiently calculates paths selectively, the hybrid algorithm combines the strength of the different algorithms.

The rest of this chapter is organized as follows. Section 3.1 presents the hybrid approach. The section first describes Scout, a selective route recomputation algorithm that takes advantage of network traffic locality. Scout description is followed by two sections that describe Scout integration with DV and LS routing algorithms. Section 3.2 presents experimental data showing the behavior of the different dynamic metric algorithms. The results of the simulation show that hybrid Scout outperforms traditional dynamic metric algorithms using significantly lower routing costs. Finally, Section 3.3 concludes this chapter.

3.1 The Hybrid Approach

One of the key ideas of the hybrid approach is that routes based on dynamic metrics are computed only for a minority of destinations, namely destinations that receive a majority of the traffic. Due to the high destination locality observed in network traffic, being able to calculate dynamic metric paths to these *hot destinations* has several advantages. First, one hopes that calculating paths to a few destinations is cheaper than calculating paths to all destinations. Second, not calculating shortest paths to all destinations simultaneously reduces the route heredity problem [94] and therefore reduces route oscillation. In addition, because these destinations receive a high percentage of routing traffic (say 50+%), rerouting these paths enables the algorithm to shift enough traffic to alleviate areas of congestion.

The solution presented here for dynamic metric routing is based on two key concepts. The first is the Scout routing algorithm that is able to selectively update paths to individual destinations. The second key idea is to integrate Scout with traditional DV and LS algorithms. This hybrid provides base static routes to a majority of destinations using DV/LS and provides dynamic metric routes to selected destinations using Scout.

Scout is a host-initiated, selective route calculation algorithm that calculates least-cost paths to individual nodes (or networks). Host-initiated means that the path calculation process to a node (or network) is initiated by that node (or gateway router to the network). Scout is selective because only paths to that destination are affected, and it is efficient at computing routes to individual destinations. However, Scout does not aggregate path computation, and therefore is not as efficient as LS and DV when computing paths to all nodes. This deficiency motivated the integration of Scout with LS and DV.

The hybrid of Scout with DV and LS routing algorithms uses the traditional routing algorithm to calculate paths between nodes using static metrics, while Scout continually refines paths to selected nodes using dynamic metrics.

The remainder of this section is dedicated to describing the hybrid Scout algorithm. The next section presents a brief description of the basic Scout algorithm and shows how it computes least-cost paths between nodes. The integration of Scout with LS and DV follows in Sections 3.1.3 and 3.1.4. Finally, Section 3.1.5 summarizes the hybrid-Scout LS and DV algorithms.

3.1.1 The Scout Algorithm

This subsection briefly describes the Scout routing algorithm. Appendix A contains a full description of Scout, including a proof of its correctness and convergence. The Scout algorithm is described in a static metric network environment; Section 3.1.2 discusses the algorithmic implications of using dynamic metrics.

In the Scout routing algorithm, each router maintains a forwarding table indicating for each destination, the next-hop neighbor on the known least-cost path to that destination. The forwarding tables are updated by Scout and are used in the same way as in LS and DV.

Scout's basic mechanism of route computation is message *flooding**. With Scout, destinations periodically flood a *Scout* message throughout the network. Let R be the node initiating the Scout message. The period between two consecutive floodings of Scout messages from R is called the *broadcast interval* (BI). A Scout $[R,\ C_R,\ x]$ contains the originating node's address, R, and the cost C_R to reach R, initially zero, and an increasing sequence number x. When a node P receives a Scout message $[R,\ C_R,\ x]$ from its neighbor Q, P first checks whether the sequence number x is valid. If not, the Scout is discarded. Otherwise, P modifies the Scout's cost C_R to include the cost of sending a message from P to Q, $C'_R = C_R + Cost(P \rightarrow Q)$. C'_R represents the cost a message will take if it is sent by P via Q to R.

In the first broadcast interval (i.e. when P has no record of receiving a Scout from R), P forwards the Scout $[R, C'_R,\ x]$ immediately after receiving the first Scout message,

*Flooding is a standard method of broadcasting information in a point-to-point network. The flooding method operates as follows: whenever a node receives a flood message, it remembers the message and sends the message to all neighbors except the neighbor that it received the message from. If a node receives a flood message it has seen, the message is discarded. Thus, a flood message terminates after every node receives the message.

initiated by R, to all neighbors except Q, the neighbor which sent P the Scout. Node P might receive more of R's Scouts in the same BI, indicating different paths and path costs to R. P remembers only the least-cost Scout to R and adjusts its forwarding table to reflect the best known path, but P does not forward these Scout messages. For each node, the next-hop to R is always the neighbor that provided the least-cost Scout in the current BI. In the next and subsequent broadcast intervals of R, P waits to receive a Scout message from its *designated neighbor* before flooding. We define node P's *designated neighbor* to node R as the neighbor of P that provided the least-cost Scout to R in the previous broadcast interval. When P receives the Scout from this designated neighbor, P forwards the least-cost Scout received in the current BI (i.e. with the current sequence number) to R to all neighbors except the one from which the least-cost Scout was received.

In the presence of node or link failures, a node P might never receive a Scout from its designated neighbor because the designated neighbor may no longer be connected. This is taken into account by requiring P to flood the first Scout message from R if P did not flood any of R's Scout messages in the previous BI. In other words, if P waited for a Scout from its designated neighbor Q in the previous BI but never received a Scout, instead of waiting for Q, or any other neighbor in the current round, P immediately floods the first Scout it sees in the current broadcast interval.

In this case, P is not allowed to wait for a neighbor in the current BI, in particular, the designated neighbor, because if there are multiple failures, waiting for the best information might cause cascading waits. Our decision was motivated by the observation that propagating more recent, perhaps sub-optimal, information is more useful than trying to wait for the best information which entails the risk of not propagating any information at all.

As an example, consider node A sending Scouts in its first broadcast interval (sequence number 0). A trace of how Scouts propagate through the network is shown in Figure 3.2. Node A sends a Scout of the form $[A, 0, 0]$, denoting the Scout originates from A, with cost 0, sequence number 0.

In this example, node A's Scout forwarded by B reaches D before the one forwarded by

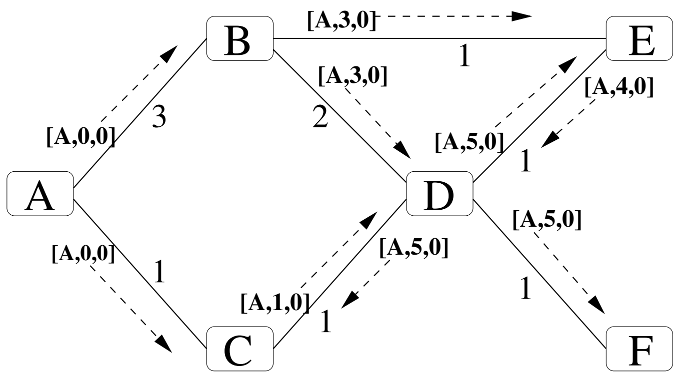


Figure 3.2 : A Scout Example. Node A Sends Scouts for the first time. In this network, boxes denote routers and edges links. The number beside each link represents the cost of that link.

C. Since D has no previous information of A's Scout, D sends the Scout $[A, 5, 0]$ to all its neighbors, after adjusting the Scout cost. Notice that D learns the optimal path to A when it receives the Scout $[A, 1, 0]$ from C. However, D does not propagate this information in the current BI. Instead, C becomes D's designated neighbor (and next hop to A). In the first BI, the only node that did not get the optimal cost to A is node F, even though it has the optimal route. That is, because of hop-by-hop forwarding, F's packets destined to A will traverse the path (F, D, C, A) with cost 3 even though F thinks it's optimal path cost is 6.

In the next BI, D will wait for C's Scout before sending its Scouts. The trace is given in Figure 3.3. Notice that D sends the Scout from C. The algorithm converges in two rounds and every node has the least-cost path to A.

There are five distinctive features of the Scout routing algorithm. First, the number of Scouts forwarded in one BI is $O(L)$, where L is the number of links†. Second, in a steady state network, the worst-case number of BI's needed for Scout to converge is proportional to the longest calculated cost path (with respect to the originating node). However, this worst case requires pathological conditions. In practice, Scout's convergence is much better.

†It is not exactly L because it may happen that two neighboring nodes send Scouts to each other simultaneously. The number of Scouts is between L and $2L$.

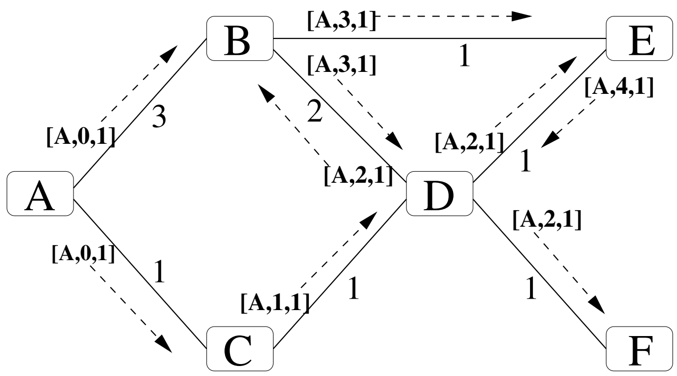


Figure 3.3 : A Scout Example. A sends Scout in the second BI

Simulations were conducted on networks with more than 100 nodes which showed that Scout always converged in no more than three BI's. Moreover, after a link failure, an alternate (not necessarily least-cost) path is guaranteed to be computed after at most two BI's, if one exists.

Third, Scouts are small and fixed size and can therefore be piggybacked on a hop-by-hop basis onto data packets, largely defraying their cost to the network. Fourth, Scout's route updates are completely controlled by the Scout generating node and not by network changes; therefore Scout routing costs are easily controlled. Fifth, Scouts require very little router computation. This is in contrast to LS and DV where a routing update may affect entire forwarding tables.

3.1.2 Dynamic Metric Scout

The Scout routing algorithm can also compute paths using dynamic metrics. That is, Scout can calculate paths to destinations when link costs change dynamically. The implementation of dynamic metric Scout is straightforward. Routers maintain each out-going link's dynamic cost as in dynamic metric LS and DV, and upon receiving a Scout from link l, the router adds l's current dynamic cost to the Scout cost and proceeds according to the usual Scout algorithm.

Given that the number of BI's for Scout to converge is proportional the longest calculated path, it may be that the rate of link cost changes in a dynamic metric network is higher than the Scout convergence rate. In this case, Scout might never converge to the actual dynamic metric shortest path to a particular destination. As stated in Section 2.4.1, the correctness of dynamic metric Scout in this scenario depends on 1) whether Scout's view of the network tracks the physical network and 2) whether paths are correctly calculated according to Scout's current view.

Similar to DV, a Scout router updates its view of the network by receiving Scout messages from its neighbors. Scout messages update the network view because whenever a router forwards a Scout, the most up-to-date link costs are added to the Scout. Thus, the network view to a destination is tracked as the destination sends Scouts on every BI.

With respect to the second condition, the dynamic metric Scout algorithm also calculates the appropriate path given its current network view: a router always stores current BI's least-cost Scout in its forwarding table, and routers always forward the current BI's least-cost Scout as dictated by the Scout algorithm. Because Scout is an iterative algorithm, Scout might not calculate the shortest path to a destination after one Scout broadcast. Therefore, although a Scout broadcast from destination d does not guarantee that every router after the broadcast will have the shortest path to d, the calculation process is still correct because Scout update and forwarding procedures guarantee reductions in path costs and produce shortest paths in a steady state network.

Like dynamic metric LS and DV, the dynamic metric Scout algorithm trades off between routing costs and the accuracy that the calculated paths approximate actual least-cost paths. For example, in dynamic metric LS and DV, the smaller the threshold and hold-downs, the more closely the calculated path track actual least-cost paths. Smaller thresholds and hold-down values allow more frequent routing updates; thus the routing algorithm's view of the network more closely tracks the actual network. Similarly, the smaller the Scout BI's, the better Scout paths track least-cost paths because smaller elapsed time between consecutive BI's reduces the difference between Scout's view of link costs and the actual costs.

The cost and performance of a dynamic metric routing algorithm can be measured by how much routing resources (in terms of routing messages, router CPU, etc.) the algorithm uses in order to provide a certain level of path quality. In our experiments, the cost of a link is a function of the link's queue length, which directly reflects link delay. Using this link metric, path quality is measured by the time a packet reaches its destination. Therefore, an algorithm that provides a lower overall network packet delay provides better performance (or tracks network changes more closely) than algorithms that provide higher delays. An ideal dynamic metric routing algorithm would provide minimum overall network packet delay. Comparisons of routing cost and performance (in terms of packet delay) for hybrid Scout and dynamic metric LS/DV algorithms are given in Section 3.2.

3.1.3 Scout-DV Hybrid

The Scout-DV hybrid algorithm combines the Distance Vector (DV) routing algorithm with Scout. This section describes the integration of the two algorithms and justifies the behavior of the resulting hybrid routing algorithm.

Scout-DV Algorithm

Scout-DV algorithm has a DV component and a Scout component. The DV component is responsible for computing the least-cost paths using static link costs and propagates link failures and recoveries in the usual way. The Scout algorithm is integrated with only a few small changes described below. The Scout component directly modifies the DV forwarding tables (the next-hop and cost fields) and uses dynamic link metrics.

Because the DV component of the hybrid algorithm computes static shortest paths between all destinations, the Scout component of the algorithm can take advantage of this. The only modification to Scout is the following:

- If the Scout algorithm does not have any information on the designated neighbor, the default is to use the next-hop neighbor computed by the DV algorithm as the designated neighbor.

This feature is used in two places: 1) when a node generates its first Scout message and 2) during the first Scout broadcast after a link failure/recovery (see below).

The two modifications to the DV algorithm are:

- When a router detects a link/node recovery, it exchanges forwarding tables with neighboring routers in the usual DV manner, propagating the Scout paths as if they were computed by DV. The receiving router performs the standard operations on the received DVP and recomputes the shortest paths for both Scout and non-Scout sending destinations.

- When a router detects a link/node failure, it follows the standard DV procedure and sends to all its neighbors a DVP containing infinity cost for all destinations that use the failed link (including Scout sending destinations). This process nullifies all DV and Scout paths that use the failed link. Again, the standard DV recomputation mechanism will compute static metric shortest paths (if they exist) to all affected destinations.

  To ensure that a straggling Scout message from the previous BI does not re-advertise a path through the failed link, the Scout-DV hybrid algorithm increments the Scout sequence counter for Scout calculating destinations that were affected by the failure. Notice this does not alter the Scout algorithm behavior in the next BI because the sequence number will then be current.

The above modifications to Scout and DV algorithms serve to better Scout's approximation of the least-cost path. The Scout modification provides an educated guess for a default designated neighbor: lacking any dynamic metric path information, the static metric shortest path is the most likely the shortest dynamic metric path. The DV modifications, on the other hand, ensure that the Scout paths affected by a component failure/recovery become *valid* in at least the same speed as affected DV paths. A *valid path* is defined as a path that connects the source of the path to the destination. In a distributive routing environment, this means that the forwarding table of every intermediate router on a valid path has, as its next-hop field to the path's destination, the next router on the path and that every link/router along the path is operational. In other words, packets are forwarded to their destinations only along valid paths.

Notice in the hybrid algorithm, the speed of convergence for paths to non-selected destinations are the same as those in traditional DV algorithms (static metric DV). This is because only the DV component of the hybrid algorithm calculates paths to non-selected destinations. Therefore, paths to these destinations are updated by the DV component in exactly the same manner as ones updated by traditional DV algorithms; thus paths to non-selected destinations in hybrid-Scout must have the same convergence guarantees as destinations in traditional DV algorithms.

The next section shows that the modifications to the DV component in the presence of component failures/recoveries provide connectivity to Scout generating nodes.

Scout-DV Interactions

In the hybrid Scout-DV algorithm, the DV and Scout components interact in order to speed Scout's least-cost path computation and to speed the establishment of valid paths after link/node failures. The purpose of this section is to show that the DV-Scout interactions described above result in valid paths. Notice that the hybrid algorithm is still correct even if the two components are completely decoupled: both DV and Scout are capable of calculating shortest paths without help from the other.

To show that valid paths are computed as a result of Scout-DV interactions, a distinction is made between destinations that generate Scouts and destinations that do not. For those that do not, their paths are updated only by the DV component, and therefore the validity of their paths is the same as in the pure DV algorithm. The correctness of DV has been proven in [22, 62].

With respect to destinations updated by Scout, their paths are also updated by the DV component. Only for these destinations do we need to verify that valid paths are computed. Because the Scout and DV components interact only during certain events, we need to ensure that after those events (or after the interactions), valid paths are provided by the hybrid algorithm. The list below enumerates the points of Scout-DV interactions and the conditions that ensure the calculation of valid paths.

1. When Scout does not have a designated neighbor, Scout's usage of DV's next-hop neighbor as the default designated neighbor does not cause Scout to diverge in its path computation.

2. After a link failure, the DV component nullifies all Scout calculated paths that traverse the failed link. Moreover, DV calculates alternate paths and these paths are valid.

3. After link recovery, the DV component calculates valid paths.

Justification of 1: In the pure Scout algorithm, if a router receives a Scout which it does not have any previous information, the router will treat the neighbor that sent the first Scout as its designated neighbor. With the DV optimization, the hybrid Scout will use the DV's next-hop neighbor as its designated neighbor. Notice that in the subsequent BI, the Scout component will use the designated neighbor it computed from its previous BI. Thus this optimization can only affect Scout behavior for the first BI; therefore the optimization does not affect Scout correctness (i.e. its approximation of the least-cost path) in calculating shortest paths to Scout generating nodes.

Justification of 2 and 3: In DV, whenever a router detects that a link has failed, it nullifies forwarding table entries to destinations whose next-hop is through the failed link. In addition, the router sends a DVP to all its neighbors indicating that it can no longer reach the list of nullified destinations. The sending of DVP's then initiates a network wide path cancellation and recalculation process.

The complication with Scout-DV is that the DV component nullifies Scout calculated paths and later recomputes these paths. The concern is whether this interaction causes the calculation of invalid paths to Scout destinations.

To show that valid paths are computed as a result of interactions 2 and 3, we must show that after the DV recalculation process, the collective Scout-DV forwarding tables reflect, for every Scout destination affected by the topology change, a sequence of valid next-hops that leads to the destination.

The crux of this justification is that Scout paths have exactly the same properties as DV paths. Therefore, when Scout paths are nullified and recomputed by the DV component, these paths will have the same resulting properties as paths calculated by DV. Thus the validity of Scout paths that are (re)calculated by the DV component is reduced to the validity of DV paths. Given that the correctness of DV has already been established, reducing Scout paths to DV paths shows that interactions 2 and 3 will produce valid paths to Scout generating nodes.

The justification of interactions 2 and 3 proceeds as follows: first, we state the necessary and sufficient conditions of DV paths. Second, we show that Scout paths have the same properties as DV paths. The validity of DV-modified Scout paths then directly follows.

From the Bellman equations [22], the three necessary and sufficient conditions of a path $(x_0, \ldots, x_n)$ calculated by DV are the following:

1. Node x_i's forwarding table entry has x_{i+1} as its next-hop to x_n, $0 \leq i < n$.

2. The cost x_{i+1} has to x_n is less than x_i's cost to x_n, $0 \leq i < n$.

3. The information of this path flowed from x_n to x_0 via the reverse path $(x_n, \ldots, x_0)$.

The Scout calculated paths also have these three properties: 1) if x_i calculates the least-cost path to x_n as $(x_i, \ldots, x_n)$, then x_i's forwarding table entry for x_n has x_{i+1} as its next-hop. This is from the basic Scout update rule. 2) Scout's costs are additive, satisfying the second property. 3) Like DV, Scout's path information flows from the destination in the reverse direction of the calculated path. This follows from the Scout flooding mechanism.

Because Scout's paths look exactly like DV computed paths, whenever the DV component updates router forwarding tables, it cannot distinguish the paths calculated by Scout from the ones calculated by the DV component itself. Thus Scout calculated paths will be altered in the same way as DV calculated paths and will have the same DV path properties after link/node failure or recovery. That is, the DV component is guaranteed to nullify all invalid Scout paths after a link failure and recompute valid paths to Scout destinations on link recoveries. QED.

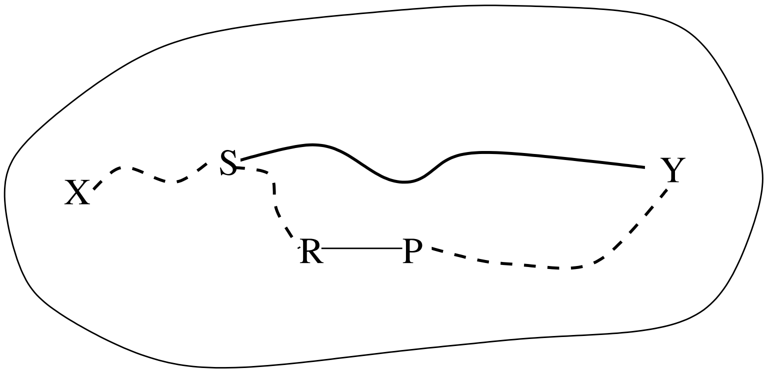


Figure 3.4 : An example of a path that the DV component calculates to a Scout generating destination after the failure of link (R, P).

Figure 3.4 shows an example of a Scout path invalidated by a link failure and subsequently recomputed by the DV component. In this figure, the dotted curve between nodes X and Y denotes the path node Y has to node X. Here, X is a Scout generating node. For this example, assume the link between R and P fails and that the DV component detects the failure before X generates another Scout[‡]. According to the hybrid algorithm, the DV component will use static metrics to recompute the failed paths to X. Notice that DV will invalidate all Scout paths that used link (P, R) but leave the valid paths untouched. In particular, the dotted curve from R to X remains valid while the curve from Y to P is invalid due to the link failure.

In this example, the path that the DV component calculates from Y to X extends a valid Scout path. The newly calculated path is denoted by the solid line from Y to S and

[‡]If X broadcasts Scouts before DV's link failure detection and that the Scouts calculate valid paths to X, then DV will not nullify the newly calculated Scout path because the path would appear to be a valid DV path.

the dotted line from S to X. Furthermore, because DV calculates static metric shortest path, the solid path from S to Y is the shortest, static metric path that connects Y to a valid Scout path to X.

3.1.4 Scout-LS Hybrid

This section describes the Scout-LS hybrid algorithm. LS, like DV, is a widely used and well understood routing algorithm. However, unlike DV, it computes paths in a centralized manner: a topology broadcast mechanism ensures that each router knows the current state of the entire network (i.e. topology and link cost), and paths are calculated by each router knowing the entire network topology. This centralized path calculation method posses slight problems to the hybrid algorithm when component failures occur. The basic hybrid algorithm is described below, and solutions to address component failures follow.

Scout-LS Algorithm

The Scout-LS hybrid, like the Scout-DV hybrid, requires very little modification to either algorithm. The LS component maintains its forwarding table, computes paths using the static link metrics, and recomputes routes on detection of link/node failures/recoveries in the usual manner to non-Scout generating destinations; LS processing of Scout paths is discussed below. For Scout generating destinations, Scout messages directly modify LS forwarding tables and compute dynamic metric paths to their initiating destinations. Like the Scout-DV, Scout is modified such that a destination's default designated neighbor is LS's next-hop neighbor to that destination. As proven earlier, this modification does not affect the validity of Scout paths.

The following subsections describes the LS integration with Scout. The main area of concern is the handling of Scout paths when the LS component detects network component failures. This problem and its solution are described below.

Component Failures

LS routers exchange topology information to compute shortest paths. Whenever a network failure occurs, the LS routers that detect the failure will broadcast this failure information, which then triggers a network wide path recomputation. Because Scout paths do not carry information as to the nodes/links they traverse, this means that whenever a Scout-LS router receives a topology broadcast indicating a node/link failure, it cannot determine whether a Scout path traverses the failed node/link. Thus if the LS component does not correct Scout paths invalidated by the failure, these paths will remain invalid in the number of Scout BI's needed to correct these paths. To speed the validation of Scout calculated paths, a mechanism is needed to correct invalid Scout paths. As in Scout-DV interactions, this Scout-LS interaction is an optimization because the Scout component will correct its paths in the following BI's.

Unlike failures, component recoveries do not pose similar problems because Scout paths affected by a recovery are still valid; therefore packets sent on these paths will reach their destinations. Thus, ensuring that Scout paths are aware of component recoveries is not as critical as in component failures. Because it is less critical and that relating Scout paths to LS paths is difficult, the Scout-LS's LS component does not alter Scout paths after node/link recoveries. Recall in Scout-DV, the DV component does "optimize" Scout paths after component recovery. This optimize was performed because the DV and Scout path computation mechanisms are very similar, allowing a natural implementation of this optimization.

For Scout-LS, the different possible approaches to cope with component failures are given below.

1. Do not update Scout paths. This means a Scout path could potentially be invalid for two Scout BI's. This is a reasonable solution in networks where the time to detect link/node failures are on the order of the Scout BI's.

2. Update all Scout paths to static metric LS shortest paths. This approach replaces

all Scout paths whenever LS recomputes paths due to a component failure. The disadvantage of this method is that it changes all Scout paths, even those that do not traverse the failed component(s).

3. Implement a DV style invalidation procedure. This approach implements a DV-like procedure to nullify Scout paths after component failures. In this method, when a failure is detected, Scout-LS routers that detect the failure will initiate the procedure to nullify all Scout paths invalidated by the failure. This DV-like procedure is proven to nullify all invalid Scout paths (Section 3.1.3). Notice that the procedure does not need to recompute the nullified Scout paths because the LS component already knows the static metric shortest paths to all destinations.

4. Attach source routes to Scouts. In this approach, Scouts maintain a source route of the path they advertise. Thus, whenever a component failure occurs, routers simply check if the link/node that failed is traversed by a Scout path; if so, the path is updated to the LS calculated static metric shortest path. However, the disadvantage of this approach is that it significantly increases the size of Scouts.

The four approaches listed above address link/node failures in different ways. The first approach simply allows the Scout mechanisms to recalculate alternate paths. The advantage of this approach is its simplicity and that it requires no changes to the Scout-LS algorithm. The principal disadvantage is that the paths to selected destinations may be invalid for 2 BI's. This is undesirable because traffic to selected destinations are assumed to be important, and therefore connectivity (i.e. valid paths) to these destinations should be established as soon as possible.

The second approach indiscriminately invalidates all Scout calculated paths. This approach is simple to implement and immediately establishes connectivity to the selected destinations. The drawback is that the invalidation causes a majority of Scout paths to unnecessarily revert to the static shortest path (assuming a majority of the Scout paths are not affected by the failure). This sudden change in routing behavior may have a negative performance effect if the number of Scout paths are high [84].

The final two approaches accurately invalidate only the affected paths. The first approach uses the Scout-DV invalidation mechanism to delete Scout paths. This approach has the disadvantage that it requires significant additions to the Scout-LS algorithm. The fourth approach simplifies the path invalidation procedure but complicates the Scout component. This approach attaches a source route to every Scout. With a source route, determining whether a path traverses a link is trivial; however, the disadvantage is that Scout messages are no longer fixed size, significantly decreasing their efficiency.

The solution adopted in this thesis is to modify the fourth approach such that Scout paths are selectively and conservatively deleted while maintaining a fixed Scout size. The solution is as follows: a 64-bit field is added to hybrid-LS Scout (initially zeros) and each router is required to identify itself with a 64-bit ID. These router IDs do not have to be unique. Every time a router forwards a Scout, the Scout's 64-bit field is bit-ORed with the router ID. A router stores a Scout's 64-bit field if it uses the path advertised by the Scout. Whenever a link or node fails, the attached routers broadcast the failure along with their 64-bit ID. Upon receiving this LS broadcast, routers recompute their paths as usual, and for each Scout path, the path is invalidated if the broadcasting router's ID is contained in the 64-bit Scout field. This containment test is a simple bit-AND operation.

For example, consider a Scout passing through three routers with the following 8 bit IDs: `00110000`, `01001000`, `00010001`. The Scout's bit field will be `01111001`, the bit-OR of all the three routers. If a link attached to the second router fails, the router will broadcast an LSP that indicates the failure and that contains the router ID, `01001000`. Routers using the this Scout path will invalidate this path because the ID in this LSP is contained in the Scout's bit field. Using this method, all Scout paths that traverse a failed component is guaranteed to be detected. However, if a router with ID `01100000`, which is NOT on the Scout path, broadcasts a component failure, then this Scout path will also be invalidated. These false invalidations are called false positives.

Because this method is conservative, it may delete Scout paths that do not traverse the failed component, as the previous example shows. Simulations show that if a router selects its ID by randomly marking two bits in the 64 bit field and if Scout paths are 15 hops long on average, then the number of false positives is around 15%. With the low false positive rate, this approach provides a good tradeoff between Scout efficiency and invalidation accuracy.

3.1.5 Algorithm Comparisons

One of the main differences between hybrid-Scout (i.e. Scout-DV or Scout-LS) and dynamic metric LS and DV algorithms is that the efficiency and effectiveness of Scout depends on the number of Scout generating destinations, Scout generation rate, and how much traffic those destinations receive. In contrast, with dynamic metric LS and DV, routing costs depend on the degree of congestion experienced in the network.

As stated earlier, in order for hybrid-Scout to effectively reroute congestion, the amount of traffic received by the Scout generating nodes must be "significant". In addition, in order for Scout to be efficient, the number of hot destinations must be "small". We experimentally quantify these two conditions in Section 3.2 and show that these conditions are typical of real networks.

In summary, the main features of the hybrid-Scout algorithms are:

1. Hybrid-Scout can selectively upgrade paths to a destination to use dynamic metrics. This is done by simply having the destination generate Scout messages.

2. Hybrid-Scout reduces route oscillation when compared to LS/DV with dynamic metrics. This is accomplished by computing dynamic metric paths to only selected destinations in a time staggered manner.

3. In hybrid-Scout, the desired route quality to a destination can be determined independently on a destination by destination basis. That is, hybrid-Scout can easily provide different path qualities to different destinations, depending on their importance and traffic usage. This adjustment is done solely on the part of the Scout generating node by changing its Scout BI.

4. Hybrid-Scout's routing traffic is independent of network traffic. Therefore, Scout routing traffic can be easily controlled by limiting the number of Scout generating nodes and their BI's. LS and DV routing costs must be bounded using trigger thresholds and hold-downs.

For this thesis, we assume that the set of "hot" destinations is statically known and configured to send Scouts. In the Internet, for instance, ISPs and popular Web sites are likely candidates. Should static configuration prove too inflexible, traffic monitoring can be used to maintain the set of hot destinations dynamically (see Chapter 11.2).

3.2 Simulation Results

This section presents extensive simulation results to evaluate the proposed hybrid Scout algorithms and to compare their behavior with dynamic metric DV and LS routing algorithms. The purpose of these experiments is to determine whether the hybrid-Scout algorithms deliver better overall network performance at lower costs. While network topology is a key parameter determining the performance of all four algorithms, a single class of topologies were chosen to study scaling effects: viz., Internet-like topologies. These topologies are characterized by a highly connected backbone network, to which multiple tree-like subnetworks are attached. The decision was made not only to simplify the experimental analysis but also to make the results relevant to present-day networks.

Since hybrid-Scout recalculates routes based on dynamic metrics only to a set of "hot" destinations (Scout generating nodes), a natural question that arises is how the routing cost varies as the number of hot destinations changes? And as the skew in the traffic distribution to these destinations is increased, is the improvement in route quality worth the extra routing cost? Answers to these questions are obtained by studying the performance of the considered algorithms under varying traffic distributions; in particular the number of hot destinations and the percentage of traffic directed to them. A related question is whether the improvement in route quality to hot destinations comes at the expense of those to non-Scout generating destinations. This section presents an experiment which addresses this question by measuring non-hot destination network performance. For simplicity, traffic destined for Scout generating nodes (hot destinations) is called *foreground traffic* and traffic destined for other nodes *background traffic*.

The experimental analysis consists of studying the effects of (1) network size, (2) foreground traffic distribution, and (3) background traffic quantity on routing cost and overall route quality delivered by the different algorithms. To evaluate the algorithms according to these three parameters, different algorithms were tested on an Internet like topology to show the different dynamic behaviors of each algorithm and their relative performance and cost. Subsequently, background traffic is introduced to examine the behavior of these algorithms with varying traffic distributions.

3.2.1 Simulation Environment

The simulation environment is based on the "x-netsim" package from the University of Arizona [28], an execution-driven, packet-level network simulator. The simulator takes as input a description of the network topology, including link characteristics such as bandwidth and propagation delay, and a set of software modules that implement the various protocols running on the routers and hosts of the network. Simulation time advances according to the calculated transmission and propagation delay of packets in the network. Software processing in the routers and hosts are assumed to have zero cost.

The four routing protocols are implemented in the simulation testbed: the two hybrid Scout routing algorithms, a Distance Vector protocol similar to the Internet RIP protocol [74][§] and a Link State protocol similar to the Internet OSPF [41] routing protocol. In these experiments, only results for the hybrid DV-Scout algorithm are presented because the behavior of the two hybrid algorithms using dynamic metrics are exactly the same. The only difference is that the hybrid LS-Scout messages are 8 bytes larger than those used in hybrid DV-Scout.

[§]Our implementation includes the "split horizon" and "poison reverse" heuristics.

Dynamic metric LS and DV were implemented using trigger thresholds. As a router continually updates its out-going link's costs, it observes whether the difference between any of its link's current cost and the link's cost that was last advertised exceeds the threshold. If so, the router initiates a network wide shortest path recomputation. Trigger thresholds are expressed as *trigger percentages*. A trigger percentage of 50% means that a link's cost has to increase/decrease at least 50% of its dynamic range in order to trigger a routing update. The lower the trigger percentage, the more sensitive the algorithm is to link cost changes, the higher the path quality, but also the higher the routing cost (traffic). Refer to Sections 2.1 and 2.2 for more details on the dynamic metric DV and LS algorithm.

The LS and DV protocols are optimized such that route changes are propagated only in response to a triggered update. In the hybrid-Scout algorithm, the DV/LS component only calculates routes based on static metrics (i.e. it does not trigger on dynamic link cost changes) and only the Scout component computes paths using dynamic metrics. Because static metric paths are calculated once at the beginning of a simulation, the cost of the initial static route calculation are removed for all algorithms. Thus, routing costs presented are the direct result of changes in dynamic metrics.

The performance of the different routing algorithms is measured by average packet delay. Packet delay is the elapsed time between the sending of the packet and its reception. Each experiment is constructed such that packet drops do not occur; therefore, the average packet delay reflects the amount of queuing that packets experience.

Not only is packet delay a good end-to-end measure of path quality, it also reflects the precision in which a routing algorithm tracks network link costs. Given that link costs are based on queue lengths (described below) and queue lengths directly affect packet delay, an algorithm that minimizes a packet's delay minimizes the cost of the path that the packet traveled. Thus, the degree in which an algorithm tracks the network link cost changes is reflected in the observed average packet delay.

The queue size for each router's outgoing link is 100 packets long. A router samples the length of its queues every 10ms. A link's current average queue length is calculated from the average of its instantaneous queue length with the previous running average. The dynamic metric cost function is linear from the static cost (average queue length of 0%) to 3 times the static cost (average queue length of 100%). All links have the same static link cost, and all three algorithms use the same static link cost, dynamic link-cost calculation mechanism, and link-cost function.

3.2.2 A Forest Topology

The purpose of the first experiment is to show the basic behavior and scalability of each algorithm. To make the experiment more relevant to today's networks, the DV, LS, and hybrid Scout-DV algorithms were simulated on the topology shown in Figure 3.5. The picture on the left shows a backbone topology consisting of 6 highly connected nodes. Three of the backbone routers are attached to a tree-like topology, shown on the right. The connections are denoted by a dotted triangle attached to a backbone router. The backbone links have three times the capacity of tree links and all links have equal static costs. There are also cross links between an interior router in each tree and the corresponding router in the neighboring tree, which is not shown in the figure. Four hosts are attached to each tree router and one host is attached to each backbone router. The network has a total of 111 hosts. For the purpose of the routing protocols, each host represents a separate destination. Leaf nodes transmit a stream of 10,000 packets to a host attached to a backbone router at 75% link capacity. Backbone routers with attached hosts that receive packets generate Scouts (i.e. they are initiate dynamic metric path computation to themselves).

Routing Behavior

Before examining the simulation results for the complete topology, we use one "tree" of the forest topology to demonstrate the routing behavior of each algorithm. For illustrative purposes, assume that $R1$ is the hot destination.

In this example, as the leaf nodes send their packets to $R1$, the queues for links ($R2$, $R1$) and ($R4$, $R1$) will begin to fill up. As they accumulate, the costs for those links increase.

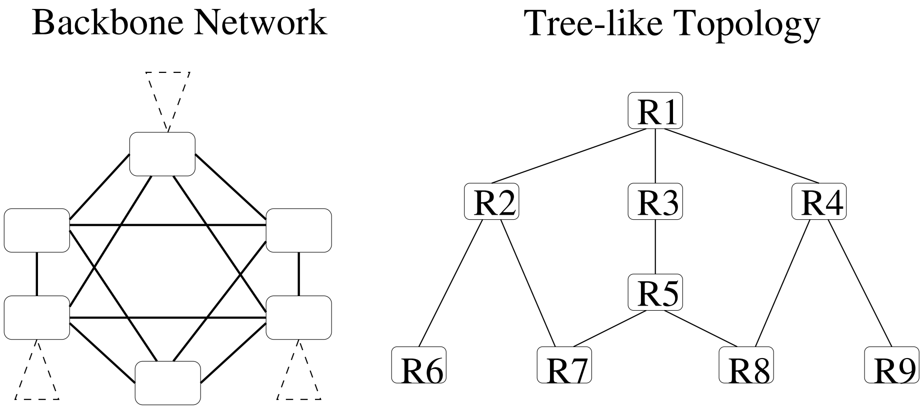


Figure 3.5 : The forest topology used in our experiments. The left-hand figure shows the backbone network, and the right-hand figure shows a tree network. The three backbone routers with a dotted triangle indicates that they have a tree network attached. Boxes represent routers and edges represent links between routers.

Once the cost of those links exceeds twice the link's base value, the shortest path from $R7$ and $R8$ to $R1$ shifts to the alternate path via $R3$, while $R6$ and $R9$'s packets still travel their primary path. After a while, the queue on $R5$ will also start to accumulate, and the traffic will shift back to the primary path. This shifting process continues until the transfers end.

The speed at which this path recalculation occurs depends on the sensitivity and granularity at which each algorithm is calculating the shortest path to $R1$. For LS and DV, this is parameterized by their trigger percentages: the lower the percentage, the faster the algorithm will trigger update messages upon link cost changes, and the faster it responds to congestion. For hybrid-Scout, it is controlled by its broadcast interval BI: the lower the BI, the better the calculated paths reflect the current link costs.

Routing Performance Versus Cost

Simulation results for the topology shown in Figure 3.5 is presented here. In this experiment, the number of hot destinations is varied by having leaf nodes transfer data to varying numbers of hosts attached to backbone routers. The traffic is uniformly distributed among these "hot" destinations. There are potentially two levels where routing based on dynamic metric can increase performance. The first is within each “tree” (as illustrated above), the second is in the backbone network, where multiple alternate paths exist.

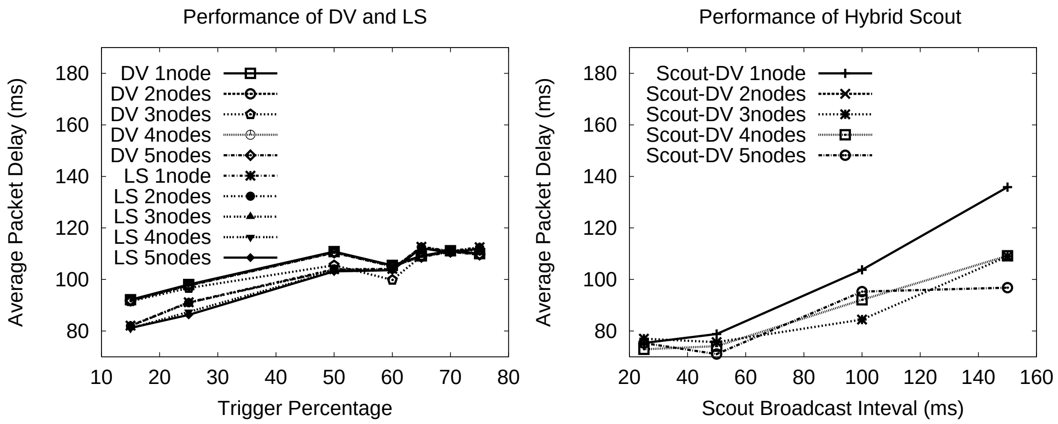


Figure 3.6 : The performance graphs for dynamic metric DV, LS and hybrid Scout algorithms. In the DV/LS graph, the top cluster of curves represent DV performance and the lower LS performance.

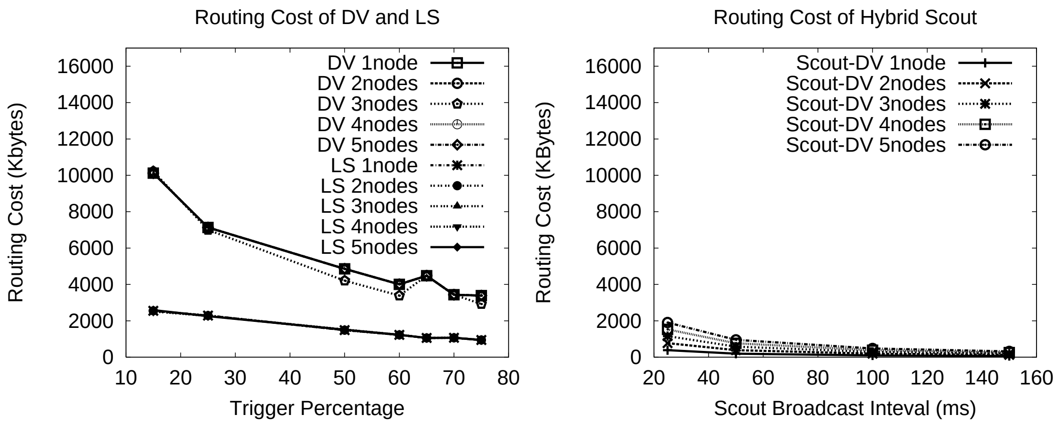


Figure 3.7 : The cost graphs for dynamic metric DV, LS and hybrid Scout algorithms. In the DV/LS graph, the top cluster of curves represent DV costs and the lower LS costs.

The performance and cost results are shown in Figures 3.6 and 3.7. In the performance graphs in Figure 3.6, the left graph shows average packet delay of DV and LS with dynamic metrics as a function of the trigger percentage, and the right graph shows hybrid-Scout’s performance as a function of the broadcast interval. The routing costs incurred by each algorithm are given in Figure 3.7. The label *x*-node denotes the number of distinct destinations to which the leaf nodes are sending packets. The amount of traffic injected into the network for the various number of destinations is fixed; only the distribution of the traffic differs.

The two figures show the effectiveness and efficiency of the hybrid-Scout algorithm on this topology. The performance and cost graphs show that hybrid-Scout achieves performance comparable to LS and DV at much lower routing cost. For example, the performance of LS/DV at 50% trigger percentage is comparable to hybrid-Scout's performance at a BI of 100ms and one hot destination. However, the routing cost of hybrid-Scout compared to LS and DV is approximately 15 and 40 times less, respectively. At five host destinations, hybrid-Scout, DV and LS achieve comparable performance at 100ms, 25%, and 50% trigger percentage; in this scenario, hybrid-Scout uses approximately 3 and 15 times less routing resources than LS and DV, respectively.

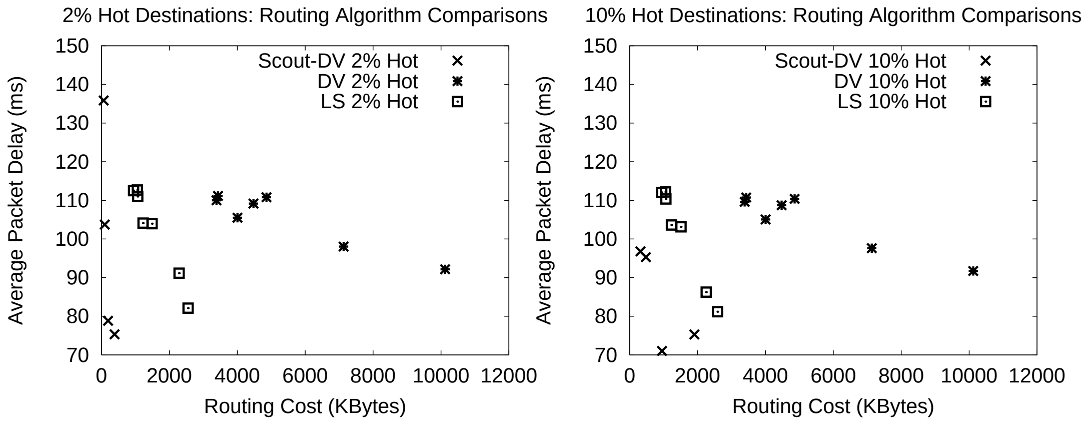


Figure 3.8 : Comparative performance of hybrid Scout, LS and DV for 1 and 5 hot destinations. The curves in these graphs are taken from graphs in Figures 3.6 and 3.7.

To highlight this cost-performance tradeoff, Figure 3.8 shows the routing performance versus routing cost of each algorithm with 1 and with 5 hot destinations. The x-axis denotes the routing cost, and y-axis the performance at that cost. Results are better the closer they are to the origin, indicating good performance at low cost. Compared to LS and DV, hybrid-Scout consistently achieves better performance at lower costs.

As seen from the left graph showing one hot destination, hybrid Scout significantly outperforms both DV and LS on this topology. The graph clearly shows that for the same performance, hybrid Scout uses significantly less network resources than both LS and DV. For example, at a packet delay of 105000us, hybrid Scout requires approximately 100Kbytes of routing traffic while LS and DV require approximately 1224Kbytes and 4004Kbytes, respectively. The right graph confirms that hybrid Scout also outperforms LS and DV with 5 hot destinations.

Routing Cost Versus Network Size

The routing cost for hybrid-Scout and LS algorithms are both $O(L)$, where L is the number of links. The size of each Scout packet is constant (8 bytes for DV-Scout, 16 for LS-Scout) whereas the size of LS packets is proportional to the number of outgoing links per router. In addition, one hybrid-Scout broadcast ($O(L)$) recalculates all the current least-cost paths to a Scout originating node. In LS, however, if there are several points of congestion (say m points), every router at those points of congestion needs to perform a routing flood to recalculate the least-cost path ($O(mL)$).

DV is less scalable than either hybrid-Scout or LS on this type of topology because the size of each routing update is proportional to the number of destinations. The more destinations a network has, the higher the routing cost. With the network used in the experiment, the size of each DV packet is approximately 600bytes, compared to 32bytes for a LS packet.

In addition, since LS and DV compute shortest paths to all destinations simultaneously, they are more likely to cause massive route shifts (many paths simultaneously rerouted to a common set of links). This shifting of many routes may result in congestion, causing additional triggered updates. In hybrid-Scout, the shifting is less severe, because 1) hybrid-Scout does not recalculate dynamic metric paths to all destinations and 2) paths calculations for different destinations are independent and staggered in time.

Table 3.1 experimentally shows each algorithm's scalability in routing cost with network size. For LS and DV, the trigger percentage was 15% and for hybrid Scout, there was 1 hot destination with BI of 25ms. The tree topology is the same as the one in Figure 3.5 and the different numbers of trees are configured as shown in Figure 3.5.

Tree Size	Hybrid-DV-Scout	Distance Vector	Link State	# of hosts	# of links
1-tree	52 Kbytes	35 Kbytes	60 Kbytes	9	10
2-trees	175 Kbytes	1,785 Kbytes	745 Kbytes	68	22
3-trees	385 Kbytes	10,120 Kbytes	2,550 Kbytes	111	45

Table 3.1 : The scalability characteristics of hybrid-Scout, Distance Vector, and Link State routing algorithms.

As the table shows, the scaling characteristics of routing cost for hybrid Scout and LS are relatively linear with the network size. LS is higher because the number of congestive points also increases with the increase in network size. The number of hosts attached to each router increased from 1 to 4 between the 1-tree and 2-trees topology. Thus the routing cost for DV increased significantly between the two topologies. This table confirms the above cost analysis.

Sensitivity to Traffic Distribution

For the hybrid Scout algorithm, the number of Scout generating destinations and the amount of traffic destined to them have a large impact on the algorithm's performance. To examine hybrid Scout's sensitivity to these two parameters, this experiment varies the number of hot destinations from 1 to 5 (approx. 1% to 5% of hosts) and the percentage of traffic destined for these destinations.

With respect to performance, Figure 3.6 shows that hybrid-Scout actually performs better when there are multiple "hot" destinations. This is because Scout calculates least-cost paths to different hot destinations at uncorrelated times, which allows hybrid-Scout to split traffic: paths to destinations that would have shared the same set of links if calculated simultaneously may not, because later route calculations take into account the effect of earlier route changes on network load.

The amount of traffic splitting that hybrid-Scout can achieve depends on the difference in path calculation times. If paths to two neighboring destinations are calculated simultaneously (and thus based on the same link costs), the likelihood that their paths will have common links is high. On the other hand, if their paths are calculated at different times and some link costs change in the mean time due to previous route changes, then their paths will have less links in common. This fact explains why hybrid-Scout's traffic splitting is more prominent at higher BI's. At higher BI's, the mean time between calculations to different nodes are higher; therefore link costs are more likely to reflect the effects of previous route changes. As a result, congestion causing traffic is more likely to be split and network performance is improved.

Hybrid-Scout's routing cost is proportional to the number of Scout generating destinations. As seen in Figure 3.7, the routing cost for five Scout generating nodes (approx. 5% of the destinations) is exactly five times the cost with one Scout generating node.

The performance and cost of LS and DV are largely insensitive to the number of hot destinations[¶], since these algorithms calculate shortest paths to all destinations. The cost of updating paths to one hot destination is the same as updating paths to all destinations. This property manifests itself as coinciding lines in the LS/DV cost graph in Figure 3.6. LS and DV are able to split traffic to a lesser degree because they compute all pairs shortest paths simultaneously, as indicated by the relatively close performance curves in Figure 3.6.

To see the cost-performance ratios of each algorithm, the right graph in Figure 3.8 shows the relative ratios for 5 hot destinations. Again, one sees that hybrid Scout is able to provide better performance at lower costs than LS and DV, and that LS is more efficient than DV.

[¶]The performance and cost vary slightly because the traffic and congestion pattern change with the number of destinations.

3.2.3 Background Traffic

The purpose of this next experiment is to test 1) whether the benefits of hybrid-Scout observed in the previous section are maintained in the presence of background traffic, and 2) whether the performance of background traffic suffers with the hybrid-Scout algorithm.

The same network topology is used as in the previous section. In this experiment, the number of hot destinations is fixed at three (i.e. roughly 3% of the destinations are sending Scouts), and the foreground traffic is fixed at 50% tree-link capacity. At this rate, the foreground traffic by itself does not cause any network congestion; any congestion and subsequent rerouting in the network is caused by the additional background traffic.

To add background traffic, each host in the network repeatedly chooses a node at random and sends 100 packets to that node at 25% tree-link capacity. The amount of background traffic is controlled through the frequency of these transmissions. Increasing the background traffic increases the total amount of traffic injected into the network. Background traffic percentage is defined as the ratio between the amount of background traffic versus the total network traffic.

The experiment uses a trigger percentage of 15% for DV and LS and a BI of 25ms for hybrid-Scout. These parameters were chosen based on their comparable performance. The average packet delay for the foreground and background traffic are shown in Figure 3.9.

The left graph in Figure 3.9 shows the packet delay experienced by foreground traffic. As the overall network traffic increases (marked by the increase in background traffic), packet delay also increases. The results also show that hybrid-Scout's ability to reroute and split traffic allows it to achieve lower packet delay than LS and DV. We conclude that the performance benefits achieved by hybrid-Scout are maintained in the presence of background traffic.

The right graph shows packet delay experienced by the background traffic in the same simulations. The packet delay for the background traffic is less than the foreground traffic delay because most of the traffic do not encounter congested links. The fact that the background traffic performance of hybrid-Scout is comparable to DV and LS shows that the

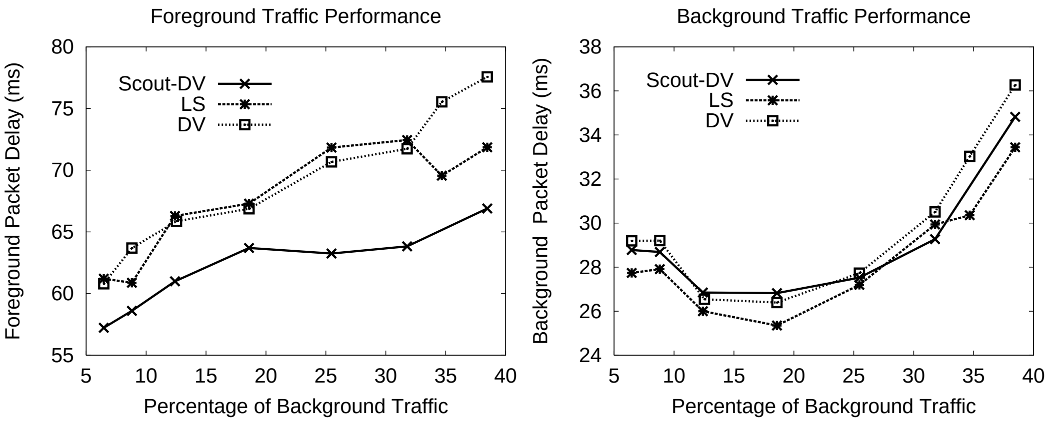


Figure 3.9 : The foreground and background performance in the large network. The percentage of background traffic is the percentage of traffic in the network that is destined for non-hot destinations.

increase in route performance for selected destinations obtained by hybrid-Scout does not come at the expense of non-selected destinations. The intuition is that by shifting traffic that contributes most to congestion (foreground traffic), the remaining traffic on those congested links are not likely to continue causing congestion. Hence, the network performance increases for non-selected destinations as well. The benefits obtained by hybrid-Scout for foreground traffic also indirectly benefit background traffic, as long as there is "enough" foreground traffic. The question of what is enough traffic is addressed in the next experiment.

An interesting feature in the background traffic delay graph is the dip near 12% background traffic. At less than 12%, the background traffic that traverse congested points is not enough to cause much rerouting; thus, packets experience queuing. Above 12%, enough congestion accumulates to cause rerouting, hence the packet delay is decreased because congestion is relieved. Of course, the packet delay again increases with background traffic, because the benefits of rerouting are offset by the increase in traffic. The dip at 12% is not observed in the foreground graph because the congestion that causes rerouting degrades the foreground performance such that there is a net increase in delay as a result of rerouting. The background traffic, on the other hand, always encounters congestion; therefore it can only stand to benefit from rerouting.

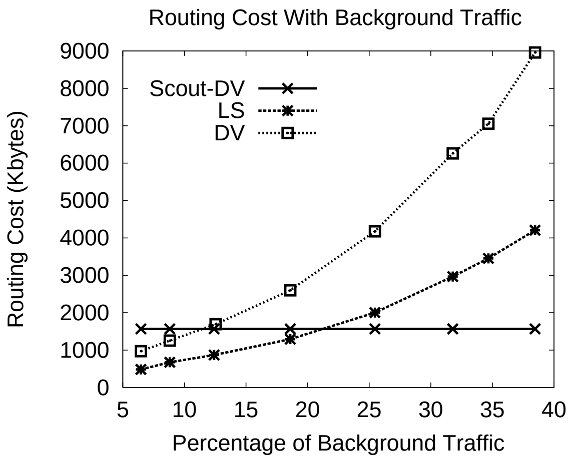


Figure 3.10 : Routing costs for the experiment in Figure 3.9.

Figure 3.10 shows the cost of each routing algorithm in this experiment. The hybrid-Scout algorithm exhibits a constant routing cost with different levels of network traffic. LS and DV's costs increase with the amount of traffic, with DV increasing most rapidly.

The cost graph shows an important characteristic of the LS and DV triggering mechanism: as the network utilization increases (marked by the increase in background traffic percentage), the LS and DV triggered updates also increase. This is undesirable because these updates compete with data packets for link bandwidth and router CPU at a time when links are already heavily utilized and routers are busy forwarding packets. This additional network load increases the probability of severe congestion and packet loss.

The cost graph also shows that LS and DV use less routing resources than hybrid-Scout at low network utilization. Here, only a few links are getting congested, resulting in few triggered updates. However, the advantages of this behavior under low network utilization are not as important as the disadvantages under high utilization levels. Since most links and routers are underutilized at low network utilization, the presence of additional routing traffic does not have a significant adverse impact on network performance.

3.2.4 Foreground Traffic

One of the premises of the hybrid-Scout algorithm is that the amount of traffic received by hot destinations must be “significant enough” for hybrid-Scout to be able to effectively reroute congestion, and that the number of selected hot destination must be “low enough” for hybrid-Scout to be efficient. The following experiment quantifies these conditions.

In this simulation, the same experimental setup as in the previous experiment (3 hot destinations) are used. However, the total amount of traffic in the network is kept constant while varying the distribution of foreground and background traffic. The performance and cost are shown in Figure 3.11. Foreground traffic percentage is the ratio of foreground traffic to total network traffic.

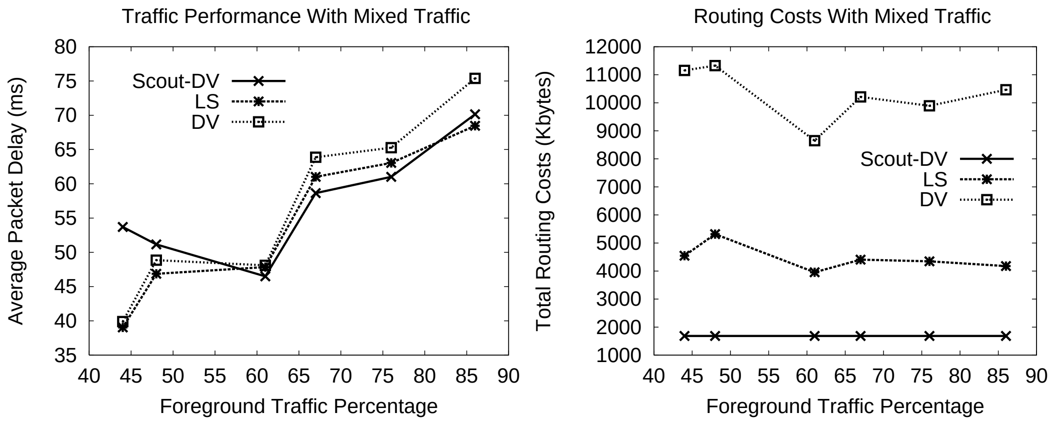


Figure 3.11 : Network performance and routing cost. The amount of traffic injected is held constant while foreground and background traffic percentages vary.

The performance graph (left graph) in Figure 3.11 shows the average packet delay of both the foreground and background traffic. As the foreground traffic percentage increases, more traffic is directed to fewer destinations, causing more packet queuing and higher average packet delay for all destinations. The performance graph also shows that hybrid-Scout is able to achieve performance comparable to LS and DV whenever the foreground traffic accounts for greater than 50% of the network traffic. Notice that hybrid-Scout’s performance is significantly worse than LS and DV at 45% foreground traffic. This is because

Hybrid-Scout can only reroute paths to hot destinations; therefore it cannot eliminate congestion when these destinations only account for a minority of the traffic.

The cost graph (right graph) in this experiment shows that in the simulated network, where 3% of the hot destinations are transmitting Scouts, the routing costs of hybrid-Scout are around 3 to 4 times less than LS and an order of magnitude less than DV.

The routing costs for LS and DV actually increase at 45% and 50% foreground traffic. The reason is that the background traffic is causing minor congestion at many points in the network (as opposed to mainly on paths to the hot destinations when foreground traffic dominates), thus triggering more LS and DV updates.

With the results of this experiment, the two questions posed earlier in this section can be answered:

1. What fraction of total traffic do hot destinations have to receive in order for hybrid-Scout to adequately reroute congestion?

   From our experiments, the answer is at least 50%. The intuition is that if the dynamic routing algorithm is able to control 50% of the traffic, the likelihood that the remaining 50% continues to cause serious congestion is low.

2. How many destinations can generate Scouts for the hybrid-Scout algorithm to be efficient?

   In our topology, if 10% of the nodes generate Scouts (in a highly utilized network), the cost of hybrid-Scout is comparable to LS. That is, hybrid-Scout is cost effective as long as the number of "hot" destinations is below 10% of the total number of nodes in the network. Note that hybrid-Scout's efficiency also depends on the network utilization. As shown in the previous experiment, at low utilization LS/DV tend to be more efficient, and hybrid-Scout is better at higher utilization levels. However, efficiency at low utilization is not as critical as the efficiency at high utilization levels.

Recall that in our study of Internet traffic locality, 1% of the destinations account for over 50% of the network traffic. This indicates that in the Internet, having the hottest 1% of the destinations generate Scouts will be as effective as LS or DV with dynamic metrics in increasing network performance. However, the cost of hybrid-Scout under these conditions should be approximately an order of magnitude less than LS and 2 orders of magnitude less than DV.

3.2.5 Path Approximations

As stated in Chapter 2, dynamic metric routing algorithms may not converge in scenarios where rate of link metric change is faster than the algorithm's convergence time. In these scenarios, an algorithm is correct if it tracks the network state and computes the intended paths over the algorithm's current network view of the network. As shown in Section 3.1.2 and Chapter 2, dynamic metric DV, LS, and Scout algorithms are correct in this sense.

The Scout algorithm (and thus the hybrid-Scout algorithm) converges to least-cost paths in at most l broadcast intervals after link metric changes, where l is the number of hops on the longest calculated path. This is a loose upper bound: in our simulations, the Scout always converges in at most three BI's. Thus, in practice, if the frequency of link cost changes is less than 2–3 BI's, then the dynamic metric Scout algorithm will converge to the least-cost paths. Unfortunately, link cost changes in most dynamic metric networks are much more frequent than Scout BI's.

In comparison to dynamic metric LS and DV, the converge of hybrid-Scout may take longer for sufficiently large BI's. However, it is important to note that unlike Scout, LS and DV must use trigger thresholds and hold-downs to limit the frequency of metric changes and maintain stability. Therefore, even if LS and DV technically converge while Scout does not in a given scenario, the calculated paths are, in both cases, approximations of the actual shortest paths because the cost metrics used in LS/DV calculations are not the actual instantaneous link costs. In addition, note that the metrics used in our simulations to determine routing cost and performance take into account effects of route accuracy and stability.

To see hold-down's effect on route quality, Figure 3.12 compares the cost and performance of LS and DV using hold-downs. This experiment uses the same experimental configuration that produced Figures 3.9 and 3.10. Here, the hold-downs for DV and LS are set to 10-12 and 4-5 triggers per second respectively, and the hybrid Scout parameters are unchanged.

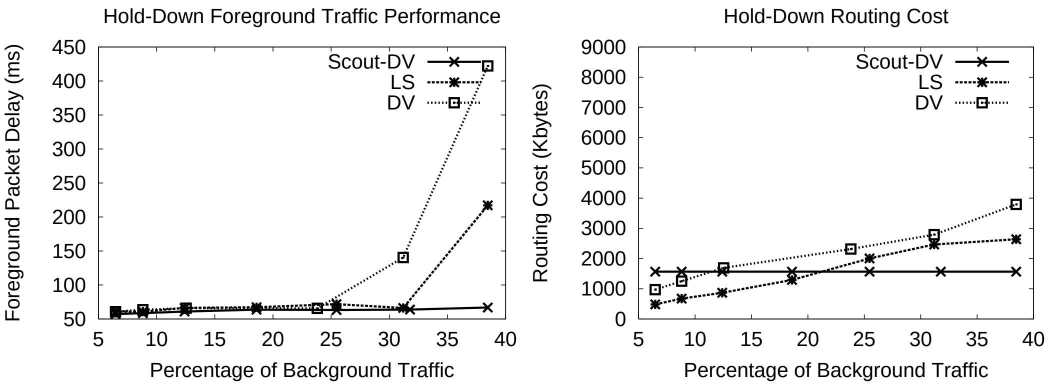


Figure 3.12 : Routing performance and cost using hold-downs. The graphs represent the performance and cost of LS/DV algorithms with hold-downs compared to hybrid-Scout. The experiments used the same configuration as ones used in Figures 3.9 and 3.10.

The right-hand graph in Figure 3.12 shows the cost incurred by each algorithm. Notice that, compared to Figure 3.10, the routing cost of LS and DV are much lower at high network utilization. This is because the hold-downs prevent excessive routing triggers. However, the reduction in routing cost of using hold-downs comes at the expense of routing performance. The left-hand graph shows that the performance of both LS and DV is dramatically worse using hold-downs. This confirms the argument that by delaying path computation, hold-downs force LS and DV to more coarsely approximate dynamic metric least-cost paths (i.e. more coarsely track the network state), thereby degrading network performance.

3.3 Hybrid-Algorithm Summary

Dynamic metric routing has been shown to increase network performance in real networks. However, they are currently not in wide use due to the dangers of routing instability and high routing costs. This chapter presents a new approach to routing using dynamic metrics that promises to overcomes the above limitations. The approach is based on the observation that real network traffic exhibit a high degree of destination locality. Analysis of Internet traffic traces shows that 1% of the hot destinations receive over 50% of the network traffic.

The proposed algorithm, hybrid-Scout, is able to calculate paths based on dynamic link metrics to selected destinations, while paths to other destinations are calculated by traditional routing algorithms using static link costs. Extensive simulations were performed to determine the effectiveness and efficiency of hybrid-Scout in rerouting congestion compared to dynamic metric LS and DV algorithms, and under what conditions. Through simulations, these questions are answered using an Internet-like topology consisting of over 100 host networks and 30 routers. In summary, our simulations show that:

1. Hybrid-Scout is effective at rerouting congestion if at least 50% of the network traffic is destined to hot destinations (i.e. hybrid-Scout generating destinations). Hybrid-Scout is more efficient than both dynamic metric LS and DV if no more than 10% of the network nodes are generating Scouts.

2. Hybrid-Scout is more scalable than LS and DV with dynamic metrics. While achieving comparable network performance (measured in packet delay), hybrid-Scout has substantially less routing cost (in routing message bytes), from 4-5 times to 1-2 orders of magnitude.

3. The selective update mechanisms of hybrid-Scout better splits congestion-causing traffic, which reduces route oscillations. This splitting is achieved by 1) only rerouting selected destinations based on dynamic link metrics and 2) by calculating new routes for those selected destinations in a time staggered manner.

4. Hybrid-Scout's routing costs are stable even under high network utilization levels. This ensures that hybrid-Scout does not exacerbate network load during high network utilization. This is in contrast to dynamic metric LS and DV algorithms that tend to increase routing traffic at high network utilization levels.

Chapter 4

Multipath Routing

The second main contribution of this thesis is the development of a complete static metric multipath routing model: from calculating multiple paths between nodes to end-host methods that utilize multiple paths to increase performance. Like dynamic metric single path routing, multipath routing offers potential performance increase over single path routing by better utilizing network resources. This introductory chapter presents the multipath routing model* and describes the various components needed for its implementation.

The chapter begins with multipath routing definitions to facilitate later discussions. Section 4.2 uses these definitions to discuss the advantages and disadvantages of multipath routing. Finally, Section 4.3 summarizes the necessary multipath routing components for increased network performance. These components are individually addressed in following chapters.

4.1 Multipath Routing Definitions

A glossary of terms to describe multipath routing models is presented in this section. The terms are structured to reflect the key parameters that define multipath networks and influence their performance. These parameters include

1. Basic definitions. The basic terms that describe a multipath routing model.

2. Path specification and calculation. *Path specification* describes the properties of the paths to be calculated between nodes. Example specifications are to calculate node

*For convenience, the term "multipath routing" is used to denote static metric multipath routing, unless otherwise specified.

disjoint paths, shortest K paths, and maximum flow paths between nodes. *Path calculation* is the actual algorithm that calculates the specified paths.

3. Multipath types. This parameter describes the paths a multipath routing algorithm provides between nodes. The two multipath types are *multi-service* paths where the routing algorithm provides different paths with different characteristics (example characteristics are high throughput and low delay) and *multi-option* paths where an algorithm provides multiple paths with the same characteristic. A routing algorithm that provides either or both multipath types is considered multipath routing algorithm.

4. Usage layer. The software layer responsible for using multiple paths to a given destination. This layer manages multiple paths by dictating which data packet should be sent on which path and when. Example layers in today's Internet protocol stack are the network, transport, and application layers.

5. Multipath usage. The way an end-host (or the usage layer of the end-host) uses multiple paths to transmit data.

Basic Definitions

The basic definitions of multipath routing are given here. First, a *multipath routing model* is defined as a routing model where the routing algorithms provide potentially multiple paths between node pairs and allows the end-hosts (or applications) to choose how to use these paths. We require that end-hosts have control over which path to use because this control offers the most flexibility in using multiple paths. This flexibility allows applications to use multiple paths in ways to best maximize their performance.

Using this definition, dynamic metric, single path routing algorithms do not implement the multipath routing model; although these algorithms may route packets between a node pair on different paths, end-hosts do not control the path a particular packet will travel. For the same reason, networks with backup paths, such as telephone networks [150], do not implement the multipath routing model.

A *multipath routing algorithm* refers to a routing algorithm that provides multiple paths between nodes so that data sent on a path travels that path through the network. A *path set* refers to the set of paths that a routing algorithm calculates for a particular network topology, and *multipath networks* are networks with routers that execute a multipath routing algorithm. That is, multipath networks are networks that offer multiple paths between node pairs.

Path Specification and Calculation

In order to calculate paths between nodes, one must first specify the characteristics of the paths to calculate. The different path characteristics depend on the intended use of the paths. For example, paths intended to maximize end-to-end throughput should be specified such that, for any node pair, the aggregate throughput obtain on multiple paths is maximized. In contrast, paths intended to minimize transmission delay should be specified such that, at any given time, there exist at least one low delay path between node pairs. *Path specification* specifies the characteristics a particular path set.

A *path calculation algorithm* is the algorithm that actually calculates the paths specified by path specification. A practical path calculation algorithm takes into account the operating resources and environmental constraints such as the distributed nature of a network. Examples of path calculation algorithms are Dijkstra's shortest path algorithm [54], Topkis's initial link disjoint paths algorithm [156], and the Bellman-Ford distributed shortest path algorithm [22, 35].

Multipath Types

Path type specifies the relationship among the paths a routing algorithm provides between node pairs. There are two path types: *multi-service* and *multi-option*. The first path type, multi-service paths, denotes paths between nodes that have different characteristics (i.e.

different path specifications). Example services that a network could provide are low delay and high bandwidth path services. Since applications may have different demands on the network, providing paths with different characteristics allows applications to choose paths that best fit their communication demands.

The second path type, multi-option paths, denotes the scenario where a routing algorithm provides multiple paths with the same path service. For example, an algorithm might provide four multi-option paths for the high bandwidth path service. That is, each end-host has, to each destination, four paths that can provide high bandwidth to a destination.

Networks that support multi-service and/or multi-option paths are called multipath networks. For example, a multipath network can be one that provides multiple service paths with only one path in each service (multi-service, single option), or one that provides only one path service with many paths within that service (single service, multi-option).

This thesis considers the general multipath routing model where networks provide multiple services each with multiple paths.

Usage Layer

Usage layer refers to the highest protocol layer responsible for managing multiple paths. The levels applicable in today's Internet protocol stack are the network, transport, and user/application layers. If the usage layer is the network layer, then it is the responsibility of this layer to decide which path a packet should travel and to do this transparently to the protocol layers above. Similarly, if the multipath usage layer is the transport layer, this means the transport layer has the freedom to send data on multiple paths.

The protocol layer that manages multiple paths needs to effectively use these paths to increase performance. The choice of usage layer depends on the tradeoffs between flexibility, performance, and the software engineering issues of implementing multipath management at a particular protocol layer. Chapter 7 discusses usage layers in more detail.

Multipath Usage

Usage mode characterizes how multiple paths are used. There are two prototypical multipath usage modes: using paths concurrently or one at a time. The choice of which usage mode is application specific. For example, for applications interested in maximizing throughput, such as FTP, the right usage mode is to use all paths concurrently to obtain the aggregate bandwidth of all available paths. On the other hand, the appropriate usage mode for delay-sensitive applications, such as Telnet, is to use one path at a time, preferably the path with the lowest delay. Another application is one that needs to send urgent messages; here, the appropriate mode may be to send urgent messages on multiple paths concurrently, minimizing the message delivery time to the minimum time of all the paths used.

In general, usage mode varies with application needs. In the foreseeable future, applications might require paths from different path services at the same time, thereby requiring different usage modes. For example, a Web session may have concurrent large file transfers and time-critical user interactions. In this scenario, the appropriate usage mode may be to use paths from different services and multiplex data among these paths according to the type of the data.

One expects that as network applications become more sophisticated, their usage modes will increase in complexity as well. The multipath routing model is able to accommodate these complex applications because it does not place restrictions on usage modes. Thus, nodes are allowed to use the offered paths in ways that best fit their communication needs.

4.2 Multipath Routing Overview

This section presents a conceptual overview of the multipath routing model in order to describe the potential benefits and costs of providing multiple paths and to highlight the components needed to make multipath routing viable.

This section is organized as follows. The first two subsections present the advantages and disadvantages of the multipath routing model and argue that the flexibility of the model offers significant network performance gains that outweigh the potential disadvantages. The latter two subsections, 4.2.3 and 4.2.4, present the key multipath implementation issues. Solutions to resolve these issues are addressed in succeeding chapters.

4.2.1 Multipath Advantages

The multipath routing model offers applications the ability to increase their network performance. Because of its multi-service paths, multi-option paths, and end-hosts' freedom to use these paths, the model provides a flexible interface to network resources that enables applications with varying network demands to increase their performance.

In general, multipath performance improvements are obtained in two ways. First, multi-service paths allow an application to use paths within a service that best suit the application's communication needs. Second, multi-option paths provide more network resources per path service, allowing applications to aggregate these path resources. These two general approaches are discussed below.

Providing the Right Paths

A multipath network with multi-service paths improves network performance because it allows applications to choose the paths that best suit their communication style. For example, an application such as FTP can improve its performance, measured in throughput, if it uses high-bandwidth service paths. Similarly, an interactive application, such as Telnet, can increase its performance, measured in response time, if it uses low-delay service paths. Because network performance depends on application demands, networks that provide paths with characteristics that fit these demands will be able to increase application network performance. Since network demands vary with applications, the generality of a multi-service paths allows a multipath network to satisfy the needs of different applications.

Providing the appropriate paths to increase performance will become more significant as the diversity of network applications increases. For example, in the foreseeable future, network applications such as IP telephony, real-time medical imaging, and video conferencing will become more prevalent. These applications need paths with very different characteristics from those of traditional applications. Specifically, many of these new applications need paths with QoS and real-time guarantees. In this environment, a multi-service network might provide paths with different delivery guarantees, allowing applications to select the paths that best suit their needs.

Notice that in a single path routing model, it is in general not possible to customize the one path between a node pair with characteristics suitable for all applications. In practice, single path routing algorithms calculate paths that compromise between throughput and delay [94]. Although the paths generated by this single path compromise are sufficient for today's applications, it seems unlikely that these paths can effectively support future applications that need paths with radically different characteristics.

Aggregating Multiple Paths

Multi-option paths increase application performance by giving applications the freedom to use multiple paths within the same path service. Performance improvement is obtained in two prototypical ways: 1) aggregating resources of multiple paths and 2) selectively using the best available path. The descriptions of these two methods are given below.

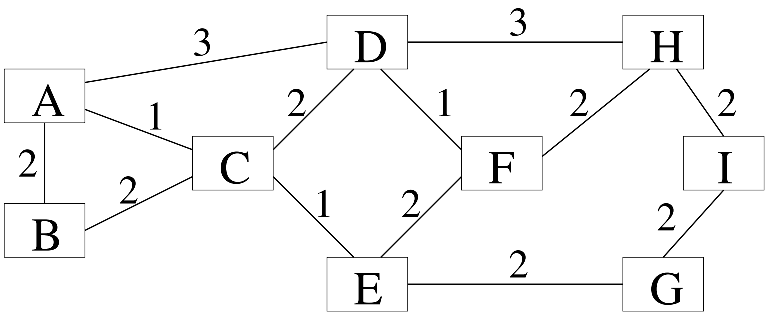


Figure 4.1 : An example network topology. Boxes represent network nodes and edges represent links. The number above each link shows the link's capacity, and all links have uniform delay.

Using multiple paths, the simplest method to increase performance is to aggregate path resources. For example, consider node B maximizing its throughput to node D in Figure 4.1. In this figure, boxes denote nodes and the number above each link denotes the link's capacity. In a single, shortest path network, the maximum bandwidth between B and D is 2 units: either the path (B, A, D) or (B, C, D), but not both. However, a multipath network can provide both paths to B, allowing B to send data to D at 4 units of bandwidth. Similarly, an QoS application can increase the probability that the network satisfy its QoS request by combining the QoS reservations it makes on multiple paths. In general, this style of resource aggregation can benefit any application that can use multiple paths in parallel.

End-hosts can also use multi-option paths to increase performance by selectively using the available paths. For example, consider an application interested in low delay. In a single path model, the application has no choice but to incur the delay of the one path provided by the network to its destination. On the other hand, an application in a multipath network can attain lower communication delays by probing the delays among the available paths to the destination and then choosing the minimum delay path. Moreover, if low latency is one of the multi-service paths, a node can choose among the paths in that service category.

In summary, end-to-end performance is measured with respect to application demands; thus, different applications maximize their performance differently. These differences are accommodated by multipath routing model's multi-service and multi-option paths. This model offers a flexible interface to network resources that allows different applications to increase end-to-end performance compared to single path routing models.

4.2.2 Multipath Disadvantages

Routing algorithms play a major role in end-host resource usage because they allocate network resources (in terms of paths) between nodes. By construction, multipath routing algorithms offer more network resources to end-hosts than do single path routing algorithms, both to specific destinations and to sets of destinations. The previous section showed that the advantage of multipath routing is that these additional resources can be used to increase end-host performance; however, a potential disadvantage is that these same resources can also be used by a greedy or malicious user to deny other users their fair share of network resources. This section argues that although multipath routing may exacerbate resource denial, the problem is actually orthogonal to single or multipath routing.

For example, consider a FIFO datagram network that does not place any restrictions on how much data end-hosts can send. In this network, excessive resource consumption and denial of services cannot be prevented [52]. In today's Internet (a single path, FIFO datagram network), nodes can blast packets to random network destinations, consuming a significant amount of resources and drastically degrading overall network performance. Although malicious users in a FIFO multipath datagram network could deny more resources than in a FIFO single path data network, the fact remains that FIFO datagram networks do not have mechanisms to prevent resource abuses, whether single or multiple paths are provided.

In general, two approaches can prevent or reduce excessive resource denial: cooperative network communities and enforced network policies. In the cooperative approach, network users/applications agree not to consume excessive resources. The Internet uses this approach via congestion sensitive transport protocols (e.g. TCP [83]). These protocols attempt to share resources fairly by regulating their sending rates in response to the available bandwidth of the path they are using. Internet resource abuses are low because these protocols are used by most users [124]. The advantage of the cooperative community approach is that it does not require any network support – Internet routers do not need any additional mechanisms to prevent resource abuse because end-hosts voluntarily do not abuse resources. However, the disadvantage of this approach is that users (e.g. malicious ones) may not abide by the convention and thus can consume excessive resources. In order to control these users, a network needs mechanisms for admission and traffic control (described below). The important point is that the success or failure of the cooperative approach is independent of whether single or multiple paths are provided between nodes: if all hosts cooperate, excessive resource consumption will not occur in either single or multiple path networks. Similarly, if hosts do not cooperate, resource abuse can occur in both types of networks.

The second method to prevent/reduce resource abuse is for the network itself to enforce admission and traffic control policies [86, 135]. These networks enforce end-host resource usage by regulating the number of senders and/or the amount of data each sender sends. Regulating network traffic requires specific mechanisms in the network (e.g. in routers) to monitor and enforce end-host sending policies. For example, a pricing network implicitly discourages resource abuse by charging users for packets they transmit [121]. Users in these networks are unlikely to maliciously consume excessive resources because they have to pay for the resources they use.

Again, notice that the enforceability of traffic control in pricing networks is independent of whether single or multiple paths are provided between nodes. Users in a pricing network pay for packet transmission, regardless of whether they send their packets on different paths or on the same path. Other networks with mechanisms and policies to discourage/prevent excessive resource consumption can be found in [91,92,113,121,135]. Although the actual implementation of the admission and traffic control mechanisms may differ, the enforceability and effectiveness of admission and traffic control policies are orthogonal to the routing model.

The advantage of the network enforcement approach is that it does not rely on the cooperation of end-hosts; thus, it is much more robust and can prevent malicious users from abusing the network. The primary disadvantage is that it requires network mechanisms to regulate traffic and prevent abuse. These monitoring and enforcement mechanisms increase network cost and may decrease performance because of additional packet processing.

In summary, although multipath routing can exacerbate resource abuse in certain types of networks, the core issues of resource abuse are orthogonal to both single and multipath routing. That is, the fact that a network is prone to resource abuse is independent of whether it provides single or multiple paths between nodes. Because of this independence, the remainder of the thesis assumes that users are greedy but not malicious. That is, end-users greedily maximize their resource usage but attempt to avoid network congestion. This is the same end-user assumption used in today's Internet.

4.2.3 Multipath Implementation Cost

The advantages of multipath routing come at a cost. Recall that routing is a two-step process: 1) calculating paths and 2) forwarding data on those paths. Implementing these two routing tasks incurs the following three cost categories:

1. The cost of computing multiple paths
2. Per packet path specification overhead in bytes
3. Router overhead of processing and forwarding data packets.

The first category corresponds to the cost of path computation, and the latter two to the cost of forwarding data on the computed paths. The three costs are described below.

Computing Multiple Paths

The first cost category, computing multiple paths, is measured in terms of routing messages (in bytes) needed to propagate routing information and the router CPU time needed to compute multiple paths. The number of messages and the amount of CPU usage depends on the path calculation algorithm and the base routing algorithm.

For example, the number of routing messages needed to compute multiple paths heavily depends on whether the routing algorithm is based on LS or DV. In an LS environment, routing message overhead is generally low because path computations are done with the knowledge of the network topology. Thus, the number of messages needed to disseminate topology information is the same independent of whether multiple or single paths are calculated. In contrast, DV based algorithms use routing messages (in the form of Distance Vector packets) as the mechanism to propagate paths; therefore, computing multiple paths generally requires more DV messages than computing single paths.

The amount of router CPU time to compute multiple paths has similar dependencies. In LS, because path computations are centralized, standard complexity analysis of centralized algorithms suffice to measure LS router CPU usage. As examples, calculating multiple K

initial link disjoint paths takes $O(K * E * lg(E))$ and calculating the shortest K loop free paths takes $O(nE * lg(E))$ [58, 141, 156], where n is the number of nodes (or routers), and E is the number of network edges. In DV, the analysis is not so straightforward because path computations are distributed. In the worse case, the message and CPU complexities are exponential [25]. However, it has been shown that the average message and CPU complexities is $\Theta(nM^3(ln(M))^2)$ [23, 157], where M is the average number of neighboring routers. Efficient multipath calculation algorithms based on DV and LS are given in Chapters 5 and 6.

Specifying Multiple Paths

Because there are multiple paths between nodes, every packet needs to specify not only its destination, but also a particular path to that destination. This is in contrast to single path networks where a destination address uniquely specifies a packet's path. The second multipath cost category refers to this additional per packet cost of path specification. The specification cost is measured in the number of bytes needed in order to ensure that a packet travels its specified path. It is critical to minimize this cost because it is incurred on every data packet. Chapter 6 describes an efficient method for packets to specify a particular path to a destination.

Forwarding Multiple Paths

Finally, the per packet path specification implies that more router processing is needed to forward each packet. The additional processing is needed for a router to decide, given the packet's destination address and path specifier, which outgoing link the router should forward the packet to. This additional processing may slow router forwarding speed and decrease network performance; thus it is critical to minimize this processing time. The cost of this additional processing is called the router forwarding overhead. Not surprisingly, this overhead is closely tied to how paths are specified in data packets. The efficient path encoding method presented in Chapter 6 has low forwarding overhead.

4.2.4 Multipath Benefits

The previous subsections list the potential advantages, disadvantages, and costs of a multipath network; this subsection concludes the multipath discussion by addressing how the benefits of a multipath network can be obtained. To obtain multipath benefits, 1) multipath networks need to calculate appropriate paths, and 2) end-hosts need to effectively use these paths. These two properties are described below.

Path Calculation

The extent of performance improvement users can obtain from a multipath network depends on the quality of the calculated paths. For example, an application can obtain higher throughput *only if* the multiple paths calculated actually provide greater combined bandwidth than the one provided by a single path routing algorithm. Similarly, an application can increase its probability of establishing a QoS connection only if the calculated paths have, either individually or combined, a higher probability of satisfying QoS requests compared to the probability of a single path.

For example, consider the three paths (A, D, H, I), (A, C, D, H, I), and (A, C, D, F, H, I) from node A to node I in Figure 4.1. These three paths are not well chosen if A wishes to increase its network throughput to node I because all three paths traverse the same bottleneck link (H, I) with capacity 2. On the other hand, if a multipath network provides paths (A, D, H, I), (A, B, C, E, G, I), and (A, D, F, E, G, I), then node A has 4 capacity units to node I.

As the example shows, a multipath network must provide the "right" paths in order for nodes to obtain higher performance gains, where the right paths depend on the applications that use the paths. In general, providing quality paths is a two-step process. First, determine the type of path services to calculate (path specification), then develop efficient algorithms to calculate them (path calculation algorithm).

Two algorithms are developed in this thesis, one that calculates paths to maximize throughput and the other to minimize latency. The complexities of both algorithms are linear in the number of paths calculated compared to computing single shortest paths. The algorithms are described in Chapter 5.

Path Usage

The second component necessary for end-hosts to obtain increased performance in multipath networks is effective end-host usage of multiple paths. The fact that a network provides quality paths between nodes does not necessarily imply that nodes are able to effectively use these paths to maximize performance. This subsection shows the importance of effective multipath usage and its impact on network performance.

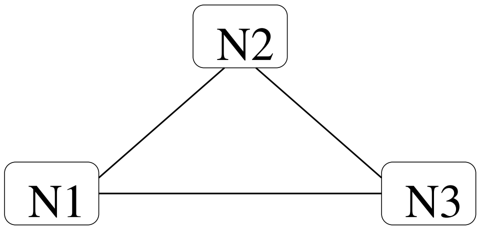


Figure 4.2 : A simple three node network with full-duplex links. All links have equal bandwidth and latency.

Consider the simple network in Figure 4.2 where all links have equal capacity and delay. Here, the multipath routing algorithm calculates two link disjoint paths (the one link and two link paths) between every node pair. Notice that the calculated paths provide the optimal paths for maximizing throughput. In this setting, the two paths calculated provide twice the capacity between nodes compared to single path routing; however, the effective throughput between nodes depends on how each node uses its paths.

To show the potential harm of naively using multiple paths, this three node network was simulated using TCP and a non-congestion aware, multipath striping protocol. The multipath striping protocol clocks the sending of its data at full link capacity and distributes the data by striping them along the two available paths. That is, given N packets destined for destination D, the protocol sends packet $2i$ on the one-hop path and packet $2i+1$ on the two-hop path to D, $0 \leq i < N/2$. The single path TCP protocol sends all packets along the shortest path. In this experiment, nodes randomly select a neighbor and then send a burst of packets to that neighbor. The times between each burst are exponentially distributed. Figure 4.3 shows TCP throughput versus the striping protocol's throughput when all three nodes are sending data.

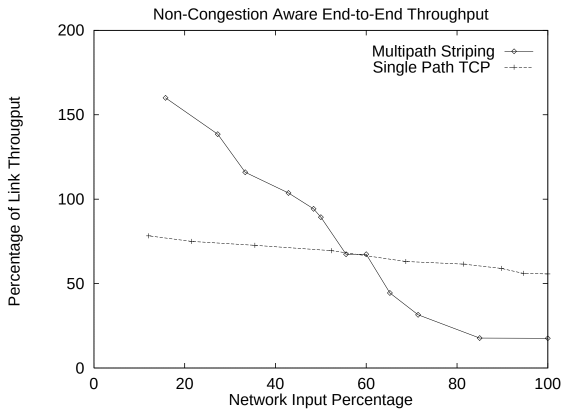


Figure 4.3 : Simulation result using the network in Figure 4.2. The graph shows the performance of single path TCP versus a multipath striping protocol without congestion control.

In Figure 4.3, the y-axis represents the average end-to-end throughput of all nodes as a percentage of link capacity, and the x-axis represents the aggregate sending rate normalized by the total network capacity.

First, the graph shows that TCP performance is very stable despite the increase in traffic. TCP does not achieve the maximum link bandwidth because its congestion and flow control mechanisms cautiously probes the network for available bandwidth and reduces its sending rate upon detection of congestion. On the other hand, the non-congestion aware multipath striping protocol achieves very high throughput at low network utilization levels. The reason is that at low network loads, the amount of contention from other connections is low; therefore, multiplexing data between the one link and two link paths allows effective aggregation of path resources, resulting in higher throughput than the single path strategy.

However, the relative performance of these two strategies changes at higher network loads ($\geq$ 60%): here, the TCP's transmission strategy proves superior because at these utilization levels, packets in the multipath striping protocol experience enough contention from other connections to cause significant performance degradation (due to packet queuing in router buffers). This contention is due to packets traveling on the two link paths competing with packets from other connections. Furthermore, the contention increases as more packets are injected into the network, resulting in degradation of both aggregate and individual throughput.

To address this problem, this thesis develops a congestion-aware multipath transport protocol, MPTCP. Details of the MPTCP protocol are given in Chapter 7. The effectiveness of MPTCP is again compared against a single path TCP protocol on the same three node network. The results are shown in Figure 4.4.

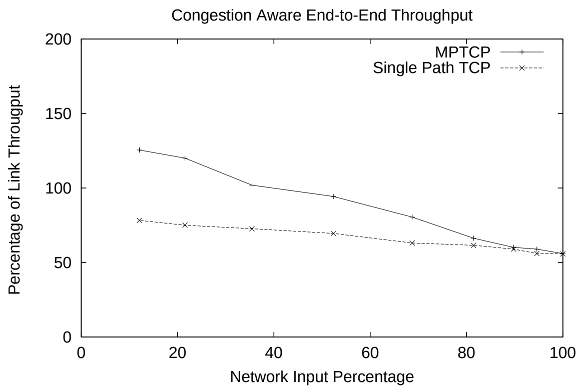


Figure 4.4 : Simulation result using the network in Figure 4.2. The graph shows the performance of single path TCP versus MPTCP.

The x-axis and y-axis have the same representation as the previous figure. As this figure shows, MPTCP outperforms the single path protocol at all levels of network utilization. This demonstrates MPTCP's ability to adapt to network conditions in order to increase throughput.

Compared to the multipath striping protocol, MPTCP does not achieve the same level of end-to-end throughput at low network utilization. This is because MPTCP is congestion aware and incrementally tests the network for available bandwidth. This cautious approach results in lower performance when the network is underutilized because it takes time for MPTCP to fully exploit available path bandwidth. However, the same cautious approach allows MPTCP to significantly outperform the striping protocol at high utilization levels.

The conclusion of this section is that in order for end users to benefit from a multipath network, the network needs to provide not only the right paths, but the end-hosts also need to be able to take advantage of the additional paths. As shown by a naive multipath striping approach (Figure 4.3), wrongly using multiple paths can degrade not only end-to-end performance, but also the performance of other connections. This section also shows that correctly using multiple paths can increase network performance and avoid performance degradations.

4.2.5 Static and Dynamic Metric Multipath Routing

Like single path routing, multipath routing algorithms can use either dynamic or static metrics. In dynamic metric multipath routing, routing algorithms monitor link costs and recompute paths upon detection of link cost changes. This thesis does not specifically address the dynamic metric multipath routing model because we believe this model can be implemented by traditional dynamic metric triggering mechanisms; thus the methods presented in this thesis for static metric multipath routing are also applicable to dynamic metric multipath routing.

As stated in Chapter 2, the fundamental difference between multipath routing and single path dynamic metric routing is that end-hosts in multipath networks control the use of its paths on a much finer time and path granularity. Thus, given appropriate paths and end-host protocols, multipath end-hosts can dynamically detect poor path performance, and then switch and use other paths that provide better performance.

For example, consider an application that wishes to minimize its communication delay to its destination, and the multipath routing algorithm calculates link disjoint paths. In single path dynamic metric routing, if the path to the destination is congested, then the application will incur the delay caused by the congestion, unless the routing algorithm recompute a better path. In multipath routing, however, the application can dynamically switch and use other paths to avoid the congestion without any router intervention. Thus, on small time scales, the ability to control path usage allows end-hosts in multipath networks to dynamically adjust to transient traffic patterns.†

Because end-hosts can adapt to small time-scale traffic patterns, a dynamic metric multipath routing algorithm does not need to recompute paths in fine time granularities. Consequently, these routing algorithms should recompute paths that consider large time-scale traffic patterns. For example, a dynamic metric multipath routing algorithm might monitor traffic patterns for weeks and then recompute paths based on the gathered statistics. This approach to path recomputation provides better paths by considering traffic trends rather than transient traffic bursts (which are addressed by end-host multipath protocols). On this time scale, problems such as route oscillations and excessive routing costs do not occur. Thus, the traditional LS and DV triggering mechanisms are sufficient to implement the dynamic multipath routing model. The multipath routing algorithms developed in this thesis are based on DV and LS.

4.3 Multipath Routing Summary

While it is clear that the static metric multipath routing model offers many advantages over single path routing models, it is unclear whether enough benefits could be extracted to offset the cost of their implementation. In short, in order to make multipath routing viable, the following questions need to be resolved:

1. What paths should be calculated between nodes and how?
2. How should routers efficiently provide multiple paths in a distributed routing environment?
3. How should end-hosts use multiple paths to gain higher performance?

† This conclusion assumes that the multipath routing algorithm provides quality paths between nodes (Chapter 5) and that end-hosts effectively use these paths to increase their performance (Chapter 7).

The first question deals with the potential gains of a multipath network. As illustrated in Section 4.2.4, one of the necessary criteria for multipath networks to increase end-to-end performance is to calculate the right paths. To address this issue, Chapter 5 surveys different multipath calculation algorithms. We then develop two algorithms, one maximizing throughput and the other minimizing delay. The two algorithms appear in Chapter 5.

The second question deals with the cost of providing multiple paths between nodes. The main cost of implementing multipath routing is solving the packet forwarding problem: how to efficiently forward packets to the same destination but on different paths? The novel solution developed in this thesis uses routing overhead linear in the number of paths between nodes and has constant per packet path specification overhead. This low overhead is achieved by requiring that paths calculated by a multipath routing algorithm satisfy the *suffix matched* property. Complete details of this forwarding method and its requirements are given in Chapter 6.

The last question is how end-hosts should best use a multipath network in order to increase their performance. As demonstrated in the previous section, the way in which multiple paths are used has a dramatic impact on individual and aggregate performance. A congestion aware multipath transport protocol, MPTCP, is developed that effectively uses multiple paths to increase throughput. The details of the protocol and its strategies are provided in Chapter 7.

The algorithms and methods developed in Chapters 5 – 7 are combined to implement a multipath network. Two multipath routing algorithms are implemented: one based on LS and the other on DV. Details of their implementation are given in Chapter 8. Finally, Chapter 9 presents simulation results that measure multipath routing's obtained performance and incurred costs.

Chapter 5

Path Calculation Algorithms

To provide multiple paths between nodes, a multipath routing algorithm first needs to calculate the paths it wishes to provide. A *multipath calculation algorithm* refers to an algorithm that calculates multiple paths between node pairs. The paths calculated between nodes directly affects the performance gains a node pair can obtain. For example, a throughput application can only increase its throughput if the paths provided has a larger aggregate bandwidth than the path provided by a single path routing algorithm.

To this end, this chapter surveys different path calculation algorithms and develops two multipath calculation algorithms, one that computes low delay paths and the other high bandwidth paths. Both algorithms are based on Dijkstra's shortest path algorithm.

Before presenting the two algorithms, we first specify two path characteristics that allow end-hosts to benefit from using multiple paths. Section 5.1 presents these two path characteristics. Then in the succeeding sections, we use these characteristics to guide the development of multipath calculation algorithms that calculate paths for low delay and high bandwidth path services.

5.1 Path Characteristics

The quality of a path set depends on the intended use of the paths in the set. For example, a path set that provides high throughput may be very different from one that provides low delay. Although path set quality (measured in potential performance improvements) depends on the intended use of the path set, in general, there are some path characteristics that allow applications to increase their performance.

In this section, we describe two such path characteristics, *path quantity* and *path inde-*

pendence. Path quantity refers to the number of paths calculated between nodes, and path independence refers to the disjointedness of the calculated paths. These two characteristics are based on the basic advantage of multipath routing – the increase of end-to-end performance by providing multiple paths between nodes. In general, this increase in performance is obtained through 1) aggregating path resources and 2) choosing the best path(s) among the available paths. We describe the two path characteristics below.

Path Quantity

The fundamental difference between a multipath network and a single path network is the number of paths provided between nodes. With more paths, nodes have more options to interact with other nodes, potentially increasing their performance. Thus, the higher the number of paths calculated between nodes, the higher potential a path set can improve end-to-end performance.

There are many ways to describe path quantity. One method is to consider the total number of paths calculated in a path set; the larger the number, the better the path set. While this is straightforward, it does not capture the distribution of paths between node pairs. For example, assuming uniform traffic distribution, a path set that provides every node pair with 3 paths should be better than one that provides half the node pairs with 1 path and the other half with 5. That is, path sets with the same averages but higher variance are less desirable because it means that some nodes have few paths while others have many.

Although there are subtleties in the precise description of path quantity, in general, a multipath calculation algorithm that calculates more paths is better than one that calculates less. However, path quantity only characterizes one aspect of a multipath set. Another characterization is the independence among paths calculated between a node pair. This is described next.

Path Independence

The second path characteristic, path independence, describes the disjointedness of the paths provided between nodes. This property is important because the more independent a path set, the more aggregate physical resources the set offers between a node pair (because those resources are not shared), and the less likely the performance of one path affects performances of other paths.

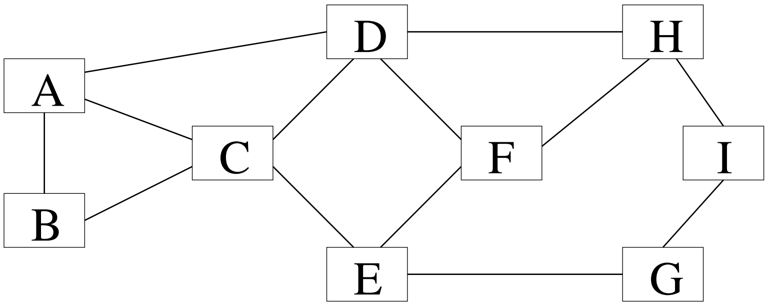


Figure 5.1 : An Example Network Topology. All links have uniform capacity and delay.

To illustrate the importance of path independence, consider the network in Figure 5.1. Assume that a path set that has two paths from A to I as (A, D, H, I) and (A, C, D, H, I) and another path set with paths (A, D, H, I) and (A, C, E, G, I). Intuitively, paths in the second set are more independent than the ones in the first set because paths in the second set do not share links. The higher independence of the second set gives node A more aggregate capacity to I. In addition, suppose a link on A's shorter path is congested, then the probability that A's other path to I is also congested is less likely in the second path set. This is because with the second set, at least one link in each paths must to be congested in order for A to suffer congestion; whereas in the first set, A would suffer congestion affects if either link (D, H) or (H, I) is congested.

In addition to increasing performance, path independence also allows end-hosts to achieve higher network availability. Assume that a network link has failed and that end-hosts can detect that a path is not delivering data* faster than the routing algorithm. In this case, an end-host has a higher probability of reestablishing connectivity on a different path if the paths provided are independent. In contrast, in a single path environment, where there is no path independence (because there is only one path between nodes), an end-host must wait for the routing algorithm to detect the failure and recompute the appropriate paths in order for the end-host to reestablish connectivity.

In summary, we described two desirable multipath characteristics, path quantity and path independence. Multipath sets with these characteristics better allow end-hosts to increase their performance, compared with path sets that do not.

5.2 Path Calculation Algorithms

This section uses path quantity and independence to guide the development of multipath calculation algorithms for high throughput and low delay path services. These two services are chosen because their paths can provide immediate benefits to current network applications. The efficiency requirement of the considered algorithms is that their message and CPU usage are within a linear factor in the number of paths calculated, compared to computing single shortest paths. This complexity bound is imposed to make the implementation of these algorithms feasible in real networks.

In both low delay and high throughput paths, the performance of these paths depends on the dynamics of network traffic. As stated in Chapter 4, a multipath routing algorithm (both static and dynamic metrics) should compute paths that allow end-hosts to adapt to transient (fine time-scale) traffic dynamics. That is, the paths provided between nodes should be such that they preserve their intended performance benefits under varying traffic conditions.

We use the two path characteristics given in the previous section to guide the development of path calculation algorithms that compute paths which preserve their performance

*End-hosts can detect failed links in a path by observing that the path is not delivering data to the destination.

benefits under different traffic conditions. Section 5.2.1 presents an algorithm that computes paths to decrease transmission delay, and Section 5.2.2 presents an algorithm that increases throughput.

5.2.1 Minimizing Delay

One of the ways end-hosts can benefit from a multipath network is to lower their communication delays (or increase responsiveness) to other nodes. This can be done by monitoring the round trip delay of available paths to the desired destination and dynamically choosing the one with the least delay[†]. Since the minimal delay that an end-host can achieve depends on the delay of the paths it has to its destination, end-host delay performance is closely tied with the paths calculated. This section develops a path calculation algorithm that provides the low delay path service.

In order to calculate paths that minimize delay, a path calculation algorithm needs to gauge the expected delay of a link. For the following discussion, we assume that link delays are given by a cost function; the higher a link's cost, the higher the expected latency of the link. In addition, for ease of explanation, we assume uniform link costs; therefore, a path's delay is determined by the number of hops.

For the low delay path service, the objective is to calculate paths so that the smallest delay path between a node pair is minimized. Given that a path's actual delay is traffic dependent, a multipath calculation algorithm needs to calculate paths so that low delay can be obtained under different traffic patterns. Below, we present several potential algorithms and highlight their advantages and disadvantages.

[†]Path symmetry is not needed here. Path symmetry refers to the property that the path from A to B must be identical to the reverse of the path from B to A. Symmetry is not necessary in real networks because network traffic in one direction has no correlation with traffic behavior in the reverse. Thus, path symmetry is not required for the correct functioning of data transport protocols.

Shortest K Paths

The most natural algorithm for minimizing delay is one that calculates paths with the shortest total delay. A family of algorithms, called the shortest K algorithms, compute K paths such that the total cost of the paths is minimized. Moreover, there are efficient solutions to implement these algorithms [45, 58, 165].

Given that the link costs reflect delay, these algorithms provide path sets with the minimum total delay. However, the problem with these sets is that they do not consider path independence. For example, consider the network in Figure 5.1 where three paths are computed from node A to H. A shortest K algorithm will compute the paths (A, D, H), (A, C, D, H), and (A, D, F, H). Although this path set the three smallest hop paths (8 hops total), it is clearly not a desirable set because of the high degree of link sharing (i.e. lack of path independence). For example, in this path set, if the links on the shortest path are congested (i.e. links (A, D) and (D, H)), then the other two paths will be congested as well because these two links are also part of those paths. This lack of independence limits the amount of benefits end-hosts can obtain under different traffic conditions.

In general, with shortest K algorithms, paths between node pairs will tend to share many links, which reduces the effectiveness of providing multiple paths.

Link Disjoint Paths

Algorithms that overcome the problem of path independence are ones that calculate link disjoint paths between nodes. Link disjoint paths are paths that do not have any links in common. Like the shortest K algorithm, there are many algorithms to efficiently calculate these path sets [36, 118, 146, 152].

However, the link disjoint algorithm achieves high path independence at the expense of path quantity and path delay. For example, consider the paths calculated by a link disjoint algorithm from A to H in Figure 5.1. One possible path set is (A, D, H), (A, C, E, F, H), and (A, B, C, E, G, I, H). Here, the paths are disjoint, but the combined length of the path set is 12 hops. This is 50% longer than the path set calculated by shortest K algorithm.

Because different paths between a node pair cannot share links, this may be too restrictive, resulting in paths of excessive delay or a small number of calculated paths. Both of these consequences lowers the probability that end-hosts can obtain low delay paths to their destinations.

Discount shortest path

In considering path quantity and path independence, we developed a path calculation algorithm called the discount shortest path algorithm. The idea of the algorithm is to add a uniform cost to all links on the previously calculated paths. Therefore the paths calculated are compromises between the paths calculated by the shortest K and link disjoint algorithms: the link disjoint algorithm is one that adds an infinite cost to used links, while the shortest K algorithm selectively adds the minimum cost to the used links so that the next shortest path calculated is distinct from the computed paths. The basic discount shortest path algorithm is a variation of Dijkstra's shortest path algorithm. Similar variations have been published in the literature [156].

The discount shortest path algorithm assumes that for paths between any two nodes, there is an upper bound on the cost of the longest path. This is reasonable since most path calculation algorithms do not wish to calculate arbitrarily long paths. C_{max} is used to denote the maximum admissible path cost between a node pair. An example of C_{max} is 3 times the cost of the shortest path [94, 129].

The discount shortest path algorithm calculates paths with the following properties: from node a to b, the i^{th} path is the least-cost path from a to b such that the path's cost is less than C_{max}. The cost of path i is calculated after adding cost increments to each link in path j from a to b, $1 \leq j < i$, where the cost increment of a path P is $(C_{max} - Cost(P))/Length(P)$.

To calculate discount shortest paths from node a to b, the algorithm first calculates the shortest path P with cost C_p from a to b. C_{max} is then calculated as `Cost_BOUND`$*C_p$ (in our implementation, `Cost_BOUND` is set to 3). Next, the cost increment for this path is calculated as $P_{incr} = (C_{max} + 1) - C_p$. That is, a path's cost is incremented by the smallest amount so that the path exceeds the C_{max} and therefore will not be admissible in subsequent computations. This cost increment is then added uniformly to the cost of all links on path P. That is, for every link in P, the cost is incremented by $P_{incr}/length(P)$.

To get the next path from a to b, this process is repeated using the newly incremented link costs. The algorithm stops when either K paths are computed or there does not exist paths from a to b with cost less than C_{max}. After computing the paths from a to b, the link costs are restored to their original costs, and the discount shortest path computation begins for another node pair. The pseudocode for the discount shortest path algorithm is given in Figure 5.2.

To calculate K paths from node a to b, the discount shortest path algorithm iterates Figure 5.2's main loop K times. In each iteration, the function `Get_Shortest_Path()` is called. Given E edges and n nodes, the function which takes $O(E * lg(E))$. In addition, on each iteration, each link in the newly calculated path is traversed, which takes $O(n)$. At the end of the loop, the added link costs are restored ($O(K * n)$). Thus, the complexity of the discount shortest path algorithm in computing K paths from a to b is $O(K * (E * lg(E) + n) + K * n) \rightarrow O(K * E * lg(E))$. Notice that this is K times the complexity of calculating the single shortest path between two nodes.

5.2.2 Maximizing Throughput

This subsection describes a path calculation algorithm for another path service, the throughput path service. Throughput oriented applications such as FTP benefit from this path service because multiple paths can increase their effective throughput. Again, the amount of throughput an application obtains depends not only on the paths calculated, but also on network traffic patterns. Thus, the paths calculated should be robust in the sense that they provide high throughput under a variety of traffic conditions.

Again, without loss of generality, we assume that expected link bandwidth can be characterized by a link capacity metric; the higher the link capacity, the higher the expected

```
numPaths = 0;
while(numPaths < K)
{
   newPath = Get_Shortest_Path(Src, Dst);
   if (newPath == NULL)
      break;
   if (numPath == 0)
      Max_cost = Cost(newPath) * Cost_BOUND;
   if (Cost(newPath) > Max_cost)
      break;
   numPaths++;
   StorePath(Src, Dst, newPath);
   Cost_diff = Max_cost - Cost(newPath) +1;
   Cost_incr = Cost_diff / Length(newPath);
   forall links l ∈ newPath
      l.cost = l.cost + Cost_incr;
}
Restore all link cost additions
```

Figure 5.2 : The pseudocode for the discount shortest path algorithm. The code shows the calculation of K discount shortest paths between Src and Dst.

available bandwidth of the link.

Maximum Flow Paths

The most straightforward algorithms to calculate throughput paths are maximum flow algorithms. Calculating a set of links that provides the maximum flow between two nodes can be done using conventional flow algorithms [5, 42]. However, applying these algorithms directly has two main drawbacks.

First, the maximum flow algorithms do not produce a set of end-to-end paths; that is, a maximum flow algorithm produces a set of links and capacities so that sending the specified amount of data on each of those links provides the maximum flow. Thus, to calculate paths requires an additional level of processing to convert the set of links into a set of paths. In addition, since routing algorithms typically bound the number of paths it calculates (say K) between a node pair, an algorithm needs to select K paths that maximizes flow from the set of links. Selecting these K paths is not trivial because it depends on how the paths are initially chosen from the maximum flow link set.

The second drawback of directly apply maximum flow algorithms is that these algorithms do not consider path length (or path costs). Given that routing algorithms typically want to control the length of the paths they calculate, the paths derived from a maximum flow calculation may not provide a good path set given a length threshold.

Capacity Removal Algorithm

Due to the drawbacks of strictly applying maximum flow algorithms, we have developed an algorithm called the capacity removal algorithm based on Dijkstra's shortest path algorithm. Similar to maximum flow, this algorithm calculates paths that aim to increase the flow between node pairs. Moreover, the capacity removal algorithm explicitly considers the number of paths to calculate and path length.

As in the discount shortest path algorithm, the capacity removal algorithm calculates successive shortest paths; after calculating a path, the algorithm subtracts the path capacity from every link along that path. The capacity of a path is the minimal capacity of all links on the path. A link capacity threshold is used so that links with capacities below the threshold are eliminated from subsequent path computations. The pseudocode for the capacity removal algorithm is given in Figure 5.3.

The description of capacity removal paths can be summarized as follows: from node a to b, the i^{th} path is the least-cost path from a to b with cost less than C_{max} and capacity greater than the capacity threshold, where path i's capacity is calculated after subtracting the path j's capacity from every link in path j, $1 \leq j < i$.

The complexity analysis of the capacity removal algorithm is very similar to the discount shortest path algorithm. To calculate K paths from node a to b, the capacity removal algorithm iterates Figure 5.3's main loop K times. In each iteration, the function `Get_Shortest_Path_CapThresh()` is called which takes $O(E * lg(E))$, and each link in the newly calculated path is traversed $O(n)$ times. At the end of the loop, link capacities are restored ($O(K * n)$). Thus, the complexity of the capacity removal algorithm to calculate K paths between a node pair is $O(K*(E*lg(E)+n)+K*n) \rightarrow O(K*E*lg(E))$. Again, like discount shortest path, the complexity of the capacity removal algorithm to calculate K paths is K times the complexity of calculating the single shortest path between two nodes.

5.3 Path Calculation Summary

This chapter presents two path calculation algorithms, one computing paths for low delay path service and the other for high throughput path service. The two algorithms developed, discount shortest path and capacity removal, are both shortest path based algorithms, and their complexities are linear in the number of paths calculated compared to computing single shortest paths.

The quality of capacity removal paths, measured in achievable throughput, is evaluated in Chapters 7 and 9. Simulation results show that the capacity removal algorithm effectively provides paths that allow end-hosts to increase their throughput.

```
numPaths = 0;
while(numPaths < K)
{
   newPath = Get_Shortest_Path_CapThresh(Src, Dst,
                                          Capacity_Threshold);
   if (newPath == NULL)
      break;
   if (numPath == 0)
      Max_cost = Cost(newPath) * Cost_BOUND;
   if (Cost(newPath) > Max_cost)
      break;
   numPaths++;
   StorePath(Src, Dst, newPath);
   Path_cap = Capacity(newPath);
   forall links l ∈ newPath
      l.capacity = l.capacity - Path_cap;
}
Restore all link capacity subtractions
```

Figure 5.3 : The pseudocode for the capacity removal algorithm. The code shows the calculation of K capacity removal paths between Src and Dst. The function `Get_Shortest_Path_CapThresh(Src, Dst, Cap_thresh)` returns the shortest path from `Src` to `Dst` such that all links in the path have capacity above `Cap_thresh`.

Chapter 6

Multipath Forwarding

Network routing is a two-step process: the first step is computing the desired paths, and the second is forwarding data on those paths. In a multipath network, the cost required to support these two steps is higher than in single path routing. A summary of the multipath overheads are

1. Algorithmic cost: router CPU and memory required to calculate multiple paths
2. Routing costs: extra forwarding table entries, CPU cycles, and routing messages
3. Per packet path specification costs: number of bits required to specify a packet's path
4. Per packet forwarding time

For all but the algorithmic cost (addressed in Chapter 5), the costs listed above are directly related to how paths are encoded and forwarded in a network. This chapter presents solutions that efficiently accomplishes this forwarding task.

In multipath networks, packets need to specify not only their destinations, but a specific path to their destinations. This means that routers need to recognize path specifications (or encodings) in order to forward packets correctly. The problem of encoding and forwarding packets along their intended path is called the *path forwarding problem*.

Methods of solving the path forwarding problem depend on whether packets are forwarded on paths within the same path service (multi-option paths) or on paths from different path services (multi-service paths). This differentiation is important because it affects the implementation of the packet forwarding method. Path forwarding on different service paths can be implemented in a straightforward manner using a service identifier; however, this encoding scheme is not sufficient for multi-option path forwarding. Because multi-service forwarding is relatively straightforward, this chapter briefly discusses multi-service forwarding but primarily focuses on multi-option forwarding.

This chapter develops an efficient solution for the multi-option forwarding problem. A forwarding method is efficient if the per packet path specification cost is small (in the number of bits) and the path specification allows fast router packet forwarding. The solution developed uses 1) a per packet overhead of a small, fixed-length path identifier, and 2) router space overhead linear in K, the number of paths calculated between nodes. To achieve these efficient bounds, the forwarding method requires that multi-option path sets satisfy the *suffix matched* property.

The remainder of this chapter is organized as follows. The following section formally defines the multipath forwarding problem, both multi-service and multi-option. The section describes multi-service forwarding and develops the basis the proposed multi-option forwarding method. In Sections 6.2 and 6.3, the multi-option forwarding method is applied to Distance Vector and Link State routing algorithms. In addition, this chapter contains proofs that 1) Distance Vector based algorithms compute suffix matched multipath sets, and 2) Link State based algorithms yield suffix matched multipath sets for the criterion of ranked k-shortest paths. Section 6.4 provides an example to demonstrate the forwarding method in the context of a multi-service multi-option network. Finally, a summary of the path forwarding method appears in Section 6.5.

6.1 The Multipath Forwarding Problem

A network can be represented as a graph $G = (V, E)$, where V is the set of network nodes and E is the set of links or edges. Each link $(x_i, x_j) \in E$ has an associated positive cost $c_{x_i x_j}$. A path p in G is a list of nodes $(x_1, \ldots, x_n)$ such that $\forall i, 1 \leq i < n$, $(x_i, x_{i+1}) \in E$. Path p has cost $\sum_{i=1}^{n-1} c_{x_i x_{i+1}}$. No node appears more than once in a *simple* or loop-free path.

Abstractly, to route packets amongst nodes in a network, each node $x \in V$ implements two functions:

$$h_x : V \to \Lambda$$

$$\mathrm{G}_x : \Lambda \to V$$

The *path selection* function h_x chooses a path identifier ψ (on a per-packet or per-connection basis) for a given destination node $d \in V$ from the set Λ. Path identifiers encode distinct paths from x to d. The forwarding function G_x takes a path identifier $\psi \in \Lambda$ and provides the next-hop neighbor on path ψ. As an example, if node x wishes to send a packet to node d on path ψ, it adds the label ψ to the packet, and forwards it to $\mathrm{G}_x(\psi)$: the next-hop on path ψ from x to d.

Single shortest path routing is a special case where there is no need to have a path identifier set Λ. In this path setting, the path selection function is the identity function: $h_x(d) = d,\ d \in V$. This is because exactly one path p from x to d is used at any given time — there is no choice of paths.* By the Bellman optimality principle, the shortest path from any intermediate node n in p to node d is the subpath of p from n to d; thus from any node, the path to a destination d can uniquely identify by d's address.

One can replace the path identifier argument by the destination and simplify the forwarding function in a single shortest path setting to be $\mathrm{G}_x : V \to V$ which identifies the next-hop for a given destination. Router x implements G_x by looking up its forwarding table.

In a multipath setting, an originating node x has a choice of several paths to a particular destination at any given time (i.e, the range of the function h_x is a set of size ≥ 1). The particular path (identified by $h_x(d) = \psi$ chosen by x for a destination d is a function of the sending host's objectives (refer to Section 5.2).

*The actual identity of the shortest path between two nodes in the network may change as the network changes, but at any given time a node can route to a destination along exactly one path.

The forwarding functions G_x, for each $x \in V$ need to respect the following constraint:

$\forall \psi \in \Lambda$ such that ψ identifies path $= (x_1, \ldots x_n)$,
$\mathrm{G}_{x_i}(\psi) = x_{i+1}, 1 \leq i < n.$

Efficiently enforcing the above constraint on the G_x functions forms the crux of the multipath forwarding problem.

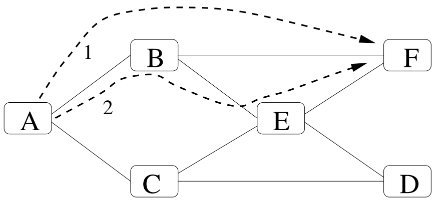
In the proposed multipath routing model, the forwarding problem can be divided into two parts, forwarding on different path services and forwarding within the same service. This division is possible because the way a packet specifies its path service is independent of how it specifies a path within that service. Thus, ψ can be represented by a fixed-length triple (Dst, ψ_1, ψ_2) where Dst is the destination address, ψ_1 specifies the path service, and ψ_2 specifies the particular path within the path service. This thesis uses this path identifier representation to uniquely identifies a particular path in a multipath network.

The following subsections use this three tuple ψ representation to address the multipath forwarding problem. The next subsection describes path forwarding on different path services (specifying ψ_1), and subsection 6.1.2 focuses on path forwarding among paths within the same service (specifying ψ_2).

6.1.1 Multi-Service Path Forwarding

Forwarding packets on different path services can be accomplished by tagging each packet with a path service identifier. A service ID is an integer that distinguishes one path service from another. Because this identifier disambiguates packets from different services, the forwarding function G can be implemented by switching on service identifiers. That is, upon receiving a packet, a router forwards the packet using the forwarding function G specified by the packet's service ID.

For example, in a multi-service single-option network, the forwarding function G for each service is the same as a single, shortest path forwarding function[†]. In this scenario, upon receiving a packet, a router simply forwards the packet to the next-hop returned by applying the function G specified by the path service.



A's forwarding table to F	
service type	next-hop
1	B
2	B

B's forwarding Table to F	
service type	next-hop
1	F
2	E

E's forwarding table to F	
service type	next-hop
1	F
2	F

Figure 6.1 : A forwarding example in a multi-service single-option multipath network.

Figure 6.1 shows selected forwarding tables of a multi-service single-option network. Here, the forwarding tables of routers A, B, and E show that each router computes two service paths to F. The dashed lines show A's two paths to F; the number above the lines show their path service number. In this setting, the forwarding function guarantees that if A sends a packet to F and tags the packet with the intended path service number, the packet will travel the intended path to F.

For example, assume that A sends a packet on path service 2 to F. This packet is than tagged with the path identifier $[F, 2]$ and forwarded, according to A's forwarding table, to node B (for simplicity, the multi-option identifier ψ_2 is omitted because there is only one path per service). Upon receiving this packet, B looks up its forwarding table for destination F with service 2 and forwards the packet to node E. E performs the same lookup function and forwards the packet to F.

Path forwarding in this scenario is guaranteed because 1) the packet's service identifier ensures that every router uses the right forwarding function (e.g. looks up the appropriate

---

[†]This example uses single-option paths because we established earlier that the single shortest path forwarding function G correctly forwards packets because the paths satisfy the Bellman optimality principle.

forwarding table entry), and 2) because one path is calculated between nodes within each service, the single path forwarding function corresponding to each service guarantees that packets are forwarded on their specified paths.

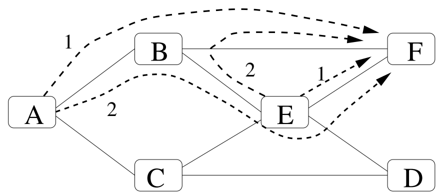
As this example shows, multi-service paths can be distinguished using a simple path service identifier. The next section shows that a similar straight-forward multi-option identifier is not sufficient to guarantee that packets are correctly forwarded on multi-option paths.

6.1.2 Multi-Option Path Forwarding

With multi-option paths, a router calculates multiple paths for the same path service. For the purpose of discussion, we assume that each multi-option path is ranked. That is, when a router computes multi-option paths to a destination, it locally assigns a unique number i to each multi-option path, indicating that the path is the i^{th} path to that destination for a particular path service. For example, the ranking of paths could reflect the i^{th} best path the router calculates within a path service.

Unlike multi-service forwarding, path forwarding for multi-option paths cannot be solved by simply tagging packets with the path's rank number. Because multi-service IDs are consistent and understood by all routers to denote a specific path service, tagging packets with a service ID unambiguously identifies a unique path service. In contrast, tagging a packet with the rank of a multi-option path, in general, does not guarantee that the packet will be forwarded on the specified path because multi-option ranks are not necessarily consistent in all routers.[‡] For example, assume that the path $(x_0, \ldots, x_n)$ is the 2^{nd} best path from x_0 to x_n. It is not guaranteed that $\forall x_i, 0 \leq i \leq n$, x_i's 2^{nd} best path is $(x_i, \ldots, x_n)$. Figure 6.2 illustrates this property.

[‡]If the multi-option IDs are consistent in all routers, then tagging packets with these IDs ensures correct path forwarding. However, making multi-option IDs consistent for a particular path calculation algorithm requires routers to know the paths computed by other routers, which increases both router computation and storage overheads. The Compute All method described in Section 6.1.3 is one such approach.



A's forwarding table to F

path number	next-hop
1	B
2	B

B's forwarding Table to F

path number	next-hop
1	F
2	E

E's forwarding table to F

path number	next-hop
1	F
2	B

Figure 6.2 : A forwarding example in a single-service multi-option multipath network.

In Figure 6.2, routers compute one path service with two multi-option paths, where the first path denotes the shortest path and the second denotes the second shortest path. The dashed lines show A and E's paths to node F. The figure demonstrates that a path number (or multi-option rank number) is not sufficient to ensure path forwarding. For example, if A wishes to send a packet on its second path to F and tags the packet with only the path number (i.e. path ID $[F, 2]$), the packet will *not* travel the intended path. To see this, after A sends the packet to B tagged with $[F, 2]$, B receives this packet and will forward the packet to B's second path's next-hop, E. E then forwards the packet on its second path, which has B as the next-hop. Notice that E *should* forward the packet on its first path (to node F). Following the example, B will then forward the packet back to E because E is the next-hop of B's second path. This results in the packet bouncing between E and B.

This example shows that because multi-option path ranks are not necessarily consistent in all routers, multi-option forwarding is not always guaranteed by simply tagging and forwarding packets based on rank numbers. To address this problem, the remainder of this chapter develops an efficient method to forward packets on multi-option paths. The next subsection describes suffix matched path sets, a crucial piece of the proposed method.

6.1.3 Suffix Matched Path Sets

This subsection presents an important class of path sets on which efficient multi-option forwarding can be performed. However, before defining this set, we first survey some existing multipath forwarding methods.

To implement the G functions in a multi-option environment, one needs a scheme for constructing path identifiers that unambiguously identify paths between nodes. One common approach, called source routing, is to use the path description itself as the identifier. In this method, path identifiers are of variable length; therefore the overhead of tagging individual packets with these path IDs increases as the size of the network grows and the path length between nodes increases. The implementation of the G functions at each node x_i for path $(x_1, \ldots, x_n)$, $1 \leq i < n$, requires reading the received packet's path ID $[x_1, \ldots, x_n]$ and then computing $\mathrm{G}_{x_i}([x_1, \ldots, x_n])$ to be x_{i+1}. No state information (e.g. forwarding table) is needed at intermediate nodes; however complete path information is needed at the sending nodes.

Although source routing is a general and flexible forwarding method, it is inefficient because of its variable length, per packet path ID: the variable length path ID increases the per packet path specification overhead, which decreases router forwarding efficiency because routers have to examine a larger packet header to determine the next-hop. In addition, source routing requires that sending nodes know the source routes of every path they wish to use, thereby increases router storage requirements. This chapter focuses on efficient path forwarding methods that use fixed-length path IDs and do not require sending nodes to know path source routes.

Another approach to multi-option forwarding is to establish a consistent set of multi-option IDs. The Compute All method is one such approach. With this approach, if K multi-option paths are maintained between source and destination pairs in an N node network, Compute All uniquely identifies a path by the triple (s, d, i), where $s, d, \in V$ denoting source and destination, and i is an integer, $1 \leq i \leq K$. To obtain these consistent IDs, each router computes, for each destination, not only its K paths, but every other router's K paths as well.

The aggregate forwarding table requirement for the Compute All method is $O(KN^2 * N) = O(KN^3)$, which is the size of the mapping from each path identifier to the next-hop (there are KN^2 paths) and is maintained by each node (there are N nodes). This requirement can be reduced by observing that a node only needs to maintain paths that pass through it. Let L be the average path length, then each path passes through L nodes on average. Therefore it suffices that each node maintains only KNL path identifiers on average, reducing the total space requirements on forwarding tables to $O(KN^2L)$. In contrast, single shortest path routing only require an aggregating forwarding tables space complexity of $O(N^2)$.

Is it possible to efficiently forward packets on multi-option paths where packets are annotated by short fixed-length path identifiers with space overhead for forwarding tables no more than $O(KN^2)$, which is K times the cost for forwarding in single shortest path systems? This question is answered in the affirmative for a interesting class of multipath sets called *suffix matched* path sets.

Suffix matched path sets: A path set P is *suffix matched* iff for all paths $(x_1, \ldots, x_n) \in P$, then $\forall i, 1 \leq i < n$, $(x_i, \ldots, x_n)$ is also in P.

Consider the path set P consisting of the single shortest paths between every pair of nodes in a network. The Bellman optimality principle ensures that P is suffix matched. If the shortest path p from node x_1 to x_n is $(x_1, \ldots, x_n)$, then for $1 < i \leq n$, the shortest path from x_i to x_n is the subpath $(x_i, \ldots, x_n)$ of p.

Proposition: Packets in an N node network with a suffix matched multipath set P can be forwarded correctly with forwarding table space $O(KN^2)$ where no more than K paths are maintained between any pair of nodes.

Proof: Let the path identifier for path $p = (x_1, \ldots, x_n)$ from x_1 to x_n in a suffix matched multipath set P be $f_{x_1}(x_n)$. Let the path identifier for the suffix paths $p_i = (x_i, \ldots, x_n)$ for $1 < i \leq n$ be $f_{x_i}(x_n)$. Recall that all suffixes of p are in P by the definition of suffix matched multipath sets.

To implement path forwarding correctly, one needs to ensure node x_i knows the suffix of its path $f_{x_i}(x_n)$ (the path $(x_{i+1}, \ldots, x_n)$) at its neighbor x_{i+1} is called $f_{x_{i+1}}(x_n)$. Therefore one needs an identifier translation or swapping function $\mathrm{F}_{x_i} : \Lambda \to \Lambda$ at each node that maps the path identifier $f_{x_i}(x_n)$ of suffix path p_i to the path identifier $f_{x_{i+1}}(x_n)$ for $1 \leq i \leq n$. That is

$$f_{x_{i+1}}(x_n) = \mathrm{F}_{x_i}(f_{x_i}(x_n)) \text{ and}$$

$$x_{i+1} = \mathrm{G}_{x_i}(f_{x_i}(x_n)).$$

In other words, when node x_i receives a packet with identifier $f_{x_i}(x_n)$ from its neighbor x_{i-1}, it forwards it to node $x_{i+1} = \mathrm{G}_{x_i}(f_{x_i}(x_n))$ along with the new identifier $f_{x+1}(x_n) = \mathrm{F}_{x_i}(f_{x_i}(x_n))$. Each x_j , $i < j < n$, performs the same two operations until the packet reaches x_n.

Notice that the definition of a suffix matched path set in a distributed environment implies that if router x_0 computes the path ($x_0, \ldots, x_n$), then router x_i must also compute the path ($x_i, \ldots, x_n$), $\forall i, 0 < i < n$. This ensures that there exists a F_{x_i} implemented by x_i that is guaranteed to translate a path identifier of x_i to x_{i+1}'s corresponding path identifier.

The space requirements for the implementation of the mapping F is proportional to the total number of paths in the network, i.e. $O(KN^2)$. This is because each path is matched with at most one other path (there are $O(KN^2)$ paths), therefore the matching table is at most $O(2KN^2)$. In addition, the space requirement for the mapping G is simply the number of paths $O(KN^2)$. Therefore multipath forwarding of suffix matched paths can be implemented with an aggregate space requirement of $O(KN^2)\ QED$.

In the transient state when suffix matched path sets are in the process of being computed at each router, correct path forwarding is not guaranteed because transient path sets are not guaranteed to be suffix matched. In fact, this absence of forwarding guarantees during route transitions affects most distributed routing algorithms, including Distance Vector and Link State based algorithms.

6.2 Distance Vector Extension

This section describes a multi-option extension to a Distance Vector (DV) based routing algorithm. Multiple services are not specifically addressed here because path forwarding between services can be accomplished with a simple path service ID. The description of the basic DV algorithm is given in Section 2.1. The next section extends the basic DV algorithm to compute multi-option paths. We then prove that the new multipath DV algorithm computes suffix matched paths sets. The proposed forwarding method is applied to the extended DV algorithm in Section 6.2.2. The computation and space complexities are also given in this section. Finally, an example of the DV suffix matched forwarding method is presented in Section 6.2.3

6.2.1 Methods of Calculating Multiple Paths in DV

The single shortest path DV algorithm can be extended to allow the calculation of multiple paths between each pair of nodes. For instance, to calculate the ranked k shortest paths to each destination, each router maintains in its forwarding table (and advertises in its DVPs) k entries for each destination. Upon receiving DVPs from its m neighbors, the router computes its own k best paths from possible $m * k$ paths received from its neighbors.

Path selection criteria other than k ranked paths, such as link or node disjoint paths, can also be implemented in a DV environment. We show that any DV-based multipath routing algorithm that satisfies the following conditions calculates suffix matched path sets.

1. A router r may install a forwarding table entry with $N_d^r = s$ only upon receiving a corresponding distance vector for destination d from its neighbor s.

2. A router r never advertises a distance vector for destination d unless it has a corresponding forwarding table entry for d.

Proposition: The path set P calculated by a multipath DV routing algorithm that satisfies conditions 1 and 2 are suffix matched.

Proof: Assume the algorithm calculates a path $(x_0, \ldots, x_n) \in P$ that is not suffix matched. Then there must exist a router x_i, $0 < i < n$ such that $(x_{i-1}, \ldots, x_n) \in P$, but $(x_i, \ldots, x_n) \notin P$. That is, x_{i-1} has a forwarding table entry with $N_d^{x_{i-1}} = x_i$, but x_i does *not* have a forwarding table entry with $N_d^{x_i} = x_{i+1}$. Thus, either x_{i-1} has installed the forwarding table entry without receiving a corresponding distance vector from x_i, or x_i has advertised a distance vector without having a corresponding forwarding table entry. But, these possibilities are ruled out by conditions 1 and 2 QED.

6.2.2 Multipath DV Extensions

In this section, we show how the proposed multipath forwarding method presented in Section 6.1 can be applied to any DV based multipath algorithms that satisfies conditions 1 and 2, and thus calculates suffix matched path sets.

The implementation of functions F and G, defined in Section 6.1, in a DV environment is defined as follows. Each router assigns an identifier ϕ for each of its paths to a given destination. Let $\phi^k_{r,d}$ be the ID router r assigns its k^{th} path to destination d. Then, the path ID used by router r in the MPDV algorithm is $pid = [d, \phi^k_{r,d}]$[§]. Adding ϕ and evaluating F and G during the packet forwarding process requires the following additions to the DV data structures.

- Forwarding table entries have two additional elements, ϕ_1 and ϕ_2. An entry in router r's forwarding table for the path $(r, s, \dots, d)$ is now a five tuple $(d, C^r_{d,\phi}, s, \phi_1, \phi_2)$. ϕ_1 is r's identifier for the path $(r, s, \dots, d)$ and ϕ_2 is neighbor s's identifier for the corresponding suffix path $(s, \dots, d)$. Note that $N^r_d = s$. $C^r_{d,\phi}$ is the cost of r's ϕ path to d.

- DVP entries have one additional element, ϕ. A DVP originating from router r that advertises r's path to destination d with identifier ϕ_1 is now a three tuple $(d, C^r_{d,\phi_1}, \phi_1)$.

Let D be the set of destinations and Φ^r_d be r's set of path identifiers to destination d. In its DVP, each router r advertises a distance vector $(d, C^r_{d,\phi_1}, \phi_1)$ corresponding to its forwarding table entry $(d, C^r_{d,\phi_1}, s, \phi_1, \phi_2)$, for each $d \in D$ and for each $\phi_1 \in \Phi^r_d$.

[§]To simplify the explanation, the path service identifier ψ_1 is omitted here. An example later in the chapter incorporates both path specifiers.

To install a route in its forwarding table upon receiving a distance vector $(d, C^s_{d,\phi}, \phi)$ from s, router r adds the entry $(d, C^r_{d,\phi_1}, s, \phi_1, \phi)$, where ϕ_1 is chosen by r to be unique in Φ^r_d; i.e. unique in its set of path identifiers to destination d. For instance, in the DV multipath algorithm for k ranked paths, r's identifier for the i^{th} path to destination d can be chosen as i. C^r_{d,ϕ_1}, the cost of this path from s to d, is calculated in the usual manner as $C^s_{d,\phi_1} + c_{rs}$.

The functions F and G can now be implemented directly via lookup of the forwarding table. For path ID $[d, \alpha]$, node r evaluates $\mathrm{F}_r([d, \alpha])$ as $[d, \beta]$ via lookup of the forwarding table entry for destination d and local identifier $\phi_1 = \alpha$, with $\beta = \phi_2$. Similarly, r evaluates $\mathrm{G}_r([d, \alpha])$ as $N^r_{d,\alpha}$ via lookup of the same forwarding table entry. A matching forwarding entry is guaranteed to exist when evaluating F and G because the path sets are suffix matched.

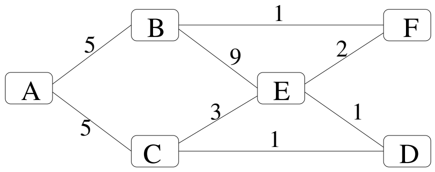
The additional storage required for the MPDV forwarding method is two path identifier fields in each forwarding table entry and one in each DVP entry. Let K be the maximal number of paths calculated to each destination, then the storage required for path identifiers is $\lceil(\log_2(K))\rceil$ bits. The additional entries in the forwarding tables and DVP for MPDV correspond to the number of alternate paths being calculated. That is, the storage overhead is a factor of $O(K)$ more than single shortest path DV. The aggregate router storage complexity of MPDV is therefore $O(KN^2)$.

6.2.3 MPDV Example

The graph in Figure 6.3 represents a network and the number above each link represents the cost of that link¶. In table below are forwarding tables of selected routers to node F. The forwarding tables are computed using the Distance Vector multipath extension algorithm. For illustrative purposes, the path calculation algorithm calculates all paths with cost less than 10. Notice that this path calculation criteria does not exclude paths that traverse a node

¶For simplicity, this example assumes that link costs are symmetric. MPDV functions correctly with non-symmetric link costs as well.

more than once. The number of paths to any destination is at most 4, ie. $K = 4$. This DV version is assumed to implement split-horizon. Below, we describe the operation of this MPDV algorithm in two phases: path calculation/forwarding table construction and packet forwarding.



E's forwarding table to F

cost	Φ_1	Φ_2	next-hop
2	1	1	F
10	2	1	B
7	3	2	D
7	4	2	C

D's forwarding Table to F

cost	Φ_1	Φ_2	next-hop
3	1	1	E
6	2	2	C
8	3	4	E

C's forwarding table to F

cost	Φ_1	Φ_2	next-hop
5	1	1	E
4	2	1	D
10	3	3	E
9	4	3	D

Figure 6.3 : Example of MPDV.

Forwarding table construction. The process of calculating paths to node F begins by F sending a DVP to its neighbors (E and B). The DVP contains the path (F, 0, 1), where F is the destination address, 0 is the cost for F to reach F, and 1 = $\phi^1_{F,F}$. When neighbor E receives this DVP, E adds the cost $C_{EF} = 2$ to every element in F's DVP. Since the path to F has cost $2 = 0 + 2$, it satisfies the path calculation criteria. E keeps this path and augments its forwarding table with the entry (F, 2, 1, 1, F): F is the destination address, 2 is the cost of this path to F, the first 1 = $\phi^1_{E,F}$ (E's name for this path to F), the second 1 = $\phi^1_{F,F}$ (node F's name for this path taken from F's DVP), and $F = N^E_{F,\phi^1_{F,F}}$ is this path's next-hop neighbor.

Next time E sends out its DVP, it will contain the path entry (F, 2, 1) advertising a path to F with cost 2 and $\phi^1_{E,F}$ of 1. When neighbor C receives this DVP containing this path to F, C augments its forwarding table with the path (F, 5, 1, 1, E) because this path is admissible under the path calculation criteria. In this example, C also chose the name of this path to be $\phi = 1$. This process of path computation continues until all routers have admissible paths to all other routers. As in the single path DV algorithm, the convergence time of MPDV is proportional to the longest path calculated between any two nodes. The complete forwarding tables to F for routers E, D, and C are shown in Figure 6.3.

Packet forwarding. Suppose node D wishes to send a packet to node F and D decides to send on its third best path, the one with cost 8. According to the algorithm, D tags the packet with the path ID [F, 4], where F is the destination address and $4 = \phi_2$ in the forwarding entry, and forwards the packet to the next-hop neighbor E. On receiving this packet, E looks up its forwarding table and matches F and ϕ_1 with the packet's path ID. The matching entry is the last entry in E's forwarding table. Once found, E replaces the packet's path ID with [F, 2] (E's ϕ_2 for this path is 2) and forwards the packet to C. That is $\mathrm{F}_E([F, 4]) = [F, 2]$ and $\mathrm{G}_E([F, 4]) = C$. C does the same lookup and forwards the packet back to D with path ID $[F, 1]$ (here $\mathrm{F}_C([F, 2]) = [F, 1]$ and $\mathrm{G}_C([F, 2]) = D$). Following the algorithm, D again performs the lookup and forwards the packet to E with the path ID [F, 1]. On receiving this packet, E finally forwards the packet to F with the ID [F, 1].

For simplicity, the MPDV path calculation algorithm presented in this example did not eliminate finite looping paths. Since this path forwarding method ensures forwarding on all calculated paths, the algorithm will forward packets on finite-looping paths (e.g. path (D, E, C, D, E, F)) as well as non-looping paths (e.g. (D, E, F)). There are well known methods of ensuring loop-free path calculation in distance vector; refer to [93] for details.

6.3 Link State Extension

This section extends Link State (LS) based routing algorithms to use the proposed multi-option forwarding method. The LS algorithm, like DV, is widely used and well understood. However, unlike DV, it relies on a centralized route calculation algorithm. A topology broadcast mechanism ensures that each router knows the current state of the entire network (i.e. topology and link cost), and routes are calculated in a centralized manner in each router. For a description of the LS algorithm, refer to Section 2.2. The next section discusses multipath extensions to the LS algorithm to compute multi-option paths, and the section following shows how the proposed forwarding method is applied.

6.3.1 Multipath LS Extensions

This section demonstrates how the proposed forwarding method applies to link state multipath algorithms that calculate suffix matched paths. Since routers in the link state algorithm have access to the entire network topology, any centralized graph algorithm for computing multiple, suffix matched paths can be used. As this section will show, the k ranked paths algorithm [45], and the initial link disjoint k paths algorithm [156] are in this class. For such algorithms, our multipath forwarding algorithm can be directly applied.

As in MPDV, let $\phi^k_{r,d}$ be the identifier that router r assigns its k^{th} path to destination d. The path ID used by router r is of the form $pid = [d, \phi^k_d]$. Forwarding table entries have two additional elements, ϕ_1 and ϕ_2. An entry in router r for the path $(r, s, \ldots, d)$ is a quadruple (d, s, ϕ_1, ϕ_2). ϕ_1 is r's identifier for the path $(r, s, \ldots, d)$ and ϕ_2 is neighbor s's identifier for the corresponding suffix path $(s, \ldots, d)$. Note that $N^r_{d,\phi_1} = s$.

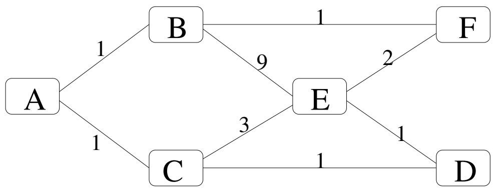
Forwarding entries are installed by the centralized link-state path calculation process. As in MPDV, ϕ_1 is chosen to be unique in r's set of paths identifiers for destination d (i.e. Φ^r_d). To determine ϕ_2 router r calculates the next-hop neighbor router s's paths set for destination d. Since the path sets are assumed to be suffix matched, s's path set to d is guaranteed to include the suffix path $(s, \ldots, d)$. Then, r sets ϕ_2 equal to s's identifier ϕ_1 for that suffix path.

As in MPDV, the functions F and G can now be implemented directly via lookup of the forwarding table. Node r evaluates $\mathrm{F}_r([d, \alpha])$ as $[d, \beta]$ via lookup of the forwarding table entry for d and local identifier $\phi_1 = \alpha$, with $\beta = \phi_2$. Similarly, r evaluates $\mathrm{G}_r([d, \alpha])$ as $N^r_{d,\alpha}$ via lookup of the same forwarding table entry.

The number of messages exchanged by MPLS routers is exactly the same as in the traditional LS algorithm. The storage overhead per router is $O(NK)$ for the forwarding tables, where N is the number of destinations and K is the number of paths to each destination. However, our multipath forwarding method requires each router to calculate each of their neighbor's K paths as well. Therefore, the suffix matched forwarding method increases the computational complexity of the multipath route calculation used in the routers by a factor

m, where m is the number of neighbors.

6.3.2 Example of LS Extension



A's forwarding Table to F

Dst	next-hop	Φ_1	Φ_2
F	B	1	1
F	C	2	2
F	C	3	3

C's forwarding table to F

Dst	next-hop	Φ_1	Φ_2
F	A	1	1
F	D	2	1
F	E	3	1

E's forwarding table to F

Dst	next-hop	Φ_1	Φ_2
F	F	1	1
F	C	2	2
F	D	3	1

Figure 6.4 : Example of the multipath Link State Algorithm.

Consider the network shown in Figure 6.4. The path calculation algorithm calculated three paths from router A to F: (A, B, F), (A, C, D, E, F), (A, C, E, F), with costs 3, 5, 6 and names $\phi^1_{A,F} = 1$, $\phi^2_{A,F} = 2$, $\phi^3_{A,F} = 3$ respectively. Suppose A sends a packet to F using its third best path. Following our algorithm, A tags the packet with the ID [F,3], where F is the destination address and 3 is the ϕ_2 for that forwarding table entry. A then forwards the packet to C, the next-hop node for this path. When C receives this packet, it searches its forwarding table to find an entry with the destination address and $\phi_1 = 3$. C's third forwarding entry matches this path ID. C then replaces the packet's old path ID with [F,1] and forwards the message to E, the next-hop element of the matching entry. That is, $\mathrm{F}_C([F, 3]) = [F, 1]$ and $\mathrm{G}_C([F, 3]) = E$. Similarly, E matches the ID [F,1] and forwards the packet with the ID [F,1] to F.

6.3.3 Suffix Matched Multipath Routing Algorithms

The previous section presented an efficient forwarding method for link state based multipath calculation algorithms that calculate suffix matching path sets. Several existing multipath algorithms generate suffix matched paths. Here we show that the k ranked path algorithm [45, 58, 141, 165] produces suffix matched paths.

Proposition: The unconstrained k shortest (ranked) simple paths algorithm [58] produces suffix matched path sets.

Proof: The unconstrained k shortest path algorithm uses the following invariant for computing multiple paths between all pairs of nodes in the network. A path of rank k between node x_1 and x_n is composed of subpaths all of whose ranks are $\leq k$. Since each node maintains k paths between itself and every other node, all subpaths of a k ranked path between x_1 and x_n are guaranteed to exist in the multipath set computed by the algorithm. QED.

The k initial link disjoint paths algorithms proposed by Topkis [156] also produces suffix matched path sets. In addition, we have already shown in Section 6.2 that any DV based multipath algorithm that satisfies two straightforward conditions produces suffix matched paths.

6.3.4 Non-Suffix Matching Paths in LS

The previous section shows that several important multipath calculation algorithms produce suffix matched paths and therefore can directly use the suffix matched forwarding method. However, there are many other path calculation algorithms, including algorithms that maximize flow between nodes, that are not guaranteed to produce suffix matched paths. Unfortunately, for these algorithms, the suffix matched forwarding presented cannot be directly applied.

To use the suffix matched forwarding method for algorithms that do not calculated suffix matched paths, one needs to convert the non-suffix matched path set into a suffix matched one. We call the conversion of a path set P into a suffixed match set as "forming a *suffix matched closure* over path set P". In general, suffix matched closures can be formed by 1) deleting paths that are not suffix matched or 2) adding paths which are suffixes of non-suffix matched paths. The solutions proposed in this section uses the latter approach. The former approach is not mentioned here because deleting paths reduces the benefits of multipath routing and does not increase the efficiency of forming suffix matched closures. The following lists possible methods to forward packets along non-suffix matched paths.

1. Lazy Compute: this method forms suffix matched closures by dynamically computing non-suffix matched paths. The algorithm works as follows: upon receiving a packet with a path ID that is not suffix matched (this is easily detected by a failure in the path ID match), a router dynamically recompute the path and forward the packet to the right next-hop. The newly computed path ID is then inserted into the forwarding table so that subsequent packet with the same ID do not need to be computed.

   The disadvantage of this method is the potential slow down in packet forwarding time for these paths because some packets require dynamic path computation. However, this disadvantage is reduced because the lazy computation occurs once per path, not once per packet on that path. The computational complexity of the lazy compute method is $O(W + pL)$, where $O(W)$ is the time needed to compute all pairs K paths, p is the number of non-suffix matched paths, L is the average path length (notice that every routers on a non-suffix matched path need to dynamically compute the path). The forwarding table storage costs of this method is $O(KN + pL)$.

2. Explicit Routing: this method is similar to lazy compute except that computation is exchanged with communication and that path IDs are of their original form (i.e. $[Dst, i]$). In this method, when a router receives a path ID that it cannot match (i.e. it is a non-suffix matched path), the router source routes the packet (notice that the router has already computed the path during its path computation phase). As the packet is source routed through the network, it establishes labels along the specified non-suffix matching path. Once the labels are established, subsequent packets that are destined on the same path are forwarded as if the path was suffix matched.

   The advantage of this solution is that it requires no extra router computation. The disadvantage is that packet header sizes need to accommodate source routes, which may complicate packet forwarding procedures. However, like lazy compute, this disadvantage is reduced by the optimization that packets on non-suffix matched paths are source routed only once per path.

3. Common Naming: the idea behind this method is for each router to first calculate a base suffix matched path set that establishes common path ID naming, and a desire set of paths are then chosen from the paths in this base set.

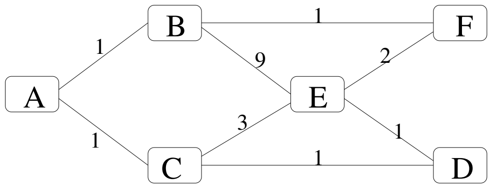
   This method works as follows: first, every router computes M shortest loop-free paths to all other nodes, $M > K$ [58, 103, 141, 165]. We call this set CM (the common naming set). This base set is guaranteed to be suffix matching. Each router then picks K paths from CM – these K paths are the ones computed by a non-suffix matched path calculation algorithm. Let the i^{th} path in K, denoted as K_i, match path j^{th} path in CM, denoted as CM_j. Now when a router forwards a packet on its path K_i, it will use the path ID CM_j. The next-hop router is guarantee to match CM_j because the set CM is suffix matched.

   The advantage of this method is that no extra routing messages are required, and nominal additional router computation are needed to establish the common naming. Moreover, unlike the lazy compute method, the path IDs can be computed *a priori*. The main disadvantage of this method is that routers need M forwarding table entries for each destination instead of K. That is, the per router space complexity of this method is $O(MN)$ instead of $O(KN)$.

The choice of methods to account for non-suffix matched paths depends on the network environment and the efficiency of different path calculation algorithms. The three different methods make different tradeoffs between router computation, routing messages, router storage, and packet overhead. Lazy compute and explicit routing are best suited for large networks where the percentage of non-suffix matching paths are low. Common naming imposes a constant CPU overhead (to calculate M shortest paths) and its efficiency is independent of the percentage of non-suffix matching paths; however, the method requires more forwarding table space.

6.4 A Multipath Forwarding Example

This section demonstrates the usage of the suffix matched forwarding method in a general multi-service multi-option network. Recall that a path identifier ψ is a triple $(Dst,\ \psi_1, \psi_2)$ consisting of the destination address, a service identifier ψ_1, and an identifier specifying a path within that service ψ_2. Section 6.1.1 showed that encoding ψ_1 requires a simple service identifier to uniquely identify different path services, and the previous two sections showed that using suffix matched forwarding, ψ_2 can be encoded as a fixed-sized integer. Notice that both ψ_1 and ψ_2 has a fixed-sized per packet overhead and requires router memory overhead proportional to the number of services and paths within each service. Therefore the path identifier $[Dst,\ \psi_1, \psi_2]$ is also fixed-sized and requires router storage overhead proportional to the number of paths.



A's forwarding Table to F

P_serv	next-hop	Φ_1	Φ_2
1	B	1	1
1	C	2	2
2	B	1	1

C's forwarding table to F

P_serv	next-hop	Φ_1	Φ_2
1	A	1	1
1	D	2	1
2	E	1	1

E's forwarding table to F

P_serv	next-hop	Φ_1	Φ_2
1	F	1	1
1	C	2	2
2	F	1	1

Figure 6.5 : Example forwarding tables in a network with multi-service and multi-option paths.

Figure 6.5 shows a six node network with link cost shown above the network edges. In this example, the network provides two path services, numbered 1 and 2. The multi-service and multi-option paths can be seen in the router forwarding tables given in Figure 6.5. The forwarding tables for A, C, and E show that the first path service (denoted by P_serv) contains two paths (denoted by the ϕ_1 column), and the second service has one path. The tables show only the forwarding entries to node F.

In this example, assume A wishes to send a packet to F on the first service's second path, which is the path (A, C, D, E, F). To do this, A tags the packet with the path ID $[F, 1, 2]$ and send it to C according to its forwarding table. On receiving this packet, C matches the destination address, path service number, and ϕ_1 to find that the next-hop for this path is D. According to the suffix matched forwarding method, C sends the packet to D with the path ID $[F, 1, 1]$. After D processes this packet, it sends the packet to E with the ID $[F, 1, 1]$. E then forwards the packet to F, thereby successfully forwarding A's packet on path (A, C, D, E, F).

Forwarding a packet on A's second service path is accomplished in the same way. The second service path from A to F is (A, B, E, F). Although the ϕ_1 and ϕ_2 entries are not necessary for single-option paths, it is shown in the Figure 6.5 to demonstrate that single-option forwarding is a special case of multi-option suffix matched forwarding. In this example, A sends a packet to B with the path ID $[F, 2, 1]$. B receives this packet and forwards to E with the ID $[F, 2, 1]$. Upon receiving this packet, E looks up its forwarding table for destination F and path service 2. According to the result of this lookup, E forwards this packet to node F with path ID $[F, 2, 1]$.

6.5 Multipath Forwarding Summary

One of the main challenges of implementing a multipath network is solving the path forwarding problem, defined as delivering packets on their intended paths. Because each router individually decides where to forward packets, ensuring that a packet travels its chosen path requires an agreement among all routers in the network. This path agreement is encoded in the form of a path ID. This chapter develops a packet encoding and forwarding method for multipath networks that uses a fixed-sized path ID and requires routing storage proportional to the number of paths calculated.

The forwarding problem is solved differently depending on whether packets are forwarded on multi-service or multi-option paths. This chapter addresses the two cases individually and combines the solution to solve the general multi-service, multi-option forwarding problem. Because paths in different services can be consistently distinguished, tagging packets with a path service identifier sufficiently distinguishes packets traveling on different services. However, specifying multi-option paths is not so straightforward because multi-option path numbers do not necessarily denote consistent paths on different routers.

To address this issue, chapter presents a novel multi-option forwarding method that uses small, fixed-length path identifiers and requires router storage proportional to the number of paths calculated. When compared to the general method for multi-option forwarding with fixed-length identifiers, the proposed method reduces the router state space needed for packet forwarding by a factor of L, where L is the average path length in the network. This chapter defines the *suffix matched* property for multipath sets and shows that many important multipath calculation algorithms produce path sets with this property. The suffix matched forwarding method presented in this chapter exploits this property.

This chapter also applies the method to two well known classes of routing algorithms—the distributed DV and the decentralized LS algorithms. For multipath extensions of DV, the storage and message overhead required is proportional to the number of extra paths calculated. In the LS multipath extension, no additional messages are needed. The additional Link State computation and storage cost for suffix matched paths are also proportional to the extra number of paths calculated. For paths calculated by non-suffix matched calculation algorithms, efficient methods to forward packets on these paths are also presented in this chapter.

Chapter 7

Multipath Transport Protocol

The three elements necessary to make multipath networks viable are 1) appropriate paths calculated between nodes, 2) efficient packet forwarding on calculated paths, and 3) effective end-host usage of multiple paths. Chapter 5 addresses the first issue by developing path calculation algorithms that provide quality paths between nodes. Chapter 6 addresses the second issue with an efficient multipath forwarding method. This chapter addresses the third issue by developing an end-host protocol that effectively uses multiple paths to increase end-to-end throughput.

A node's method of transmitting data has a large impact on network and end-to-end performance. In the current Internet, TCP is the predominant end-host transport protocol that manages the transmission of data [124]. Studies have shown that the transmission mechanisms of TCP have dramatic performance impacts on both end-to-end performance and the performance of other connections [18,27,83,119,120,123]. In a multipath scenario, the way in which nodes transmit data has an even greater performance impact because nodes not only have to decide how much data to send on each path (as in the single path environment), they also have to decide how to distribute the data over multiple paths. As Section 4.2.4 showed, the way in which multiple paths are used can result in significant performance enhancement or degradation.

This chapter develops the MPTCP transport protocol that effectively manages multiple paths to increase end-to-end throughput. MPTCP demonstrates that applications can immediately reap the benefits of multipath networks without any application modifications.

The remainder of this chapter is organized as follows. The next section addresses the issue of usage layer and presents arguments in favor of developing MPTCP. Section 7.2

describes how multiple paths should be used in order to increase throughput. Section 7.3 describes the single path TCP protocol. Then in Section 7.4, the MPTCP algorithm is presented. Simulation results that show the effectiveness of MPTCP are provided in Section 7.5. The conclusion of this chapter is that MPTCP is able to effectively use multiple paths to improve end-to-end and network-wide throughput.

7.1 Usage Layer

The multipath usage layer is the protocol layer responsible for distributing data over multiple paths. The choice of which layer should manage multiple paths depends on performance issues as well as software engineering concerns such as system compatibility, maintainability, and clean layering abstractions. There are three possible usage layers in the current TCP/IP protocol stack: network, transport, and user/application. A description of the different usage layers and our motivation for choosing the transport layer are given below.

The lowest usage layer is the *network usage layer*. Implementing multipath routing at this level means the decision of which packet should travel which path is made in the network protocol layer. In the current Internet, this implies that the IP layer must be modified to multiplex data among multiple paths. The advantage of this usage layer is that protocol layers above the network layer do not need modifications in order to use the multiple paths. However, this usage layer has two severe limitations. First, because the network layer is unaware of the demands of the user/application, the IP layer does not have the necessary information to properly distribute data onto paths that best improves application performance. Second, because higher level protocols are unaware of the multipath capabilities of the network layer, events such as out of order delivery and large delay variances may adversely affect the performance of these protocols [83].

The second possible usage layer is the *transport usage layer*. This usage layer shifts the responsibility of multipath management one level up in the protocol stack to the transport layer. The principal disadvantage of this usage layer is the required modifications to the transport and network protocols. Network protocols need to be modified to understand multiple paths to the same destination, and transport protocols need modifications to manage these paths.

The transport usage layer has three principal advantages: 1) the transport layer has the proper information to efficiently use multiple paths because it is already responsible for connection based congestion and flow control. 2) Unlike the network usage layer, the transport protocol performance will not be degraded by side-effects of using multiple paths (e.g. out-of-order deliveries and large delay variances) because the protocol itself knows that it is transmitting on different paths. 3) Since the transport layer is responsible for packet fragmentation and reassembly, it is able to use multiple paths with great flexibility.

The third possible usage layer is the *application usage layer*. This layer offers the greatest flexibility and information to achieve high performance, but requires that application developers have expert knowledge in path management such as congestion and flow control. Although there may be few specialized applications that can use this usage layer, expecting general application developers to have sufficient knowledge to effectively use multiple paths is unrealistic. As an example, one cannot assume that most application developers today can manage data transfer on a single path as good or better than TCP.

After considering the possible usage layers, we chose to develop a multipath protocol in the transport usage layer. The transport layer was chosen because it provides a simple mechanism that allows general users to obtain performance benefits from multipath networks. The protocol developed, MPTCP, maximizes end-to-end throughput. Maximizing throughput was chosen because it is a general service that can be used by many applications and is a standard measurement of network performance.

The purpose of developing the MPTCP protocol is to demonstrate that applications (end-users) can benefit from a multipath network without application changes. We envision that when multipath networks become prevalent, different transport protocols will be developed that cater to different application needs. For example, applications with QoS requirements will use a QoS transport protocol. Similarly, a transport protocol may be developed in a pricing network that minimizes the monetary cost of sending messages. Notice that because this chapter develops a multipath transport protocol, this does not preclude specialized applications from developing their own transfer protocols. In fact, we believe applications such as Web browsers may do exactly this to increase their performance.

7.2 Throughput Optimization

One of the benefits of multipath routing is that end-hosts can aggregate the resources of multiple path to increase performance. In maximizing throughput, the appropriate strategy is to use multiple paths concurrently. However, the amount of data that should be transmitted on different paths depends on the current availability of network resources.

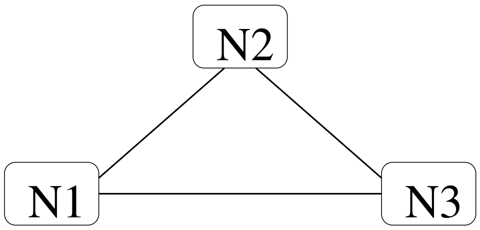


Figure 7.1 : A simple three node network. All links have equal bandwidth and latency.

For the purpose of explanation, consider a simple three node network shown in Figure 7.1 where all links have uniform bandwidth and delay and that the routing algorithm calculates two link disjoint paths between every node pair.

In this network, assume that N1 wishes to maximize its data transmission to N3. If N1 is the only node using the network, then N1 can obtain twice the throughput if it uses two paths (paths (N1, N3) and (N1, N2, N3)) than if N1 only uses the shortest path (N1,N3). Because N1 is the only node using the two paths, there is no contention for network link bandwidth, allowing N1 to fully utilize both paths.

Now consider the scenario where both N1 and N2 are maximizing their throughput to N3. Assuming that both nodes use their two paths to N3 and that the nodes share contended link bandwidth equally, then each node's individual throughput will be the same as if the nodes only used one path. The reason is that the link (N2, N3) is being used by both connections, as is link (N1, N3). Because of this contention, each node only obtains half the link bandwidth from each path.

When three nodes transmitting at the same time, one can see that if all three nodes use their two paths equally to maximize their throughput, then each node will only get 1/3 of the bandwidth from each path. Given that each node uses two paths, this means each node obtains 2/3 the throughput of a single path. Notice that if all three nodes simply use the shortest path, their effective throughput will actually be higher. The consequence of naively using multiple paths is confirmed both in Figure 4.3 in Chapter 4 and in experiments presented later in this chapter.

This simple example shows that optimizing performance is a balance between conserving network resources (e.g. using only the shortest path) and employing additional resources (e.g. using multiple paths) to surpass single path performance. First, at low levels of network utilization, the optimization point favors using multiple paths because there are plenty of underutilized network resources; thus using them allows increased throughput and does not noticeably affect other connections. However, as network utilization increases and the amount of underutilized resources decreases, the performance optimization point shifts. After this point, a strategy that conserves resources achieves better performance because the amount of additional resources used to deliver packets on non-shortest paths causes enough contention to noticeably degrade performance of other connections, while at the same time, offering little throughput gains for the transmitting node.

Thus one of the goals in designing a throughput based multipath protocol is to ensure that the protocol can adapt to this shift in optimization point. The MPTCP protocol developed in this chapter has this adaptive ability. The description of the protocol is given in succeeding sections.

7.3 TCP

Before presenting the MPTCP protocol, this section first describes the single path transport protocol, TCP, on which MPTCP is based. Transmission Control Protocol (TCP) is an end-to-end protocol that provides a reliable bit stream service between application programs [83]. That is, it ensures that the destination application receives the same data stream sent by the sending application. Furthermore, TCP aims to fully utilize the available bandwidth of the path it uses. In other words, TCP aims to maximize a connection's throughput while avoiding path congestion. Due to its efficiency and application interface, TCP has become the predominant transport protocol in today's Internet [122]. A brief description of TCP's data transfer mechanisms are given below.

In TCP, the TCP sender sends data to a TCP receiver using variable sized packets called segments. For every segment sent by TCP, a sequence number is assigned to the segment. These sequence numbers are used by the receiver to reassemble segments into their original order and to acknowledge the reception of each segment. If the receiver does not acknowledge a segment, the TCP sender will retransmit the unacknowledged segment. This acknowledgment and retransmission scheme ensures that all data transmitted are eventually received.

In addition to reliably transmitting data, TCP also provides flow control and congestion control. Flow control ensures that the sender only sends as much data as the receiver can buffer, and congestion control ensures a TCP connection never overloads a path with data. That is, congestion control ensures that the path a TCP connection is using never remains in a state of congestion. Congestion and flow control is achieved by the use of a sliding window called the congestion window. The size of the window varies to indicate the amount of unacknowledged data a sender is allowed to send without overloading the path or the receiver.

Congestion control in TCP consists of three algorithms: slow start, congestion avoidance, and fast retransmit and fast recovery. The slow start algorithm is active when the congestion window is below a certain threshold called *ssthresh*. The sender in slow start increases its congestion window by one segment each time an acknowledgment (ACK) is received. Thus in slow start, every ACK causes the sender to send two new segments: the first segment is sent to fill the congestion window made available by the ACK and the second segment by slow start's increase of the congestion window. In slow start, the size of the congestion window is doubled each round trip time (RTT). The motivation for the slow start algorithm is to quickly probe the amount of available path bandwidth.

The congestion avoidance algorithm is active when the congestion window increases beyond ssthresh. In congestion avoidance, the sender increases its congestion window by one segment every RTT. In this mode, the sender sends one new segment for every acknowledged segment, and after n segments have been received where n is the current congestion window size, the sender sends one additional segment. The goal of congestion avoidance is to slowly probe the path for additional bandwidth.

The third TCP congestion control algorithm is the fast retransmit and fast recovery mechanisms which deal with TCP's response to lost segments. In TCP, a lost segments is detected when the sender does not receive an ACK for the segment. A timeout mechanism is used to trigger the sender to send unacknowledged segments; however, the timeout mechanism typically takes a long time and thus reduces TCP performance. The fast retransmit mechanism speeds this process by retransmitting a "lost" segment after receiving three *duplicate ACKs*. A TCP receiver sends a duplicate ACK whenever it receives a segment that is not the next segment in the sequence number. Using fast retransmit, the TCP sender assumes that receiving three duplicate ACKs indicates that the oldest unacknowledged segment never reached the receiver.

The fast recovery mechanism further optimizes fast retransmit by adjusting the congestion window such that after a fast retransmit, the TCP sender is allowed to send half a congestion window's worth of new data. This optimization avoids stalling the sender while waiting for the lost segment's ACK. When the lost segment's ACK is received , the fast recovery algorithm terminates and reduces the TCP congestion window by *half*. After the termination of fast retransmit and fast recovery, the TCP connection continues transmission in congestion avoidance mode.

For more information about the TCP protocol, refer to [12, 27, 78, 83, 84, 119, 128, 168].

7.4 MPTCP

This section describes the implementation of a reliable bit stream multipath transport protocol. The protocol, called MPTCP, aims to maximizes end-to-end throughput. MPTCP requirements are that it

1. Performs congestion and flow control among multiple paths

2. Provides a reliable bit stream service (same as single path TCP)

The first requirement states that MPTCP must be sensitive to network congestion and not overrun the receiver. Congestion control allows high effective throughput even at high levels of network utilization. The second requirement states that an MPTCP receiver must receive the same bit stream sent by the MPTCP sender. In using multiple path, this implies that MPTCP need to fragment a sender's data stream, send data on multiple paths, and reconstruct the original data stream at the receiver.

The approach taken to implement MPTCP is to modify the single path TCP to use multiple paths. This approach has two main advantages. First, by extending TCP, one is able to leverage the algorithmic advancements made in TCP; thus the performance improvements made in TCP over the past decade can be directly translated into MPTCP performance. The second advantage is that it allows MPTCP to readily take advantage of future TCP improvements because MPTCP can easily upgrade whenever TCP upgrades.

7.4.1 The MPTCP Algorithm

This section describes the MPTCP algorithm. The base TCP algorithm used in MPTCP is TCP New-Reno [84]. New-Reno was chosen because of its wide use and advanced congestion control mechanisms. These mechanisms were described in the previous section.

MPTCP extends TCP very naturally. When a sender opens an MPTCP connection to a destination, MPTCP opens K concurrent and independent TCP connections to the same destination, where K is the number of paths the network provides between the sender and receiver. In MPTCP, the TCP connection establishment procedure is unaltered (e.g. a TCP three-way handshake is performed on each connection). Whenever the sender wishes to send a data stream, it passes it to MPTCP. MPTCP then divides this data stream into MPTCP segments and sequence numbers each segment*. The size of an MPTCP segment is such that it is no larger than the size of the underlying TCP segment size (i.e. TCP's minimum transmission unit) minus the length of MPTCP control information such as MPTCP sequence number and MPTCP segment size. Because the MPTCP segment size does not exceed TCP segment size, this ensures that TCP does not fragment an MPTCP segment in order to send the data, increasing MPTCP's overall efficiency. Notice that MPTCP's different sub-TCP connections may have different segment sizes; therefore the size of an MPTCP segment size varies depending on the TCP connection it is sent on.

When a destination TCP connection receives segments from its TCP sending peer, it reconstructs the received messages in the usual manner. The MPTCP receiver then reads the TCP data stream to recover MPTCP control data. Using the control data, MPTCP receiver then reconstructs the original MPTCP data stream from all its sub-TCP connections. This data stream is then returned to the receiving application. Notice that MPTCP does not need to explicitly acknowledge segments because the underlying TCP connections ensure reliable data delivery.

To make maximum use of each TCP connection, MPTCP sends on each of its TCP connection the number of segments allowed by the connection's congestion and flow control mechanisms. To do this, MPTCP provides the next MPTCP segment to a TCP connection only when the connection is ready to send a new segment of data. This dynamic load balancing allows MPTCP to fully utilize each TCP connection.

*Notice there are two levels of sequence numbers, one for MPTCP and the other for TCP. The two sequence numbers are independent.

Because MPTCP uses TCP to transmit data, it inherits TCP's congestion and flow control mechanisms. For example, when an MPTCP's sub-TCP connection detects congestion, the sub-connection will decrease its sending rate in the same manner as a normal TCP connection. Thus, MPTCP's congestion control is as good as TCP's congestion control. MPTCP performs flow control in the same way. Figure 7.2 shows an example MPTCP connection.

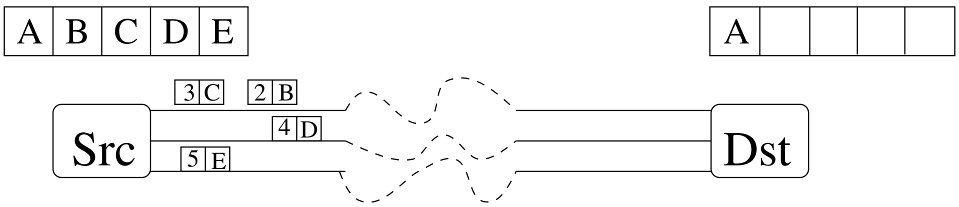


Figure 7.2 : This example shows an MPTCP connection using three paths. The original message is disassembled and sent on the three available paths using TCP. The destination node reassembles the original message using MPTCP sequence numbers.

In Figure 7.2, Src uses MPTCP to send a data stream to Dst. The figure shows that the Src MPTCP opens a TCP connection on each of the three paths provided by the network. The TCP connections are represented by the horizontal lines in the figure. The sender passes a message stream to MPTCP which are divided into five segments, labeled A through E. MPTCP sequence numbers each segment and sends them onto the different TCP connections. The boxes above each TCP connection in Figure 7.2 denote TCP frames being transmitted by that connection. The number in the box denotes the MPTCP sequence number, and the letter denotes the data segment. In this example, the MPTCP receiver has already received segment A.

7.4.2 MPTCP Fairness

In a realistic multipath network, nodes using MPTCP will compete with nodes using other protocols such as single path TCP. This scenario raises the concern that nodes using MPTCP have an unfair advantage over TCP nodes because MPTCP not only uses multiple paths but also competes for bandwidth equally with other TCP connections. This section extends the MPTCP algorithm to address this fairness concern.

Before addressing MPTCP and TCP fairness, it is worth noting that TCP itself does not provide fairness among competing TCP connections [168]. For example, in all current implementations of TCP, if two connections share a link, then the connection with a lower bandwidth-delay product will consume more bandwidth on that link. Furthermore, many researchers such as Keshav and Kalmanek advocate that in order to enforce bandwidth fairness, routers must support more intelligent queuing disciplines than Internet's current FIFO queuing [90, 113, 135]. This implies that MPTCP and TCP fairness is difficult to achieve because the underlying network and TCP implementations do not have fairness mechanisms.

In light of these factors, this section addresses the fairness concern by making MPTCP secondary connections less aggressive in competing for bandwidth. A MPTCP secondary connection is a connection that is not using the first path (primary path) in a path set. Details of our solution are given below.

Congestion Backoff Percentage

To make an MPTCP's sub-TCP connection less aggressive, we alter the TCP's congestion backoff strategy. Recall that in normal TCP, whenever the protocol senses network congestion (e.g. via unacknowledged packets), TCP reduces its congestion window by $1/2$, or a backoff percentage of 50%. To make secondary connections less aggressive, the proposal sets secondary connection's congestion backoff percentage to $X\%$, where $X \leq 50$. The primary MPTCP connection is unaltered (i.e. its congestion backoff percentage is 50%). The congestion backoff percentage also sets the connection's ssthresh to same value as the reduced congestion window size.

Notice that this method should affect a secondary connection's throughput only when congestion is encountered. That is, in the absence of contention from other connections, a secondary connection should still exploits the bandwidth of a path as before. A secondary connection's performance should differ only when competing connections cause link congestion. In this case, secondary MPTCP connections will back off more, thus giving the TCP connections using that link more share of the bandwidth.

Unfortunately in practice, this is not always true: in TCP-new Reno for example, because the protocol probes network bandwidth by causing congestion, secondary connections will back off more even though there may not be any contending connections. On the other hand, with more advanced TCP algorithms that do not probe path bandwidth through causing congestion [27], their TCP secondary connections should behave the same in the absence of congestion.

The impact of congestion backoff percentage on TCP fairness and MPTCP performance is shown in Section 7.5.

7.4.3 Path Information

The previous sections describe the MPTCP protocol; however, they did not describe how MPTCP obtains path information and how MPTCP specifies a packet's path. This section discusses these two issues.

For MPTCP to obtain information about the paths a network provides (both the type of services and the number of paths in each service), it must query its local router. The result of the queries should be a set of path IDs that the router calculates between network nodes. Querying the local router is sufficient because the router knows the type and the number of paths calculated to different destinations. How this information is exchanged and stored between hosts and local routers depend on the details of the actual network. We believe that this information can be efficiently propagated using a protocol similar to ones used in local area networks to distribute information (e.g. NIS, ARP, and DNS). In addition, because this information sharing is purely local, it does not have significant impact on the scalability, cost, or performance of multipath networks. For these reasons, the actual protocols to propagate this path information are not described in this thesis.

In order to specify a path in a multipath network, a packet needs to specify a destination address and a path identifier. The forwarding method developed in Chapter 6 requires that each packet is tagged with a path ID of the form $[Dst,\ \psi_1, \psi_2]$ where Dst is the destination address, ψ_1 specifies the path service, and ψ_2 specifies the particular path within that service. Our implementation of MPTCP uses this path ID format.

In our simulator, IP headers are augmented with a path ID field. This field is interpreted by routers to determine where to forward a packet; the forwarding procedure is given in Chapter 6. Whenever MPTCP sends a segment, it sets the segment's path ID field, thereby ensuring that the segment travels the specified path.

7.4.4 Limitations

The MPTCP protocol described in this chapter uses multiple, independent TCP connections. That is, the different TCP sub-connections operate independently and are unaware of each other. The advantage of this approach is that the MPTCP protocol makes minimal modification to TCP, which allows MPTCP to easily change and upgrade the underlying TCP protocol. The disadvantage is that the different connections are not able to work together. This section lists some of the limitations of MPTCP as well as some possible improvements.

One limitation of non-cooperative sub-connections is that a sub-TCP connection cannot reduce the retransmission duties of another connection. For example, consider an MPTCP connection that opens three TCP sub-connections and that connection 1 has the highest bandwidth path to the destination. Suppose that during data transmission, this connection's path becomes very congested and subsequently drops many of this connection's packets. In MPTCP, connection 1 is solely responsible for retransmitting all the lost segments; this may be many segments depending on the congestion window size and the severity of the congestion. In an ideal situation, however, MPTCP should be able to transmit connection 1's lost segments on different connections in order to off load the connection's retransmission duties.

Notice that *not* off loading connection 1's unacknowledged segments to other connection may not reduce the amount of segments the sender sends to the receiver because MPTCP dynamically load balances between TCP connections. That is, MPTCP will not give new data to TCP connections that are not ready to transmit new data. Despite dynamic load balancing, requiring a connection to retransmit a large number of segments forces the MPTCP receiver to buffer more segments from other connections. This results in higher receiver memory usage, increased MPTCP connection latency, and potentially lower throughput.

Another example where cooperating TCP connections can improve performance is where a connection's forward path is underutilized but the reverse path is congested. In this scenario, even though a TCP connection's forward path can transmit more segments, TCP will not send them because the ACKs are returning slowly or are being lost on the reverse path. Ideally, in this case, different sub-TCP connections should cooperate such that the receiver sends ACKs on the least congested reverse path. This way, the sender can fully exploit the bandwidth of its forward paths. However, this optimization is not possible in MPTCP because its sub-TCP connections are independent,

In short, designing a multipath transport protocol requires a tradeoff between the sophistication of the protocol and its performance. This section indicates that MPTCP can obtain higher performance if its sub-connections cooperate. However, the potential increase in performance comes at the cost of implementing cooperative mechanisms. In this thesis, we make the tradeoff toward simplicity rather than performance. Simulations show that the despite its simplicity, MPTCP is able to effectively exploit available network resources. These simulation results are given next.

7.5 MPTCP Experiments

This section experimentally evaluates MPTCP performance. Because MPTCP increases end-to-end throughput, the evaluation of MPTCP is in terms of effective throughput. In addition, fairness issues between MPTCP connections and other single path TCP connections are also addressed here.

MPTCP throughput is evaluated here not only because throughput is a common network performance metric, but also because many applications such as FTP will benefit from increased network throughput. The experiments conducted in this section examine MPTCP performance with varying network topologies and traffic patterns. The results show that MPTCP effectively increases network throughput, even under high network utilization levels.

Another issue this section addresses is the fairness of MPTCP. Since MPTCP uses more paths and competes with other single path TCP connections, it seems that TCP performance would significantly degrade when competing with other MPTCP connections. The experiments in this section evaluate the performance of TCP connections versus MPTCP connections with different congestion backoff percentages. The result of the simulations show that congestion backoff slightly improves the throughput of competing TCP connections.

The experiments in this section are organized as follows. In Section 7.5.1, the performance of MPTCP is simulated on a simple, 3 node network. Section 7.5.2 uses the same network to evaluate MPTCP fairness with respect to TCP connections. Finally, Section 7.5.3 verifies MPTCP performance and fairness in a more realistic network scenario with an Internet-like network topology and traffic pattern. The conclusion of these experiments is that MPTCP effectively utilizes available network resources and that its increased performance has nominal effects on other TCP connections.

7.5.1 Aggregate Throughput

The experiment conducted in this section measures MPTCP throughput. This experiment uses a basic three node network shown in Figure 7.1. The simple network is used to understand basic MPTCP behaviors.

In the network depicted in Figure 7.1, all links are full- duplex and have equal capacity and delay, 1000KB/s (kilobytes per second) and 10us respectively. In addition, each router can buffer up to 50 packets for each of its out-going links. The multipath routing algorithm provides two link disjoint paths between each node, the one- and two-hop paths. Each node has two clients that send a burst of packets to a random client on another node, and the interval between bursts are exponentially distributed. The average burst interval is varied to change the network utilization level. The burst size is 1000 packets and each packet is 1500 bytes long.

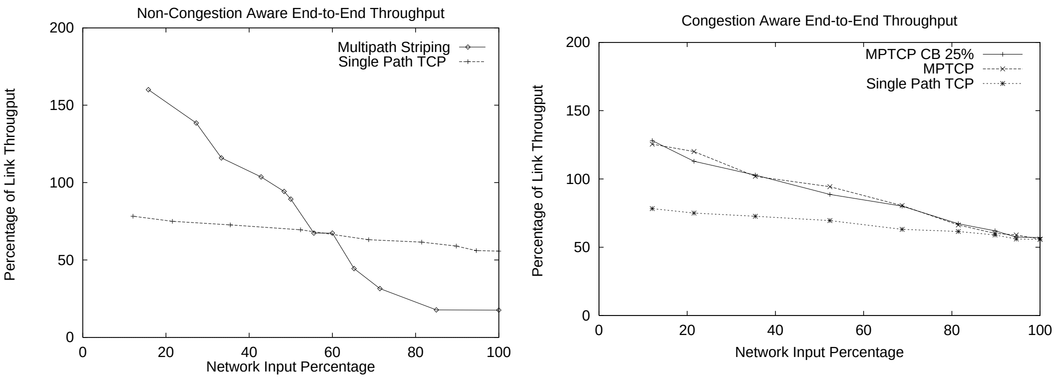


Figure 7.3 : Non-congestion aware multipath protocol versus MPTCP on a triangle network. The left figure shows the throughput percentage of a naive striping multipath protocol versus single path TCP. The right figure shows the performance of MPTCP versus single path TCP.

The graphs in Figure 7.3 features four transport protocols: TCP, naive multipath striping protocol, MPTCP without congestion backoff, and MPTCP with 25% congestion back-off percentage. TCP and the two MPTCP protocols have been described earlier. The multipath striping protocol works as follows: the multipath striping protocol clocks the sending of its data at full link capacity and distributes the data by striping them along the two available paths. That is, given N packets destined for destination D, the protocol sends packet $2i$ on the one-hop path and packet $2i + 1$ on the two-hop path to D, $0 \leq i < N/2$. Although the striping protocol does not perform congestion control, it does acknowledge received packets and retransmit lost ones.

In Figure 7.3, the y-axis represents the average end-to-end client throughput as a percentage of link capacity, and the x-axis denotes the amount of traffic injected into the network, normalized by network capacity. Thus, $x = 100\%$ denotes all nodes (there are 2 traffic generating clients in each node) simultaneously sending packets. The throughput result of each protocol is represented by their respective labeled curve. In each experiment, all clients are using the same protocol to transmit data to each other.

In both graphs, as the frequency of client transmission increases (marked by network input percentage on the x-axis), the average effective performance decreases. This is because as clients send more data, the amount network contention increases, resulting in lower end-to-end average throughput. Notice that although average end-to-end throughput decreases with increase network traffic, the *aggregate* network throughput increases because the network is more fully utilized.

The left graph in Figure 7.3 compares the performance of the naive multipath striping protocol versus TCP. This graph shows that when the network is relatively unused, the striping protocol performs better than its single path counterpart. However, when clients send data at greater than 60% of network capacity, the performance of the striping multipath protocol quickly degenerates. The reason for the degradation is that the striping protocol does not adapt to the shift in performance optimization.

As stated in Section 7.2, in order to maximize throughput in a multipath network, a protocol needs to alter its data transmission strategy in response to network traffic conditions. At low network utilization (low traffic levels), using more network resources (in terms of using more paths) increases effective throughput. However, at high network utilization, the performance optimization point shifts so that conserving resource usage achieves better performance. In the three node topology, the optimization point begins to favor conserving resources when utilization reaches 60%. At this point, a strategy that conserves resources achieves better performance. The inability to adapt to this optimization shift is indicated by the degradation of the striping protocol's performance curve.

The right graph in Figure 7.3 shows that MPTCP adapts to the optimization shift (both with and without congestion backoff). The graph shows the average end-to-end performance when the nodes are using MPTCP versus TCP. Notice that MPTCP performance is consistently better then TCP, even when nodes are transmitting all the time. This demonstrates that MPTCP is effective in shifting its transmission strategy as the performance optimization point shifts. For example, at full network utilization, MPTCP's performance gracefully degrades to the TCP performance, which is the optimal point at full utilization. This ability to gracefully degrade to TCP indicates the effectiveness of MPTCP's congestion control mechanism at exploiting available bandwidth without incurring performance degradation due to network contention.

The right graph in Figure 7.3 also shows the performance difference of non-congestion backoff MPTCP versus MPTCP with 25% congestion backoff percentage. As the performance curves show, the congestion backoff percentage has nominal affects on competing MPTCP performance.

Comparing the two graphs, the naive striping protocol performs better than MPTCP only when the network is less than 20% utilized. The reason is that the MPTCP congestion control mechanisms are conservative and thus take longer to fully utilize available path bandwidth. However, the same conservative mechanisms allow MPTCP to significantly outperform the naive protocol at high network utilization levels.

The result of this experiment is that MPTCP is able to effectively use multiple paths to increase throughput, and that even during high levels of network contention, MPTCP outperforms its single path counterpart.

7.5.2 MPTCP Congestion Backoff Percentage

The previous experiment shows that MPTCP outperforms TCP when all clients are using the same protocol. As mentioned earlier, in a realistic multipath network, nodes using MPTCP will be competing with nodes using TCP. This section examines the impact of congestion backoff percentage on MPTCP performance and the performance of competing TCP connections.

The impact of MPTCP congestion backoff percentages is shown in Figure 7.4. The curves labeled SP-x denote the throughput obtained by a TCP connection while competing with x MPTCP secondary connections (x ranges from 1 to 4). The simulations were conducted on the three node network in Figure 7.1. In this experiment, N1 establishes a

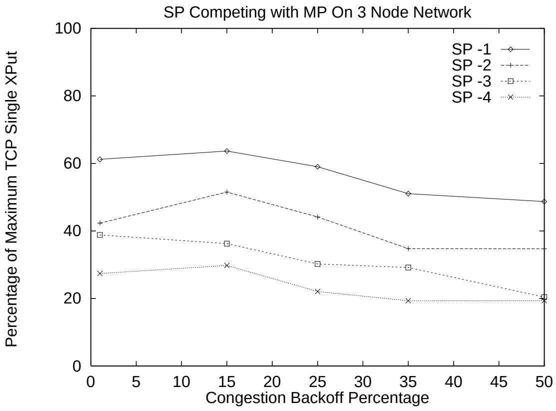


Figure 7.4 : Congestion backoff percentage on a three node network.

TCP connection and sends data to N3 while x MPTCP connections from N2 simultaneously transfer data to N3. Thus, N2's the secondary MPTCP connections compete with N1's TCP connection on link (N1, N3).

In Figure 7.4, the y-axis denotes the TCP throughput as a percentage of TCP's throughput without any competing connections. The x-axis indicates the congestion backoff percentage of the MPTCP secondary connection(s). The throughput of the secondary connections is not shown in the figure.

As the graph shows, TCP is able to obtain higher throughput when secondary MPTCP connections back-off more. However, the performance gain is not very much. For example, consider the SP-1 curve which shows a single TCP connection competing with a single MPTCP secondary connection (the top curve in Figure 7.4), at congestion backoff percentage of 25%, the TCP connection obtained 60% of the available TCP bandwidth. This is only 10% more than competing with a secondary MPTCP connection at 50% congestion backoff percentage.

Contrary to the experimental results, one expects that TCP performance would be more or less inversely proportional to the MPTCP congestion backoff percentages. Detailed analysis of TCP traces shows the cause of the observed behavior. As the congestion backoff percentage decreases, MPTCP secondary connections retract their congestion windows more whenever they sense congestion. This has the temporary effect of giving the TCP connection more of the network bandwidth. However, as the MPTCP/TCP connections resume and increase their sending rates, the network eventually drops packets. When packets are dropped, the majority of dropped packets will be from the TCP connection because this connection is sending the majority of packets on the congested link. These packet drops reduce the TCP congestion window, decreasing its throughput.

Because the probability that a connection's packets are dropped on a congested link is proportional to the number of packets the connection sends on that link, the performance gains TCP obtains from less aggressive secondary MPTCP connections are diminished. Therefore, TCP performance does not significantly exceed the performance of MPTCP secondary connections.

In spite of this phenomenon, decreasing congestion backoff percentage does give TCP connections slightly more share of the network bandwidth. This motivated us to set the MPTCP congestion backoff percentage at 25%.

7.5.3 Foreground and Background Traffic

The previous two sections evaluate the performance and fairness of MPTCP on a three node network with simple traffic patterns. The purpose of this section is to determine whether the MPTCP benefits observed in the three node network can be translated in a larger, more realistic network and traffic model. In addition, fairness is also evaluated in this section by measuring the effect of MPTCP on TCP performance. Specifically, whether MPTCP performance improvements come at the expense of other TCP connections.

To better reflect the Internet topology, this experiment uses a cluster network topology. The simulated network is composed of 195 links and 10 cluster networks, with 10 nodes per cluster. Refer to Section 9.1 on the construction of cluster networks. Every node in the network can send and receive data, and the capacity removal algorithm is used to compute paths between nodes (Chapter 6). For a more realistic traffic pattern, a combination of foreground and background traffic is used. A foreground traffic node transmits a large stream of packets (5,000 packets) to a receiving node, which is randomly chosen with the restriction that it must be in a different cluster than the sending node. A background traffic node sends a stream of packets ranging from 100 – 2,000 packets to another network node chosen at random without restrictions. After transmission, background nodes wait an exponentially distributed amount of time (a Pareto distribution with average of 20 seconds) and then proceed to transfer data to another destination node picked at random. Three of the 100 nodes are randomly chosen to transmit foreground traffic. The remaining 97 nodes generate background traffic. All nodes transmit using either MPTCP or TCP. The results of the simulations are collected when all foreground nodes finish transferring.

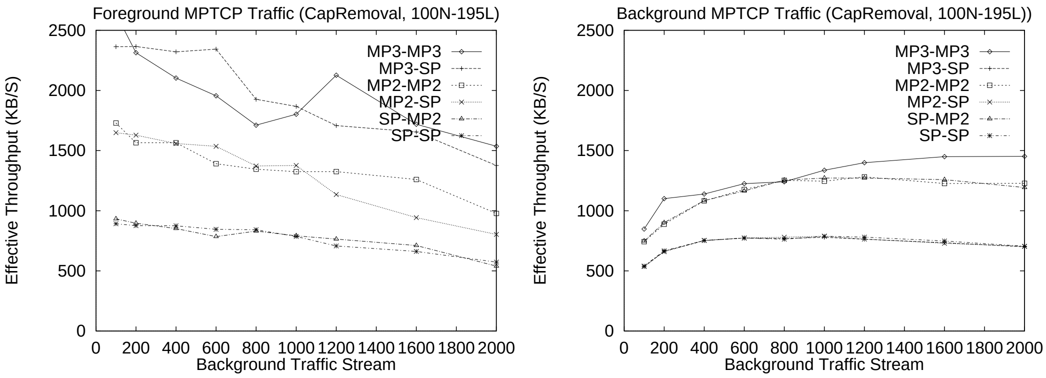


Figure 7.5 : The foreground and background performance of MPTCP using the capacity removal algorithm. The left graph shows the average foreground throughput and the right graph shows the average background throughput.

The two graphs in Figure 7.5 show the MPTCP and TCP foreground and background traffic throughput. In both graphs, the x-axis shows the burst sizes of the background traffic, which range from 100 to 2,000 packets, and the y-axis shows effective throughput in KB/s. Each curve is labeled to represent the transport protocol used by foreground and background nodes. The first label denotes the foreground transmission style, and the second denotes the background transmission style. SP stands for using TCP, and MP2 and MP3 stand for using MPTCP with 2 and 3 paths respectively.

For example, the curve with label MP3-SP denotes that the experiment was conducted where the foreground nodes transmitted using 3-path MPTCP, and the background nodes used TCP. The analysis of Figure 7.5 is divided into MPTCP performance and MPTCP's effect on TCP connections. These analyses are given below.

MPTCP Performance

The left graph in Figure 7.5 shows the foreground performance curves. The lowest two curves show the foreground throughput of TCP connections. The two curves only differ in their background transmission styles. Similarly, the middle two curves show 2-path MPTCP performance, and the two highest curves denote the 3-path MPTCP foreground performance.

Notice that as network utilization increases (marked by increasing background traffic bursts), both 2- and 3-path MPTCP gracefully degrades its performance to reflect shift in the performance optimization point. As in the three node network, this shows that MPTCP is able to effectively increase performance even at high network utilization levels. In addition, this graphs confirms two important points: 1) the capacity removal algorithm effectively offers throughput-service paths, and 2) MPTCP is able to increase throughput given additional network resources.

The right graph in Figure 7.5 shows the background throughput performance. The curves are labeled in the same manner as foreground performance graph. The bottom 3 coinciding curves denote the background TCP performance. The difference in these curves is the foreground transmission style. The middle 2 curves (SP-MP2 and MP2-MP2) show the background performance of 2-path MPTCP. These two curves are just below the MP3-MP3 curve. Notice at low background burst sizes, the background traffic performance does not achieve the same level of throughput as the foreground traffic. The reason is that the small burst sizes were not large enough for the TCP algorithms to achieve steady state. Thus, the background performance differences are not as evident as in the foreground performance curves. Despite the smaller performance distinction, the higher performance achieved by 2-path and 3-path background MPTCP connections again demonstrates the protocol's ability to increase throughput given more network resources.

In summary, the performance result in this subsection confirms that the MPTCP's throughput on the three node network translates to a larger and a more realistic network. In addition, MPTCP's ability to increase network performance when more paths are calculated demonstrates that the capacity removal algorithm successfully provides high throughput paths.

MPTCP and TCP connections

The second issue this experiment reveals is the interaction between MPTCP and TCP connections. Notice that for both the foreground and background graphs, the throughput curves are grouped by transmission modes (i.e. SP, MP2, or MP3). These tight groupings show that the performance obtained by foreground transmission mode is largely independent of background transmission mode, and vice versa. This shows that the MPTCP performance increase has nominal impact on the performance of TCP connections.

A good example is the SP-MP2 and SP-SP foreground performance curves. Here, the two foreground TCP's performance curves are almost identical while SP-MP2 background performance is consistently better than SP-SP background performance. This shows that the background MP2 and SP connections has roughly the same effect on the TCP foreground connections. Similarly, the tight grouping of background performance curves between SP-SP, MP2-SP, and MP3-SP show that the 2- and 3-path MPTCP foreground connections do not noticeably affect the performance of background TCP connections.

However, this result seems counter-intuitive. Intuitively, given that MPTCP uses more network resources to increase its throughput, it seems that this performance increase should come at the expense of the performances of other connections. The reason this intuition is flawed is that it overlooks the fact that network traffic is bursty; therefore even under heavy network usage, there are still underutilized network resources in the network. So if MPTCP mainly uses these resources to increase its performance, it will not significantly degrade the performance of other connections. The tight performance groupings in the simulation results show that most of the performance gains obtained by MPTCP are due to using otherwise underutilized resources.

The conclusion of this experiment is that 1) MPTCP successfully increases end-to-end performance and 2) that the majority of performance increase obtained by MPTCP does not come at the expense of other TCP connections.

7.5.4 Network Resources and Throughput

The previous experiments show that using capacity removal paths, MPTCP can increase network performance over TCP; however, the experiments did not show the dependency between the physical network connectivity and multipath routing's performance potential. This dependency is important because it indicates how much a given network can reap the performance benefits offered by multipath routing. The purpose of this section is to determine this dependency by evaluating the relationship between network connectivity and MPTCP performance. To this end, the experiments in this section vary two parameters: 1) the network connectivity and 2) the number of paths calculated between nodes.

The experiment conducted in this section uses a 20 node network with the number of links ranging from 40 to 120 (network connectivity from 2 to 6). The network topology is randomly generated and all links have equal bandwidth and latency. The capacity removal algorithm is used to compute paths between nodes. In the simulations, each node in the network randomly selects another node and sends a burst 5,000 packets. The inter-burst time is randomly and exponentially distributed using Pareto distribution with a 2 second average. This experiment used a large burst size to obtain steady-state throughput and small inter-burst times to keep the overall network traffic high. Figure 7.6 shows the simulation results.

The x-axis in Figure 7.6 denotes network connectivity, and the y-axis denotes throughput of MPTCP and TCP connections normalized by link capacity. Each curve shows the average throughput of all nodes using the transmission style denoted by the curve's label.

This figure shows three noteworthy results. First, 3-path MPTCP achieves the highest performance for all network connectivities, followed by 2-path MPTCP and TCP. This confirms the performance MPTCP benefits observed in the previous experiments.

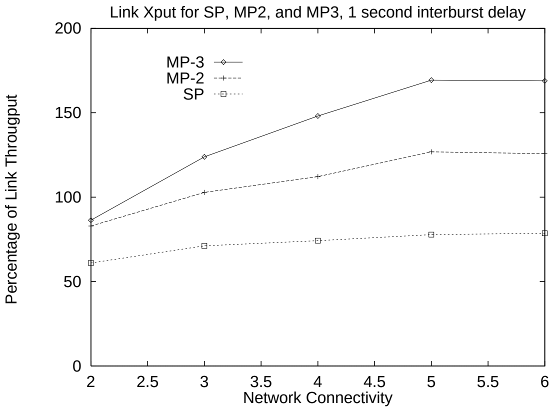


Figure 7.6 : MPTCP throughput on a 20 node network with varying network connectivity.

Second, the performance difference of using multiple paths increases as network connectivity increases. This correlation between performance increase and connectivity shows that the more connected a network, the more end-to-end path resources are available (as calculated by the path calculation algorithm), resulting in higher MPTCP throughput. As a consequence, this implies that the benefits of multipath routing are less in sparse topologies and more in dense topologies.

The third observation is that single path routing severely underutilizes the resources of a well connected network. Notice that the performance of TCP increased only slightly as connectivity increased from 2 to 6. In contrast, 2-path and 3-path MPTCP effectively utilize the available connectivity to increase throughput. Furthermore, the graph shows that 3-path MPTCP obtains higher performance gains than 2-path MPTCP.

These results show that the combination of the capacity removal path calculation algorithm and the MPTCP protocol effectively utilizes available physical network resources to increase end-to-end throughput.

7.6 MPTCP Summary

This chapter develops a transport protocol MPTCP that increases end-to-end throughput using multiple paths. MPTCP provides a reliable bit-stream service and operates by establishing multiple independent single path TCP connections. Moreover, the MPTCP sender dynamically load balances the different TCP connections by sending data on each connection according to the connection's sending rate. The MPTCP receiver then reassembles the data from its receiving TCP connections.

Extensive simulations were conducted in this chapter that demonstrate the effectiveness of MPTCP. The simulations show that MPTCP is able to effectively increase end-to-end throughput in large, Internet-like cluster networks and under both light and heavy network utilization levels. In addition, the experiments demonstrate that the performance improvements achieved by MPTCP do not necessarily come at the expense of other TCP connections, but rather, most of the performance improvements are obtained by using otherwise underutilized network resources. The conclusion of this chapter is that MPTCP effectively utilizes multiple paths to increase end-to-end throughput.

Chapter 8

Multipath Routing Algorithms

The previous three chapters presented components necessary to make multipath routing viable: algorithms that compute multiple paths, an efficient multipath forwarding method, and a multipath transport protocol that maximizes end-to-end throughput. This chapter implements two multipath routing algorithms using the developed path calculation and path forwarding components. The two multipath routing algorithms, MPDV and MPLS, based on the Distance Vector and the Link State routing algorithms respectively.

Both routing algorithms developed here use the suffix matched forwarding method and calculate capacity removal paths. Suffix matched forwarding was chosen for its efficiency, and the capacity removal paths are calculated because this path calculation algorithm highlights some key differences between DV and LS path computation styles. Moreover, since the next chapter evaluates the throughput offered by a multipath network, the capacity removal routing algorithms described in this chapter can thus be directly applied to next chapter's experiments.

The description of MPDV and MPLS are given below. Section 8.1 presents MPDV, and Section 8.2 the MPLS algorithm.

8.1 The MPDV Capacity Removal Algorithm

The Distance Vector (DV) routing algorithm is widely deployed in today's networks. Routing protocols that use this algorithm include RIP [7], BGP [133], and EPG. For a basic description of the algorithm, refer to Section 2.1. This section lists the extensions needed to enable a multipath DV algorithm to calculate capacity removal paths.

The description of MPDV is divided into two parts: the additional data structures needed to implement capacity removal MPDV and the actual path calculation algorithm.

8.1.1 Data Structure Costs

This subsection describes the additional data structures needed to convert single path DV (SPDV) to calculate capacity removal paths and to use the suffix matched forwarding method. Data structure changes are needed in MPDV forwarding tables and in MPDV Distance Vector packets (MPDVP's). MPDVP's are multipath versions of single path DVP's, and are exchanged by MPDV routers to compute paths between nodes.

MPDV Forwarding Tables

In the basic SPDV algorithm, each forwarding table entry consists of three elements: the destination address, the next-hop neighbor on the known least-cost path to the destination, and the cost of that path. Extending DV to use the suffix matched forwarding method requires adding two local path IDs to each forwarding table entry, ϕ_1 and ϕ_2 (refer to Section 6.2.2 for details).

However, adding ϕ_1 and ϕ_2 does not provide enough information for MPDV to calculate capacity removal paths. To calculate these paths, MPDV needs to add, for each forwarding table entry, a *capacity-source route* of the path that the entry advertises. A capacity-source route lists a path's links and the capacity of each link. Capacity-source routes are needed for two reasons: first, the source route component is used to eliminate looping paths, and second, the capacity of each link is used by the capacity removal algorithm to calculate paths.

MPDVPs

These additional elements in a router's forwarding table entries also increase the size of MPDVP's. In SPDV, a DVP contains a list of path entries where each entry consists of two elements: a path's destination address and the cost of the path. As given in Section 6.2.2, the suffix matched MPDV extension requires adding a local path identifier ϕ

to each MPDVP entry in order to ensure correct path forwarding. In addition to this local path ID, the capacity removal MPDVP entry also needs to carry the capacity-source route of the advertised path. This addition ensures that whenever a router receives a path from a MPDVP, the router has the necessary path information to perform capacity removal computation.

In summary, the additional data structure overhead of MPDV occurs in the routing forwarding tables and MPDVP messages. For the forwarding table, the extra router memory overhead is 1) capacity-source route and 2) two local path IDs for each path. To break down this cost, each path ID takes 1 byte and each element in a capacity-source route is 5 bytes (4 bytes for link specification and 1 for denoting link capacity). As for MPDV message overheads, two extra elements are added per MPDVP entry: 1) capacity-source route and 2) one local path ID for each advertised path.

The way these data structure additions are used to compute capacity removal paths is described next.

8.1.2 The MPDV Algorithm

Given the extra information in the MPDVP and forwarding table entries, each MPDV router calculates capacity removal paths. The procedure to establish suffix-matched local path IDs for path forwarding is not given here because it is described in Section 6.2.

The description of capacity removal MPDV starts with MPDVP construction. When a router constructs a MPDVP, it adds a MPDVP entry for every path it wants to advertise. A MPDVP entry consists of the path's destination address, cost, local path ID, and the path's capacity-source route. This information is contained in the router's forwarding table entry for that path. On receiving a MPDVP, a router updates the costs of each advertised path by adding the cost of the incoming link to the advertised paths; the incoming link is the link on which the router receives the MPDVP. In addition, the router appends the incoming link and the link's capacity to each MPDVP entry's capacity-source route. These two updates ensure that every path advertised in the MPDVP correctly reflects the cost and capacity-source route with respect to the receiving router.

After updating each advertised path, a router executes the capacity removal MPDV algorithm as follows. First, the router discards any MPDVP paths that contain loops. A path loops if it contains a link in its capacity-source route that appears more than once. After discarding looping paths, the router combines, for each destination listed in the MPDVP, the paths it already has to that destination (stored in its forwarding table) with the MPDVP paths to that destination. Now that all available paths to a destination are grouped together, the capacity removal algorithm proceeds to select, for each destination, the best K paths in that group.

Selecting K paths to destination D starts by sorting all available paths to D by path cost. We refer to P_i as the path with the i^{th} smallest cost to D and L_i as the set of links in P_i. Notice that L_i and its capacity are given in P_i's capacity-source route. Next, for each P_i, its capacity $Cap(P_i)$ is calculated as $MIN(Cap(l)), \forall l \in L_i$. If $Cap(P_i) \leq$ a preset minimum capacity threshold, then $P_i$is discarded. Otherwise P_i is admissible. If P_i is admissible, then $\forall l \in L_i, \forall j > i, \forall l' \in L_j$, if $l = l'$, then $Cap(l') = Cap(l') - Cap(P_i)$. That is, if path P_i is admissible, then subtract the capacity of P_i from all links in P_i that also appear in path $P_j, j > i$. This process starts from the least cost path to D, P_1, and ends when all paths in the group are processed or K admissible paths are found. Finally, the router installs the selected paths into its forwarding table for destination D and discards the remaining paths. This capacity removal process is repeated to find K best paths to every destination. The pseudocode of the algorithm is shown in Figure 8.1.

For each destination, the complexity of the capacity removal MPDV algorithm, as given in Figure 8.1, is linear in the number of advertised paths and the length of each path. Since there are N destinations and each router receives at most K paths per destination, the computational complexity of the MPDV algorithm is $O(NKL)$, where L is the average path length. This is a factor of KL more than SPDV, whose complexity is $O(N)$.

Notice that the paths calculated by MPDV are slightly different from the centralized capacity removal algorithm described Chapter 5. In the centralized algorithm, the i^{th} path

```
Let $A_{Dst}$ = set of advertised paths to $Dst$
Let $F_{Dst}$ = set paths to $Dst$ in router forwarding table
Let $P_i$ = $i^{th}$ least cost path in $T_{Dst}$

$T_{Dst} = A_{Dst} \cup F_{Dst}$;
$F_{Dst} = \emptyset$;
i = 1;
while(| $F_{Dst}$ |< $K$ or $T_{Dst} \neq \emptyset$)
    $T_{Dst} = T_{Dst} - P_i$;
    if ($F_{Dst} == \emptyset$ )
        Max_cost = Cost($P_i$) * Cost_BOUND;
    if (Cost($P_i$) > Max_cost)
        break;
    if (Cap($P_i$) < Capacity_Threshold)
        continue;
    $F_{Dst}$ = $F_{Dst} \cup P_i$;
    i++;
    $\forall l \in P_i, \forall P_j \in T_{Dst}$, if ($l$ == $l' \in P_j$)
        Cap($l'$) = Cap($l'$) - Cap($P_i$);
```

Figure 8.1 : The pseudocode for the MPDV capacity removal algorithm. The code shows the calculation of K capacity removal paths to Dst.

to Dst is the i^{th} shortest path to Dst with path capacity greater than the capacity threshold, where path i's capacity is calculated after subtracting path j's link capacities, $1 \leq j < i$. In contrast, with MPDV, R's i^{th} path to Dst is the i^{th} shortest path advertised by R's neighbors with path capacity greater than the capacity threshold (again, after subtracting path j's link capacities, $1 \leq j < i$). Because the MPDV algorithm calculates paths based on paths advertised by neighboring routers, the MPDV algorithm always calculates suffix matched paths. In comparison, the centralized algorithm calculates its paths regardless of the paths calculated by neighboring nodes, and as a result, the centralized algorithm does not always calculate suffix matched paths. Chapter 9 shows that this slight difference in path calculation has an impact on the number of calculated paths and on the amount of offered performance.

8.1.3 MPDV Capacity Removal Example

Figure 8.2 shows an example of MPDV's capacity removal computation process. For this example, K is 3 and the minimum capacity threshold is 0. In the figure, boxes denote routers and edges denote links, and all links have unit costs. The edges are labeled by two characters: the first letter is the link's name and the second denotes the link's capacity. For example, A-2 labels link A with 2 capacity units. The dotted lines in the figure show the paths $N5$ receives from its neighbors to node $N1$.

The MPDV paths are propagated by MPDVP exchanges: initially, $N1$ sends a MPDVP to its neighbors that contains a path to itself. When $N2$ and $N4$ receive this MPDVP, they append the incoming link's label and capacity to the MPDVP entry. Specifically, $N2$ appends link A with capacity 2 to the capacity-source route to $N1$; therefore the capacity-source route of the path is [A-2]. $N4$ does the same and the capacity-source route to $N1$ becomes [C-1].* $N2$ and $N4$ then compute paths based on the new information given by $N1$'s MPDVP and propagate the newly computed paths. This example shows that $N5$ has

*Notice that because source routes are appended, the sequence of links in a capacity-source routes is the reverse of the actual paths. This does not affect the correct functioning of the algorithms.

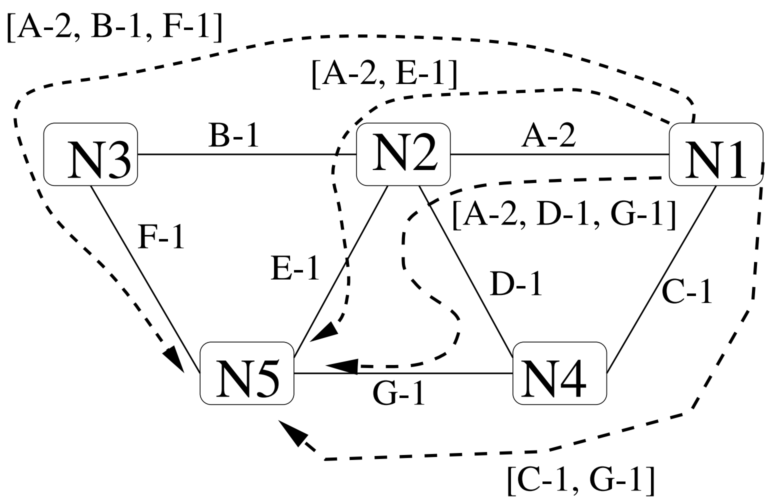


Figure 8.2 : An example of the capacity removal MPDV algorithm. The dotted lines show the paths to $N1$ that $N5$ receives from its neighbors.

received 4 paths to $N1$: (C,G), (A,E), (A,D,G), and (A,B,F). In Figure 8.2, the capacity-source route of each path is given in "[]" next to the path.

After $N5$ receives MPDVPs from its neighbors advertising these paths to $N1$, $N5$ begins the capacity removal algorithm. First $N5$ sorts the paths by cost: (G,C), (E,A), (G,D,A), (F,B,A). Next, $N5$ inspects each path in ascending order of path cost. The first path, (G,C), is admissible because $Cap(C, G) = 1 > 0$. Therefore the link capacities of C and G are subtracted by $Cap(C, G) = 1$. The path (E,A) is also admissible, and the capacities of links E and A are also subtracted by 1. Next, $N5$ considers the path (G,D,A). Here, $Cap(G, D, A) = 0$ because the capacity of link G is 0 (link G's capacity was subtracted by 1 from path (G,C)). Since $Cap(G, D, A) \leq 0$, this path is discarded. Finally, path (F,B,A) is admissible with capacity 1. With $K = 3$, $N5$ keeps paths (G,C), (E,A), and (F,B,A) in its forwarding table. Now whenever $N5$ advertises paths to $N1$, it puts these three paths into its MPDVP.

8.1.4 Capacity Removal MPDV Costs

Given the capacity removal MPDV algorithm, this subsection discusses the algorithm's costs. The MPDV costs are categorized by 1) per packet forwarding overhead, 2) router CPU usage, 3) routing message, and 4) router memory overhead.

The first and the most performance critical overhead is the per packet path forwarding overhead. This overhead consists of the additional per packet path ID cost (in the number of bits) and additional router processing needed to interpret path IDs and to forward packets. Because of the suffix matched forwarding method, MPDV's per packet overhead of path specification is constant and fixed-length. Moreover, the suffix matched path IDs are small integers, thereby allowing efficient router forwarding table lookup which decreases the per packet forwarding time [149, 160]. With low per packet forwarding overhead, MPDV decreases the time to transmit and forward data packets.

The second and third MPDV cost categories are router CPU usage and routing messages. With respect to the message complexity, the previous section showed that each MPDVP contains at most $O(NK)$ entries where each entry is length $O(L)$. Thus the message complexity of each MPDVP is $O(NKL)$, a factor of KL more than that of SPDV.

The message complexity of the entire MPDV algorithm is the product of the complexity of individual MPDVP's and the number of MPDVP's sent. Although it has been shown that the worse case number of messages for a Distance Vector style algorithm is $O(2^N)$ [25], the average case complexity is $\Theta(NM^3(ln(M))^2)$ [23,157]; here M is the average number of neighboring routers. Therefore the average message complexity of the capacity removal algorithm is $\Theta(NKL * NM^3(ln(M))^2) = \Theta(N^2M^3KL(ln(M))^2)$. Again, this is a factor of KL more than that of SPDV.

Moreover, because DV style algorithms exchange paths via messages, the computational complexity is the same as the message complexity. That is, the average MPDV computational complexity is also $\Theta(N^2M^3KL(ln(M))^2)$.

With respect to MPDV's router memory requirement, MPDV routers need memory proportional to the number of paths calculated and their length, $O(KNL)$. This is compared to

SPDV where memory and message overheads are proportional only to the number of paths calculated $O(N)$. However, it is worth noting that although MPDV requires KL additional memory, this does not mean that the forwarding tables are KL times larger (for simplicity in our previous discussion, we used the term "forwarding table overhead" synonymously with "router memory overhead"). The reason is that the capacity-source routes are not needed for packet forwarding; therefore these source routes can be stored elsewhere, say in slower and cheaper router memory. Thus, although the MPDV message overhead is KL more than SPDV, the performance critical forward table overhead is actually only a factor of K.

In summary, capacity removal MPDV overheads in router memory, router CPU, and routing messages are a factor KL more than SPDV. In addition, the performance critical cost category, per packet forwarding overhead, is low because the suffix matched forwarding method uses small integer, fixed-length path IDs, allowing efficient packet transmission and forwarding.

8.2 The MPLS Capacity Removal Algorithm

This section describes the implementation of an LS based multipath routing algorithm that calculates capacity removal paths. The LS routing algorithm is the basis of many widely used routing algorithms. Unlike DV, LS routers share topology information and each router computes paths with the knowledge of the entire network graph. As the implementation shows, this style of path computation has a large impact on the amount of resources LS needs to compute multiple paths.

8.2.1 The MPLS Algorithm

Before describing the MPLS algorithm, we briefly review of the traditional single path LS (SPLS) algorithm. In SPLS, each router periodically broadcasts its local topology in a Link State Packet (LSP). This information consists of the router ID and the cost of each of its out-going links. Routers gather these broadcasts, locally construct the entire network graph, and perform a shortest path computation to all other nodes. For each router, the result of the path computation is stored in its forwarding table.

To convert SPLS to MPLS that compute capacity removal paths is very straightforward. First, the LSP is augmented with a capacity component: an LSP now consists of the sending router's ID plus the cost and capacity of each of the router's out-going links. MPLS routers periodically broadcast their LSPs as in the SPLS algorithm. After all routers broadcast their LSPs, each router constructs the entire network graph knowing the cost and capacity of each network link. After collecting this information, routers individually execute the centralized capacity removal path calculation algorithm.

The centralized capacity removal algorithm is described in Section 5.2.2. To summarize, the algorithm finds successive shortest paths to destinations using only links that are above a capacity threshold. When the current shortest path is found to a destination, the algorithm tests if it exceeds the cost bound. If so, path computation for that destination ends. Otherwise, the path is stored in that destination's forwarding table, and the path's capacity is subtracted from each link in that path. This process continues until K paths are found or until no other admissible path exists. In this chapter's implementation, the cost bound is set at twice the cost of the shortest path, and the capacity threshold is zero.

The time complexity for the LS capacity removal algorithm to calculate K paths to a destination is $O(K * E * lg(E))$, where E is the number of edges in the network.

8.2.2 Basic MPLS Forwarding

The previous subsection described the MPLS capacity removal computation, but it did not describe the packet forwarding on the computed paths. Like MPDV, this implementation of the capacity removal MPLS algorithm uses the suffix matched forwarding method. However, unlike MPDV where establishing local path IDs is straightforward, establishing path IDs in MPLS may require additional computation and messages, depending on the type of paths computed. The basic MPLS suffix matched forwarding method is given here, and extensions to cover non-suffix matched paths are given in the next subsection.

The essence of the suffix matched forwarding method is the establishment of local path IDs between nodes. These local path IDs guarantee forwarding of packets on their intended paths. In MPLS, establishing local path IDs is a three-step process. In the first step, every router labels the i^{th} path to destination D with the path ID i, $i \leq K$. Second, for each path $(x_1, \ldots, x_n)$ that x_1 computes, x_1 finds the path ID of the path $(x_2, \ldots, x_n)$ as calculated by x_1's neighbor x_2. That is, router x_1 finds x_2's j^{th} path to x_n such that the j^{th} path is $(x_2, \ldots, x_n)$. This calculation is very simple because x_1 has the entire network topology in its memory, and the path ID assignments are deterministic (i.e. x_1 and x_2 will both calculate the path $(x_2, \ldots, x_n)$ as x_2's j^{th} path to x_n).

In the third step, x_1 matches the local path ID of the path $(x_1, \ldots, x_n)$ with x_2's path ID of the path $(x_2, \ldots, x_n)$ and stores the two path IDs in the forwarding table entry for path $(x_1, \ldots, x_n)$. Now that the two path IDs are established on this path, suffix matched path forwarding can be used. This process of calculating and matching path IDs is executed by every router for every calculated path. For more details on the MPLS suffix matched forwarding method, refer to Section 6.3.

8.2.3 Non-Suffix Matched Paths

The three step process described above correctly forwards packets if the path set is suffix matched. Unfortunately, the centralized capacity removal algorithm does not always compute suffix matched paths. This means that in step 2, when a router attempts to find the a match for a non-suffix matched path, it will discover that the path's next-hop neighbor does not calculate the suffix of that path. Fortunately, there are techniques to resolve these non-suffix matched paths; a list of techniques is provided in Section 6.3. The one used in here is the explicit routing method. We choose this method because of its efficiency and because we can explicitly measure its message and storage costs.

The explicit routing method operates as follows: during step 2 of path ID matching, if a router fails to match the suffix of a path, the router stores the source route of this non-suffix matched path. When a router receives a packet with a path ID specifying the non-suffix matched path (indicated by the source route), the router source routes the packet to its destination. Furthermore, explicit routing uses the label swapping optimization to establish local path IDs after source routing the first packet [85]. Therefore, subsequent packets traversing the same path will be forwarded using the conventional fixed-length path IDs. This optimization significantly reduces the source routing cost because it is done only *once* per non-suffix matched path. Although, these labels need to be re-established when paths are recalculated (say due to link failures), the amortized cost of source routing packets on non-suffix matched paths is small because path recomputation occurs much less frequently compared to packet forwarding. An example of the explicit routing method is given in Figure 8.3.

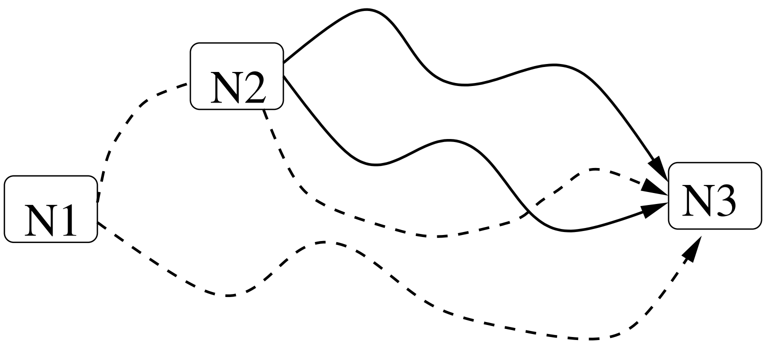


Figure 8.3 : An example of the capacity removal MPLS non-suffix matched forwarding. Here N1 computes 2 paths to N3 denoted by the dotted lines, and N2 computes the two solid paths to N3. N1's paths through N2 are not suffix matched because N2 did not compute a path that is the suffix of N1's path.

In this figure, both N1 and N2 compute two paths to N3, denoted by the dotted and solid lines respectively. Notice N1's path that passes through N2 is not suffix matched because N2 did not compute the suffix of N1's path. According to the explicit routing method, upon receiving the first packet destined on this non-suffix matched path, N1 will source route this packet on the dotted path through N2 to N3. In addition, this first packet will establish local path IDs along routers on this path so that subsequent packets destined for this path will not be source routed, but instead, use the path ID established by the source-routed packet. Thus, subsequent packets on non-suffix matched paths will be forwarded in the same manner as packets on suffix matched paths.

8.2.4 Capacity Removal MPLS Costs

For the most part, the implementation of the capacity removal MPLS algorithm is straightforward. This section summarizes the various costs incurred by the algorithm: 1) per packet forwarding overhead, 2) router CPU usage, 3) routing message, and 4) router memory overhead.

Like MPDV, the performance critical per packet forwarding overhead of MPLS is very efficient: fixed-size per packet path ID which can be efficiently indexed in router forwarding tables. However, unlike MPDV where all paths are suffix matched, some paths in capacity removal MPLS are not. For these paths, source routing is required. Since the label swapping optimization is used, source routing is incurred only once per path, and non-suffix paths need to be re-source routed only after path recomputation, which occurs infrequently compared to data packet forwarding. Thus the cost of source routing does not significantly contribute to packet forwarding complexity.

In the second cost category, router CPU overhead, the time complexity of the centralized capacity removal algorithm to compute all pairs K paths is $O(KNE * lg(E))$. Since each router also needs to calculate paths for its neighbors, the time complexity is $O(KNEM * lg(E))$, where M is the average number of neighboring routers. This is compared to shortest path LS whose complexity is $O(E*lg(E))$. Although the CPU complexity is a factor KNM more, we believe this computation overhead can be offset by leveraging the technology curve (e.g. faster processors and larger memories) [11] and developing more efficient capacity removal algorithms that use dynamic programming.

The routing message cost is divided into two categories: the extra cost in the LSPs and the costs incurred by source routing packets on non-suffix matched paths. For the first category, the cost overhead is nominal – one capacity specification per link in an LSP. In this implementation, this adds only one additional byte per link in a LSP; thus the MPLS's

LSP complexity is the same as single path LS, $O(NE)$. The second message cost category, source routing cost, depends on the number of non-suffix matched paths and the length of each path. However, the use of the label swapping optimization significantly decreases the source routing cost. Given that the source routing message cost depends on the number of non-suffix matched paths and their length, this cost is experimentally quantified in Section 9.4.

With respect to router memory, there are two types of router storage: general purpose memory and forwarding table memory. As with MPDV storage, the memory requirement for the general purpose memory is not as crucial as the forwarding memory, which is performance critical.

A MPLS router's general storage complexity is typically dominated by storing the entire topology in memory: each router needs to store every other router's LSPs, incurring $O(NM)$ memory costs. With non-suffix matched paths, an MPLS router also needs to store the non-suffix matched source routes, requiring an additional $O(NpL)$ storage, where p is the average number of non-suffix matched paths to a destination passing through a router, and L the average path length.

With respect to forwarding table storage, the MPLS cost is proportional to the number of paths calculated. However, due to non-suffix matched paths, a router may store paths that pass through the router but which the router itself does not compute. For example, router N2 in Figure 8.3 will need to store three paths, even though it only calculated two. This additional path entry comes from establishing a local path ID with N1 for N1's non-suffix matched path. Thus, the router forwarding table storage complexity for capacity removal MPLS is $O(KN + NpL)$.

Because router storage costs depend on the number of non-suffix matched paths and their length, this cost is experimentally quantified in the next chapter.

In general, the routing costs for the capacity removal MPLS routing algorithm are low, and because MPLS has the entire network topology in memory, calculating multiple paths is straightforward. However, there are costs in MPLS that cannot be analytically determined, specifically, the message cost of source routing packets and router storage costs. For these costs that depend on non-suffix matched paths, the next chapter experimentally measures their actual costs.

Chapter 9

Multipath Cost-Benefit Analysis

In this chapter, we experimentally evaluate the performance improvements obtained by using capacity removal MPDV and MPLS algorithms, and then measure each routing algorithm's message, computation, and storage costs. The goal of these measurements is to determine whether the cost incurred by a multipath routing algorithm justifies the end-to-end performance gains obtained using the provided paths.

The performance is measured in terms of throughput, latency, and message drop probability (message delivery reliability). Throughput is measured using MPTCP. The higher the MPTCP obtained throughput, the better throughput performance of the multipath network. Latency and drop probability are measured using a multipath ping program. This program uses multiple paths to measure round-trip delay and drop probabilities. To make the measurements more realistic, the ping experiments are conducted using the background traffic generated by the throughput experiments.

In addition to measuring the performance offered by a multipath network, this chapter also measures the performance (or efficiency) of the routing algorithms that implement multipath networks. The two algorithms, capacity removal MPLS and MPDV, are evaluated in this chapter in terms their packet forwarding cost, router memory requirement, router CPU usage, and routing message cost. Although many of the routing costs are analytically evaluated, some routing costs depend on the network topology and the paths computed between nodes. This chapter experimentally measures these network dependent costs, as well as the constants of the analytically formulated costs.

The results of the experiments show that even in sparse topologies (a 100 node, Internet-like cluster topologies with link to node ratio less than 2:1), the capacity removal MPDV and MPLS are able to provide paths that allow both MPTCP and multipath ping to achieve higher performance, compared to their single path counterparts. Moreover, the costs incurred by MPLS and MPDV are low enough such that we believe they can be feasibly implemented in large-scale networks.

This chapter is organized as follows. The next subsection describes the experimental environment. Section 9.2 presents MPTCP throughput, and Section 9.3 presents the latency and drop probability performance. Finally, Section 9.4 presents the various routing costs of the two routing algorithms. A summary of the experiments appears in Section 9.5.

9.1 Simulation Environment

The simulation environment consists of the network simulator and the network topology. The network simulator used is based on the "xsim" package from the University of Arizona [28], an execution-driven, packet-level network simulator. The simulator takes as input a description of the network topology, including link characteristics such as bandwidth and propagation delay, and a set of software modules that implement the various protocols running on the routers and end-hosts in the network. Simulation time advances according to the calculated transmission and propagation delay of packets in the network. In the simulator, software processing in the routers and hosts is assumed to have zero time cost.

Software processing occurs in three places in our simulation: in end-host processing, router path computation, and router packet forwarding. In each of these places, we believe this zero time cost assumption does not significantly affect our results. First, end-host processing is usually several orders of magnitude faster than end-to-end network latency; therefore accounting for their processing costs has little impact on end-to-end measured performance. Second, since router path computation is *not* performance critical, occurs infrequently compared to packet forwarding, and the computational complexity of our MPDV and MPLS implementation are based on their single path counterparts, accounting their computational time should not significantly alter our performance measurement. And third, with respect to the performance critical cost of router packet forwarding, our multipath forwarding method adds only one additional hash lookup in the forwarding process; therefore, the additional processing cost of forwarding is also very small and should not have a large impact on our simulation results.

To make the performance and cost measurements more realistic, the experiments in this chapter use an Internet-like clustered topology. Cluster topology is created by first constructing a group (or cluster) of small *flat topologies*. Flat topologies are parameterized by the number of nodes (N) and links (E). The construction process randomly picks two nodes and connects them until all E links have been connected. The only restriction on node connection is that no more than one link can exist between a node pair. After connecting E links, the network is inspected for connectedness. If the network is connected, a flat topology is generated and returned. Otherwise, the network is discarded, and the flat topology construction process is repeated.

A clustered network topology is build by inter-connecting a set of flat topologies. The flat topologies are connected by randomly connecting nodes from different flat topologies (clusters). Links connecting nodes within a cluster (intra-cluster links) have 1000 KB/s bandwidth and unit cost, and links connecting clusters (inter-cluster links) have 2000 KB/s bandwidth and two units of cost. All links have a 10us transmission delay. These parameters where chosen to reflect the higher connectivity within clusters and the larger bandwidth of inter-cluster links.

9.2 Throughput Performance

This section evaluates the throughput offered by the capacity removal MPDV and MPLS routing algorithms. Throughput is measured by end-hosts transmitting packets using MPTCP on networks running each routing algorithm. Recall Chapter 7 showed that the capacity removal algorithm effectively computes paths with additional network resources, and that the MPTCP protocol is able to translate the additional resources into increased throughput. This section extends the results of Chapter 7 and investigates whether these throughput improvements can actually be obtained using the capacity removal MPLS and MPDV routing algorithms.

9.2.1 Basic Throughput Performance

To determine whether MPTCP is able to achieve the same level of performance using MPLS and MPDV, the first experiment conducted in this section uses the same experimental parameters as the MPTCP experiment in Section 7.5.

In summary, the experimental network consists of 100 nodes organized in 10 node clusters with 17 intra-cluster links and 25 inter-cluster links (a total of 195 links).* Traffic is divided into two categories: foreground traffic and background traffic. A foreground traffic node transmits 5,000 packets to a node in another cluster, and a background traffic node sends a burst of packets ranging from 100 - 2,000 packets to a destination chosen at random. The size of each packet is 1500 bytes. The inter-burst times of background nodes are exponentially distributed (a Pareto distribution with average of 20 seconds). The three foreground nodes are picked randomly and the rest are background nodes. The performance results of MPTCP on both MPDV and MPLS are given in Figures 9.1 and 9.2.

The graphs in Figures 9.1 and 9.2 show the MPTCP foreground and background performance on MPLS and MPDV respectively. In both figures, the left graph shows the foreground performance curves, and the right graph shows background performance. The x-axis denotes background traffic burst sizes, ranging from 100 to 2,000 packets, and y-axis denotes the effective throughput in KB/s for the respective foreground or background traffic. Curves in each figure are labeled to represent the transport protocol used by foreground and background nodes: the first label denotes the foreground transmission style and the second the background transmission style. SP stands for using SPTCP, and MP2 and

---

*This network topology is particularly desirable because it is not too connected and not too sparse. The network is not too connected so that the traffic generated will produce network contention in order to show interesting properties of multipath routing, and the network is not too sparse so that meaningful multiple paths can be calculated between nodes. In short, the 1:2 node to link ratio highlights performance differences in single path and multipath routing.

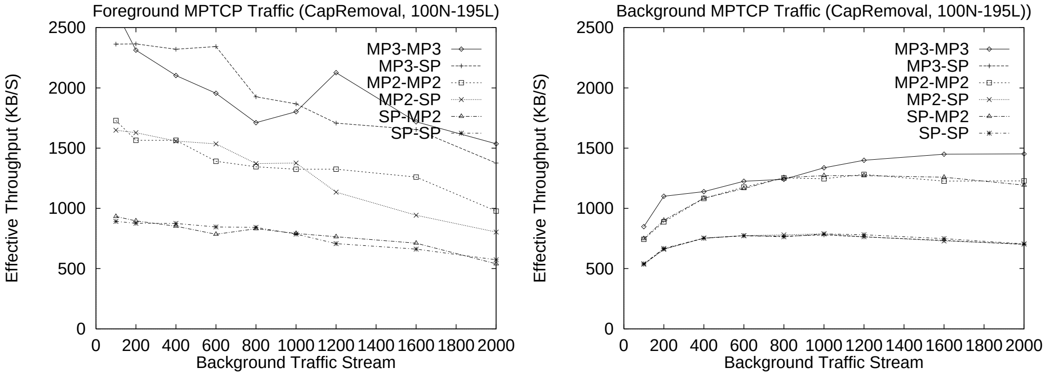


Figure 9.1 : The foreground and background MPTCP performance using the MPLS capacity removal algorithm (same as Figure 7.5).

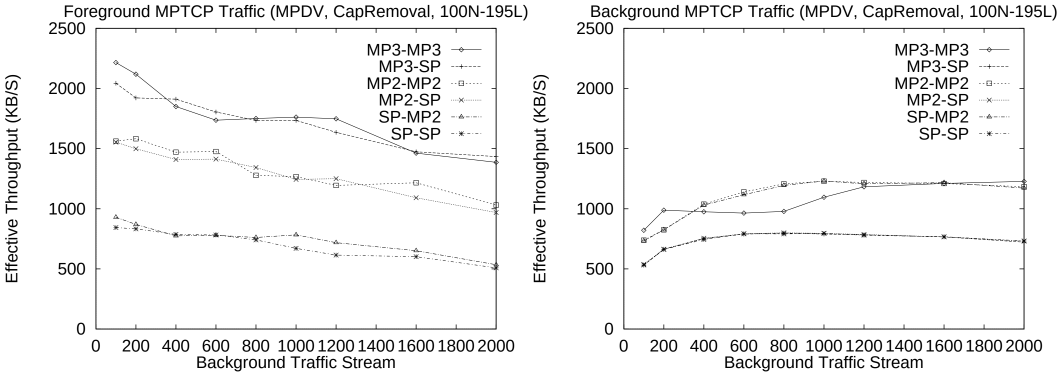


Figure 9.2 : The foreground and background MPTCP performance using MPDV capacity removal algorithm.

MP3 stand for using MPTCP with 2 and 3 paths respectively. For example, the curve labeled MP3-SP represents the network performance when the foreground nodes used 3-path MPTCP and the background nodes used SPTCP.

The graphs in Figures 9.1 and 9.2 show that foreground and background MPTCP performances exhibit the same characteristics described in Chapter 7. Mainly, MPTCP is effective in increasing network throughput even at high levels of network utilization, and the performance gains of multipath MPTCP do not come at the expense of SPTCP connections. For detailed MPTCP analysis, refer to Section 7.5. In addition to MPTCP behavior, two other important points are illustrated in the two figures.

First and foremost, both figures show that MPTCP obtains a high level of performance with both MPLS and MPDV. Moreover, the graphs show that MPTCP foreground performance is roughly proportional to the number of paths provided by each algorithm. This demonstrates that MPLS and MPDV algorithms are able to provide paths with additional resources, thereby allowing MPTCP to improve its throughput. The two figures confirm that in a realistic network setting, the combination of MPTCP and capacity removal MPDV/MPLS algorithms achieves higher network throughput than is achievable in single path routing.

The second important result in Figures 9.1 and 9.2 is that MPTCP performance is higher in MPLS than in MPDV. Performance using MPLS is the same as offered by the centralized capacity removal algorithm shown in Figure 7.5 because MPLS calculates paths using the same centralized capacity removal algorithm. In contrast, MPTCP's performance on MPDV is worse than on MPLS. The performance difference is due to the differences in MPLS and MPDV path calculation. Specifically, MPDV always calculates suffix matched paths, and MPLS does not.

In the following three subsections, we investigate the MPLS and MPDV performance disparity caused by their path calculation differences.

9.2.2 MPLS and MPDV Throughput Differences

To investigate the impact of MPLS and MPDV path computation differences, the same simulation was rerun in a sparser topology to reduce the number of available physical paths, thereby emphasizing the differences in path computation styles. This sparse network has 15 intra-cluster and 20 inter-cluster links (170 links), as compared to the original network which has 17 intra-cluster and 25 inter-cluster links (195 links). Figure 9.3 shows the MPTCP performance results in this simulation environment.

To highlight performance differences, the graph in Figure 9.3 shows only MPTCP foreground traffic performance in the sparse network. For each curve, the background nodes use the same transmission style as the foreground nodes. The graph shows the MPTCP

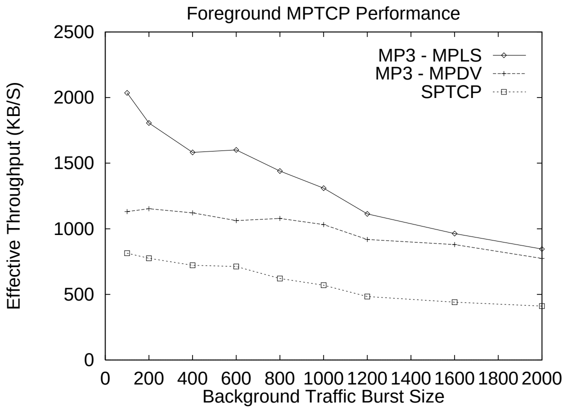


Figure 9.3 : The foreground performance of MPTCP and SPTCP using the capacity removal MPLS and MPDV algorithms on a sparse 100 node, 170 link network topology.

performance using 3-path MPLS, 3-path MPDV, and SPTCP. Three-path MPTCP is featured because using more paths accentuates the performance differences between MPLS and MPDV. The background performances are not shown because they exhibit the same performance characteristics as in Figures 9.2 and 9.1.

As with the previous topology, Figure 9.3 shows that MPTCP is able to obtain multipath performance benefits using MPDV and MPLS. This shows the generality and versatility of MPTCP and MPLS/MPDV in increasing network throughput even in a topology where the ratio of links to nodes is less than 2:1. In addition, the figure shows that MPTCP foreground performance decreases as the amount of background traffic increases. This is again consistent with our observation that as total network traffic increases, average individual throughput decreases.

The notable feature illustrated by Figure 9.3 is the MPTCP performance difference between MPDV and MPLS. MPTCP performs better under MPLS than under MPDV, especially in the region where network utilization is low. Analysis of the number of paths calculated by each routing algorithm reveals the source of this performance disparity. Figure 9.4 shows the number of paths calculated by MPLS and MPDV in this network.

In Figure 9.4, the x-axis denotes rank, the maximum number of paths the routing algorithm is allowed to calculate between a node pair, and the y-axis shows the total number

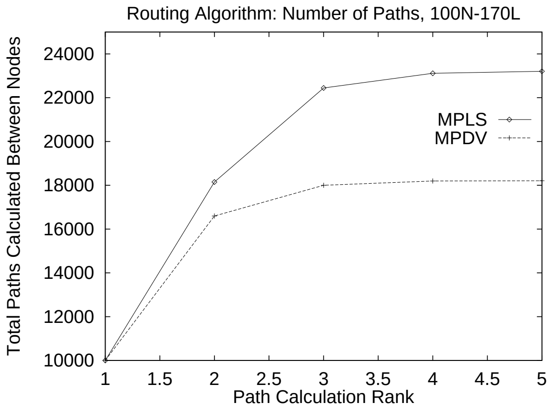


Figure 9.4 : The number of path calculated by each the MPDV and MPLS algorithms in the 170 link sparse network topology.

of paths actually calculated. At rank = 1 ($K = 1$), both algorithms calculate the single shortest paths. Since there are 100 nodes, the number of calculated path is $100^2 = 10,000$ paths.

As the figure shows, the difference between the paths calculated by MPLS and MPDV increases with rank. Moreover, this increase is sub-linear because the number of possible paths is limited by the physical connectivity of the network. The figure shows that at rank 3, MPDV calculates 18000 and MPLS calculates 22000 paths, a difference of 22%. These extra MPLS paths allow MPTCP to achieve higher throughput.

9.2.3 MPLS and MPDV Path Calculation Process

MPLS calculates more paths than MPDV because MPLS can calculate paths that are not suffix matched. In other words, MPDV is restricted in its path calculation because it always calculates suffix matched paths. Figure 9.5 shows how this restriction influences the paths that an algorithm can calculate.

In Figure 9.5, the boxes denote routers and curves denote paths. Here, capacity removal paths to N1 are being calculated. Assume that N2's shortest path to N1 is the dotted path labeled A, and that N3's shortest path to N1 is the solid path C. Assume further that rank = 2, and all links have equal capacity (i.e. calculating link disjoint paths). In the MPLS

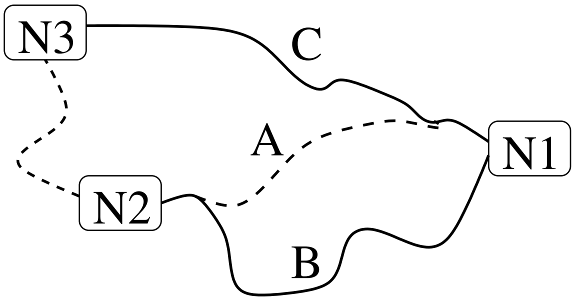


Figure 9.5 : An example of the capacity removal path calculation to $N1$.

algorithm, N3 will calculate two paths: the first path is C, and the second one passes through N2 via the dotted curve and connects to path B. Using MPDV, however, N3 will calculate only one path. This is because N2 will select and propagate only path A and discard path B, since path B is inadmissible because it has common links with path A. So when N3 receives N2's advertised path A, N3 discards A because A is inadmissible due to the links shared with path C.

The calculation of non-suffix matched paths is precisely the reason MPLS calculates more paths than MPDV. Recall, a path set P is suffix matched if and only if $\forall \alpha \in P, \alpha = (x_0, \dots, x_n)$, then $(x_i, \dots, x_n) \in P, \forall i, 0 \leq i < n$. Figure 9.6 shows the percentage of non-suffix matched paths that MPLS calculates in this sparse topology. The percentage of non-suffix matched path is the number of non-suffix matched paths divided by the total number of paths calculated.

The Figure 9.6 shows that at rank = 3, approximately 35% of the paths calculated by MPLS are non-suffix matched. Because of these paths, MPLS is able to calculate approximately 22% more paths than MPDV (Figure 9.4). These extra paths, in turn, contribute to higher MPTCP performance.

9.2.4 MPLS and MPDV Throughput Summary

This section analyzed the throughput of MPLS and MPDV capacity removal routing algorithms. The experiments show that even with relatively sparse topologies (100 nodes

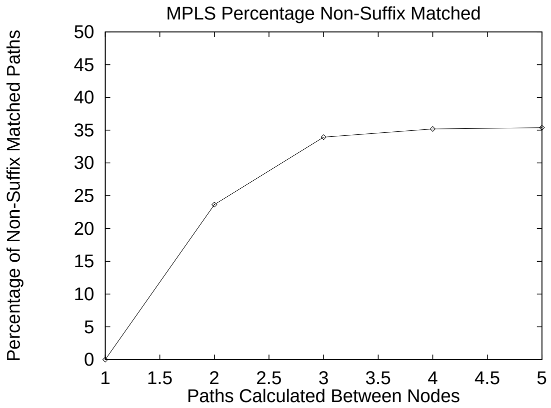


Figure 9.6 : The percentage of non-suffix matched paths calculated by MPLS in a 100 node, 170 link cluster topology.

with 195 links and 170 links), MPLS and MPDV were able to calculate paths that allow end-hosts to increase their throughput. Our experiments demonstrate that the capacity removal MPLS and MPDV algorithms, in conjunction with MPTCP, substantially increase end-to-end throughput compared to TCP using single path routing.

Further examination of MPLS and MPDV shows that under certain circumstances, MPDV does not compute as many paths as MPLS. The reason is that capacity removal MPLS can calculate non-suffix matched paths, but capacity removal MPDV cannot. This difference in path computation results in lower MPDV performance in scenarios where calculating only suffix matched paths produces a significantly lower number of paths. The performance impact of this disparity is reflected in MPTCP's achievable throughput.

9.3 Latency and Message Drop Performance

The previous section showed that multipath routing can be of immediate benefit to today's applications by offering increased network throughput. The purpose of this section is to show that multipath routing can also be used to reduce end-to-end network latency and message drop probability (or increase likelihood of message delivery), two performance metrics also of immediate benefit for today's applications.

To measure these two performance metrics, this section uses a multipath ping program that measures observed round-trip latency and message drops. Round trip latency is an important performance metric for interactive applications such as telnet and web sessions, and message drop probability is used for applications that want to increase the reliability of their message transmission.

To provide a realistic traffic scenario, the multipath ping program is run in the presence of the background traffic generated in the throughput experiments. Simulation results show that under all traffic conditions, the multipath ping program achieves lower round-trip delay and a lower number of message drops compared to a single path ping program. The conclusion of our experiments is that in addition to providing increased throughput, multipath routing *simultaneously* allows decreased round-trip delays and message drop probabilities.

9.3.1 The Multipath Ping Program

The operation of a ping program is very simple: given a destination address, the program sends a message to the destination, and upon receiving this message, the destination's ping program sends the message back to the sender. Round-trip latency is then calculated by subtracting the time the sender sends the message by the time the sender receives the reply message.

The single path ping program does exactly as described in the ping program specification: the sender sends a ping message to its destination; the destination then echos a ping upon receiving the ping message. However, because the multipath ping program can send messages of more than one path, multipath ping can selectively choose the lowest delay path to the destination, and similarly, the destination can choose the lowest delay path back to the sender. The multipath ping program developed in this section uses the minimal delay path to and from destinations.

The multipath ping program operates as follows. Given a destination address, the sender duplicates the ping message on all available paths to that destination. Upon receiving the first ping message, the destination immediately sends a reply message on all available paths back to the original sender. The destination ignores subsequent ping messages that arrive on different but slower paths. When the original sender receives a destination's reply (the first one), it calculates the round-trip time. The sender also ignores the subsequent destination reply messages that arrive on slower paths. If the sender does not receive any ping message replies, the message is counted as a dropped message.

Notice that in addition to minimizing delay, the multipath ping approach also reduces the probability that a ping is dropped by duplicating messages on multiple paths. Although this duplication approach may not be appropriate for many interactive applications (because it sends more messages), there are applications that need to send urgent messages and want these messages to have minimum transmission time and message drop probability.

For those applications that wish to lower communication delays without message duplication, they (or a low latency protocol) can periodically "ping" all available paths and then solely use the path that provides the lowest measured delay. For example, a possible low latency protocol could duplicate only retransmitted messages on multiple paths. This approach not only urgently retransmits a dropped message, but also uses this retransmitted message to find the current least delay path. The newly selected least delay path is then solely used for subsequent message delivery, until the next message drop.

For the purpose of this experiment, the multipath ping protocol described above measures the appropriate performance metrics: the reductions in round-trip latency and message drops that a multipath network can provide. The simulation results are presented next.

9.3.2 Round-trip Latency

Both the multipath and single path ping programs are implemented in our network simulator. The experimental configuration used to measure their performance is the same as the one used in the first throughput experiment: the 100 router, 195 link cluster network topology, MPTCP and SPTCP generated traffic, and the MPLS capacity removal routing algorithm (the performance results capacity removal MPDV are similar and not shown here).

In the latency experiment, each ping host sends a ping message to a randomly chosen destination. The time between a host's transmission of two consecutive pings is exponentially distributed (a Pareto distribution with average of 5 seconds). The latency and the percentage of dropped pings of single path ping (SP_ping) and multipath ping (MP_ping) are given in Figures 9.7 and 9.8.

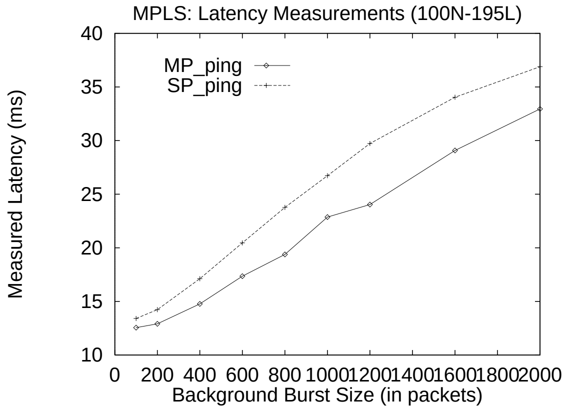


Figure 9.7 : The measured round-trip latency observed by the MP_ping and SP_ping programs. The experiment is conducted using a 100 node, 195 link cluster network.

The graph in Figure 9.7 shows two latency measurements: the latency of 3-path MP_ping on a multipath network where the background traffic uses 3-path MPTCP, and the latency of SP_ping where the background traffic uses SPTCP. In this figure, the x-axis denotes the background traffic burst sizes, and the y-axis denotes the average latency observed by the each ping program. Notice that the 3-path MPTCP generates more background traffic than SPTCP. That is, for the same background burst size (x-axis), the multipath network provides higher overall throughput than the single path network. In addition to offering higher throughput, this experiment shows that multipath routing also provides lower round-trip latency and message drop probability.

The latency graph in Figure 9.7 shows that as the background traffic increases (indicated by increasing the burst size), the round-trip latency increases correspondingly. This is expected because higher network traffic increases router queuing, which in turn increases network delay. In addition, the graph shows that as network traffic increases, the disparity in round-trip latency between MP_ping and SP_ping also increases. The reason is that at low network traffic, all paths have relatively low queuing delays, so the delay difference between the minimal delay path and the static metric shortest path is relatively small. However, at high network traffic, the delay differences among multiple paths increase; hence the increase disparity in SP_ping and MP_ping round-trip latencies.

9.3.3 Message Drop Probability

The second performance metric measured message drop probability. Message drop probability is defined as the likelihood that a transmitted message is *not* received at its destination: if a message is duplicated, then the message drop occurs when all duplicates are dropped by the network and never reaches the intended destination.

The multipath ping program is used to measure message drops. Because ping messages and their replies are sent on multiple paths, a message drop occurs only if *every* path to or from the destination drops the ping or reply ping messages. That is, assuming equal likelihood of packet drops, if the drop probability of one path is p, then drop probability of SP_ping is $2 * p$ (a message has to travel to and from the destination), and the drop probability of MP_ping is $2 * p^K$, where K is the number of paths between nodes. The drop measurements for SP_ping and MP_ping are given in Figure 9.8.

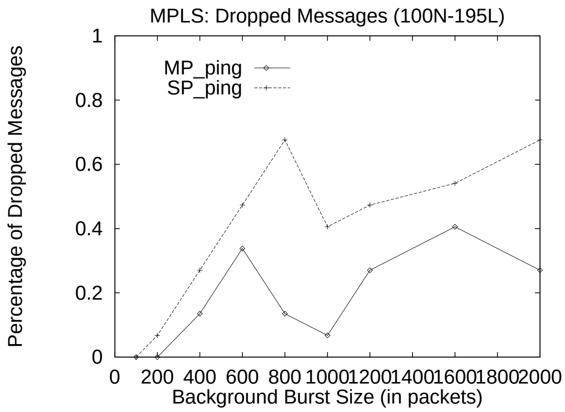


Figure 9.8 : The measured ping drop percentages of MP_ping and SP_ping. The results shown are collected in the same experiments that produced the latency measurements in Figure 9.7.

The graph in Figure 9.8 shows the percentage of dropped pings observed by SP_ping and MP_ping. The x-axis shows background traffic burst sizes, and y-axis shows the percentage of dropped ping messages. The results shown in this figure are gather in the same simulations that produced the latency measurements in Figure 9.7. The graph confirms our analysis that the MP_ping program incurs less message drops compared to SP_ping. At high network utilization levels (e.g. background burst sizes $\geq$ 600 packets), MP_ping incurs approximately half the drops compared to SP_ping.

However, given that MP_ping uses three paths, one expects the MP_ping to drop messages more seldom than 50% of SP_ping drops. One factor causing higher MP_ping drop rate is that MP_ping's background nodes (which use 3-path MPTCP) generate, according to Figure 9.1, approximately two times more traffic than SP_ping's background nodes (which use single path TCP). Because of the higher traffic rates, the probability, p, that a path will drop a packet in a multipath environment is higher than in a single path environment.

The spikes and dips of message drop percentages in Figure 9.8 are a product of random traffic generation. Because the initiation times of ping messages and background traffic use a random distribution, the spikes are the result of a combination of an unusually high traffic generation in the presence of ping measurements. Similarly, the dips are the result of the opposite scenario. The two curves have similar shape because their traffic use the same random distribution seed. However, since MPTCP nodes achieve higher throughput (thereby reducing transmission times), the background traffic sending times slightly differ between TCP and MPTCP sending nodes; thus the two curves do not have precisely the same shape.

9.3.4 Latency and Drop Probability Summary

This section shows that in addition to increasing throughput, a multipath network can also increase application performance through decreased round-trip latency and message drop probability. To quantify these performance improvements, a multipath ping program is developed that uses multiple paths to minimize round-trip latency and message drops.

Using a 100 node Internet-like cluster network and realistic MPTCP/TCP background traffic, we measure the performance of the single path (SP_ping) and multipath ping (MP_ping) programs. Simulation results show that not only do multipath networks increase throughput compared to single path networks (Section 9.2), multipath networks *simultaneously* allow the MP_ping program to achieve lower round-trip delay and lower drop probability compared to the SP_ping program on a single path network.

The conclusion of our performance experiments is that the additional resources offered by a multipath network can be effectively used to increase network performance, measured in throughput, latency, and message drop probability.

9.4 Routing Costs

The previous sections show the improvements in throughput, latency, and message drops offered by the capacity removal MPDV and MPLS routing algorithms. This section examines the overhead incurred by the two routing algorithms. The runtime cost of MPDV and MPLS, as stated in Section 4.2.3, consists of three components.

1. Packet forwarding: overhead of per packet path specification and additional router processing needed to deliver packets on multiple paths.

2. Router Storage: additional memory needed to store and support multiple paths.

3. Path computation: routing and algorithmic overhead of calculating multiple path. This refers to number of routing messages and bytes exchanged between routers and the amount of CPU needed to compute paths.

Of the three cost categories, the first is the most performance critical because packet forwarding directly influences the speed in which packets are delivered through the network. Packet forwarding consists of two components: path specification and router forwarding. First, given that every packet must specify its path, path specification must be space efficient to decrease the cost of transmitting and storing packets. Second, router's processing for each packet should be efficient in order to reduce packet forwarding time.

The second and third cost categories affect the feasibility of multipath routing. That is, maintaining low router storage and computation overhead is important because if they incur prohibitively high costs, multipath routing cannot be feasibly implemented in real networks. However, these two costs do not directly affect network performance because they do not directly influence packet forwarding speed.

The result of this section's cost analysis shows that capacity removal MPLS and MPDV efficiently implements packet forwarding. Because of the suffix matched forwarding method, data packets in both routing algorithms specify their paths using small, fixed-length path IDs. These IDs, in turn, allow fast path lookups which speed packet forwarding time. This section also shows that both routing storage and route computation costs are reasonably low such that we believe MPLS and MPDV can be feasibly implemented in current routers.

The remainder of this section presents the cost results for each category in the order presented above. The next section addresses the per packet forwarding overhead, followed by router storage and multipath computation overhead.

9.4.1 Per Packet Forwarding Overhead

This subsection examines the per packet forwarding overhead for the capacity removal MPDV and MPLS algorithms. The per packet forwarding overhead refers to 1) the amount of extra information (in bytes) that each packet needs in order to specify the path to its destination, and 2) the additional router processing needed to forward these packets on their specified paths. It is critical to minimize these two costs because they are incurred on every data packet.

Path ID Specification

The suffix matched forwarding method is used in both MPLS and MPDV implementations. This forwarding method guarantees fixed-length path IDs for suffix matched path sets. In particular, the method guarantees the number of bits needed to specify Q paths to a destination is $\lceil \log_2(Q) \rceil$ bits. In MPDV, each router has at most K paths to each destination, where K is the rank. Therefore, the per packet forwarding overhead is $\lceil \log_2(K) \rceil$ bits, and the total cost of path specification is therefore the size of the destination address plus $\lceil \log_2(K) \rceil$.

In MPLS, however, this analysis is not so straightforward. Because routers have to forward packets on non-suffix matched paths, a router may have to specify more than K paths to a destination. For a destination D, the actual number of paths a router has to distinguish to D is number of paths the router calculates to D plus the number of non-suffix matched paths to D that passes through the router (see Figure 8.3). That is, for a destination D, given that a router calculates K' paths and that p' non-suffix matched paths to D pass through the router, the path specification overhead is $\lceil \log_2(K' + p') \rceil$ bits.

This additional per packet forwarding overhead is the price MPLS pays for computing more paths. Notice that the per packet overhead is still $\lceil \log_2(Q) \rceil$ bits, where $Q = K' + p'$.

To make the number of MPLS paths more concrete, in the 170 link topology with $K = 3$, the percentage of non-suffix matched paths is 30%. Therefore on average, a router needs to specify approximately $2K$ paths to a destination (if non-suffix matched percentage is 50%, then a router needs exactly $2K$.). In addition, because each non-suffix matched path potentially needs a forwarding table entry on every router it traverses, the average non-suffix matched path a router has to support is thus $2KL$, where L is the average path length. In the 170 link topology, L is 6. It turns out that the actual number of paths a MPLS router needs to specify to a destination is much less than this upper bound. The next section, which focuses on router storage overheads, re-addresses this issue.

In both MPDV and MPLS implementations, 1 byte is used to specify the path ID. Thus every MPLS router can support up to $2^8 - K$ non-suffix matched paths to each destination. That is, given that $K = 3$, and $L = 6$, the number of non-suffix matched paths a router needs to distinguish is 36 (according to the 2KL formula above), which is much less than $2^8 = 256$. Indeed, in all our simulations, 1 byte was sufficient to uniquely identify all paths to a specific destination.

In summary, the suffix matched forwarding method used in MPLS and MPDV provides efficient guarantees on the size of per packet path IDs. The per packet overhead of path specification for both MPLS and MPDV is fixed-size and small. In MPDV, because it calculates suffix matched paths, the path ID size is precisely $\lceil \log_2(K) \rceil$, where K is the rank. In MPLS, where routers calculate more paths to destinations (via non-suffix matched paths), the cost is higher. However, in practice, a 1 byte path ID is sufficiently large enough for MPLS routers to distinguish all paths to a destination.

Packet Forwarding

Using suffix matched forwarding, the packet forwarding process consists of receiving the packet, looking up the packet's destination address and path ID to find the next-hop address, updating the packet's path ID, and then forwarding the packet to next-hop router/network. Like the per packet forwarding overhead, efficient packet forwarding is critical to network performance because it directly affects the speed at which packets are delivered to their destinations.

The packet forwarding overheads incurred by the suffix matched forwarding method are 1) the additional path ID lookup to find the next-hop address, and 2) updating the packet's path ID.

With respect to forwarding table lookup, the state-of-the art single-path IP forwarding lookup methods hash on destination addresses to retrieve the next-hop address [48, 160]. In a multipath environment, an additional path ID lookup is required. Because of suffix matched forwarding, the path ID is a small integer and therefore can be easily indexed (or hashed) once the destination's forwarding table entry is found. This requires one additional lookup and thus nominally affects routing lookup time. Notice that because hashing is used to determine next-hops, the number of extra paths calculated by a multipath routing algorithm does not have a linear affect on routing lookup time. In addition to hashing speeds, another issue is the size of the forwarding tables. The larger the forwarding table size, the slower or more expensive the memory. The next section shows that MPDV and MPLS forwarding tables are, on average, only a factor of K larger than in single path routing.

The second forwarding overhead is updating a packet's current path ID with the path ID in the forwarding table. This updating process involves writing the new path ID (stored in the router's forwarding table) into the packet's header. Since high speed network switches, such as ones in ATM [130], also perform this procedure on every data unit, we believe that the path ID update procedure can be efficiently implemented in routers as well. Moreover, there are proposed router architectures that explicitly use path ID updates to forward packets [134]. Due to these advances, we believe updating path IDs can be efficiently implemented in multipath routers and will not significantly degrade their packet forwarding efficiency.

9.4.2 Router Storage Overhead

Router storage refers to the amount of memory a router needs in order to support a routing algorithm. In high speed routers, there are two types of router memory, fast memory (e.g. SRAM) and general purpose memory (e.g. DRAM). Since packet forwarding is performance critical, fast memory is typically used to store router forwarding tables, thereby allowing fast forwarding table lookup [48]. General purpose memory is used to store all other information that a routing algorithm uses.

With respect to fast memory, it is important to keep the size of forwarding tables small because larger tables imply slower fast memory access times and/or more expensive (in terms of Dollars) router costs. However, since information stored in the general purpose memory is not performance critical and that the memory is inexpensive, the general storage memory constraint is that it needs to be reasonably low so that multipath routing algorithms can be feasibly implemented in routers.

The experiments measuring router storage overhead is thus divided into forwarding table size and general router storage size. Their measurements are given below.

Forwarding Table Size

As proven in Chapter 6, the suffix matched forwarding method guarantees that a router's forwarding table size is proportional to the number of paths the router has to all destinations. Given Q paths to each destination, a router would then need $O(NQ)$ storage for paths to all destinations. This analysis applies directly to capacity removal MPDV because every MPDV router has at most K paths to any destination; thus the storage requirement per MPDV router is $O(KN)$. This storage requirement is only a factor K more than single path DV, which requires $O(N)$ storage.

Capacity removal MPLS storage is not as simple to analyze because the centralized capacity removal algorithm can calculate non-suffix matched paths; therefore a MPLS router may have to store non-suffix matched paths that it did not calculate. In MPLS, a router needs to store, for each destination D, the paths it calculates to D plus the non-suffix matched paths to D that pass through the router. Notice that this sum is also the number of paths a MPLS router has to distinguish. From the analysis in the previous chapter, the per MPLS router forwarding table size is $O(KN + NpL)$, where p is the average number of non-suffix matched paths to a particular destination that pass through a router, and L is the average non-suffix matched path length.

However, in reality, the number of additional forwarding table space is much less because many non-suffix matched paths do not need additional forwarding table entries. Figure 9.9 shows an example.

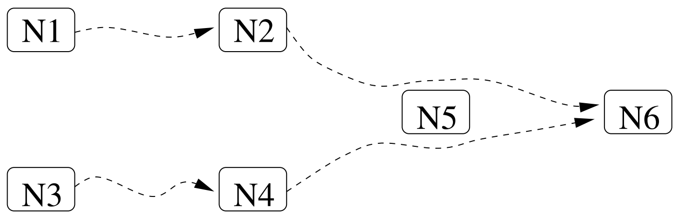


Figure 9.9 : An example of a non-suffix matched path set. Here, N5 does not need to store N1 and N3's non-suffix matched paths.

In Figure 9.9, nodes N1 - N4 computes paths to N6 that passes through N5, but N5 does not calculate the corresponding suffix of the path to N6. The figure shows that there are four non-suffixed matched paths, shown in dotted lines. When router N1 and N3 attempts to find a match for their paths to N6, they will find the correct match in N2 and N4 respectively; therefore, as far as N1 and N3 are concerned, their paths to N6 is suffix matched. In this example, N2 and N4 will fail to find matches in N5. Using the explicit routing method, N5 will add two forwarding table entries (i.e. paths) to make N2 and N4's paths suffix matched. Notice that adding these two paths in N5 automatically makes N1 and N3's paths suffix matched as well.

Due to this property, the actual number of extra forwarding table entries are much smaller than the worse case analysis. Figure 9.10 shows the actual forwarding table storage of MPDV and MPLS routing algorithms on the 100 node, 170 link sparse topology. We show the numbers for a sparse topology because for MPLS capacity removal paths, the number of non-suffix matched paths increases as network connectivity decreases.

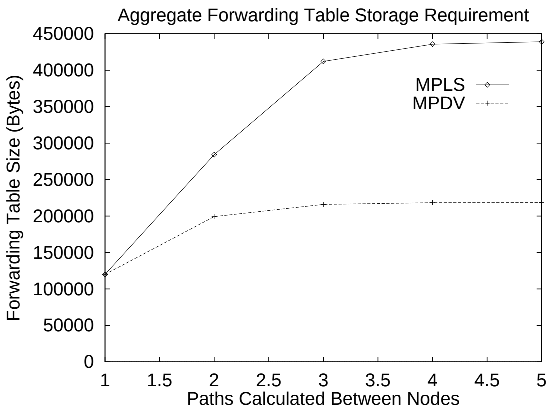


Figure 9.10 : The aggregate forwarding table storage for MPLS and MPDV in a 100 node, 170 link cluster network.

In Figure 9.10, the y-axis shows the aggregate router MPLS and MPDV storage requirement, in bytes, and the x-axis shows the rank of the routing algorithm. Each forwarding table entry is 12 bytes. Not surprisingly, the curves are heavily correlated with the number of paths each algorithm calculates (Figure 9.4). Recall from Figure 9.4, MPLS calculates 22% more paths than MPDV at rank = 3. From Figure 9.10, the extra 22% paths (non-suffix matched paths) that MPLS calculates translate into approximately two times more forwarding table space. Thus, $2NK \sim NK + NpL \Longrightarrow 2K \sim K + pL \Longrightarrow pL \sim K$. That is, experimentally, pL is approximately K.

Since multipath routers compute K paths to every destination, one expects that router forwarding tables need at least K times more storage than their single path counterparts. Indeed, our results show that MPDV adheres to this lower bound while capacity removal MPLS needs more forwarding table space for non-suffix matched paths. Our experiments show that for 100 node 170 and 190 link Internet-like networks, the storage complexity is roughly $2NK$.

To put these asymptotic complexities in perspective, according Figure 9.10, computing 3 paths between nodes requires, *per* router, approximately 4,500 bytes in MPLS and 2,000 bytes in MPDV (divide the y-axis by 100). Given that a 100 node routing domain is quite large, the forwarding table sizes measured here are realistic and applicable to many routing domains. Because the actual sizes of the forwarding tables are fairly small, we believe the amount of fast memory needed to store multipath forwarding tables are not prohibitively costly nor will the performance of the memory be significantly slow when compared to current, single path routing requirements.

General Router Storage Requirement

The second router storage category is general router memory, which stores a routing algorithm's non-performance critical information. Although not performance critical, this storage requirement nevertheless needs to be reasonably low so that multipath routing algorithms can be feasibly implemented. This section experimentally measures the general memory requirements of capacity removal MPDV and MPLS routing algorithms.

In general router memory, the information stored primarily includes the data needed for a routing algorithm to perform path computation. Thus for MPDV, a router stores information of every path it computes, which includes the path's cost, next-hop, and capacity-source route. In MPLS, a router stores the most recent LSPs the router has received from other routers and the source routes of non-suffix matched paths. From the storage analysis in Chapter 8, the per router storage complexity is $O(NKL)$ and $O(NM + NpL)$ for capacity removal MPDV and MPLS respectively. To place these complexities in perspective, Figure 9.11 shows the router aggregate general storage cost for the 170 link network.

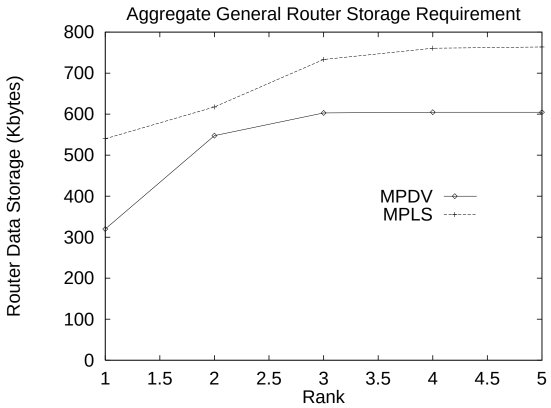


Figure 9.11 : The total general storage cost for MPLS and MPDV in the 100 node, 170 link cluster network.

In Figure 9.11, the x-axis shows the rank of the labeled routing algorithm, and the y-axis shows the aggregate general router storage (in Kbytes) that each algorithm requires. Because the MPDV cost is proportional to the number of paths calculated and the average path length, the cost curve of MPDV has the same shape as Figure 9.4, which shows the number of paths MPDV calculates. The general storage requirement for the 195 link network has the same properties as shown in Figure 9.11.

For MPLS, the general storage cost consist of storing LSPs ($O(NM)$) and the source routes for non-suffix matched paths. Since the number of non-suffix matched paths increases with rank, the MPLS curve in Figure 9.11 increases as rank increases. Although Link State based algorithms typically have to store other information such as the network graph and different data structures used to compute paths, these costs are not shown in here because they are heavily dependent on specific implementations of the path calculation algorithm.

To put these storage requirements in perspective, Figure 9.4 shows that each router needs approximately 6 and 7.3 Kbytes of general memory to store the information needed by MPDV and MPLS to calculate 3 paths per destination. Notice that these memories only measure the information needed by the *routing algorithm*. In practice, a *routing protocol*, which implements routing algorithm, would require additional memory for its internal data structures. However, since protocol costs are need for both single path and multipath routing, these costs do not affect the additional costs incurred by a multipath routing algorithm.

Given that the additional per router general storage requirement of MPDV and MPLS are relatively small (on the order of Kilobytes) in comparison to the amount memory in routers (on the order of Megabytes), we believe that current router's memory capacity can satisfy the additional storage requirements needed by a multipath routing algorithm.

9.4.3 CPU Usage in Path Computation

Router computation cost refers to the CPU cycles needed to compute multiple paths. This cost is not performance critical because path computation is typically performed on a separate processor than ones that forward packets [92]; furthermore, path computations are relatively infrequent compared to packet forwarding. Nevertheless, this cost needs to be low so that path computations can be performed in reasonable time. This subsection analyzes the computational complexities of the MPDV and MPLS capacity removal algorithms.

For the capacity removal MPLS algorithm, the computation analysis is straightforward. The algorithmic complexity of the centralized, naive capacity removal algorithm is $O(KNE * lg(E))$. In addition, since MPLS uses the suffix matched forwarding method, a MPLS router needs to compute, for each destination, paths to that destination from itself as well as from all of the router's neighbors; therefore, the naive computation complexity is $O(MKNE * lg(E)))$, where M is the average number of neighboring routers.

In MPDV, the amount of computation depends on the number of paths each router has to examine and the length of those paths. This is more than the complexity of single path DV which depends only on the number of paths calculated. The reason is that the capacity removal algorithm requires examination of each link in a path. Given that MPDV computes K times more paths and average path length is L, the computation complexity is $O(KL)$ more than SPDV. Given that the average computational complexity of SPDV is $\Theta(N^2M^3(ln(M))^2)$ (Section 8.1), the average complexity of the capacity removal MPDV algorithm is thus $\Theta(KL * N^2M^3(ln(M))^2)$.

We believe optimization such as dynamic programming for MPLS and efficient path encoding for MPDV can reduce the CPU overhead of capacity removal path computation. However, despite the lack of these optimizations, the complexities of the two algorithms are reasonably low; therefore, we believe that they can be feasibly implemented in current routers.

9.4.4 Routing Message Cost

Routing message cost refers to the number of messages needed to propagate information in order to perform route computations. Because our multipath implementations use static metrics, routing message cost is incurred only during topology changes, which occurs relatively infrequently compared to data forwarding. Nevertheless, this cost cannot be prohibitively high because it may affect data forwarding speed (Chapter 3). This subsection analyzes the message cost of MPLS and MPDV algorithms.

As stated in Section 8.1, the average message complexity of capacity removal MPDV is $\Theta(N^2M^3KL(ln(M))^2)$; a factor KL more than the SPDV. This complexity is derived from the product of the size of each MPDVP and the number of MPDVP's transmitted: the size of each MPDVP is $O(NKL)$, and the average number of MPDVP's transmissions is $\Theta(NM^3(ln(M))^2)$.

For MPLS, the routing message overhead comes in two categories. The first category is the messages needed to disseminate topology information. The requirement is the same as the single path LS algorithm, which is $O(NE)$. Notice that compared to SPLS, the MPLS topology messages require an additional capacity specification per link (1 byte); this additional cost does not affect the MPLS message complexity.

The second MPLS message overhead is the cost needed to support non-suffix matched paths. Our implementation of MPLS uses the explicit routing method that source routes non-suffix matched paths. This method does not require any additional computation, but incurs message cost proportional to the number of non-suffix matched paths and their lengths. Thus, the explicit routing message overhead is $O(NpL)$.

However, since the number of suffix matched paths is topology dependent and not all non-suffix matched path need to be source routed (illustrated in Figure 9.9), the analytical bound does not provide a concrete bound. Therefore, we experimentally quantified this cost. Given that message costs increase as network connectivity increases (refer to message complexity analysis), Figure 9.12 shows the routing message cost for MPLS and MPDV in the 195 link topology. The message costs for the 170 link topology exhibit similar characteristics and are not shown here.

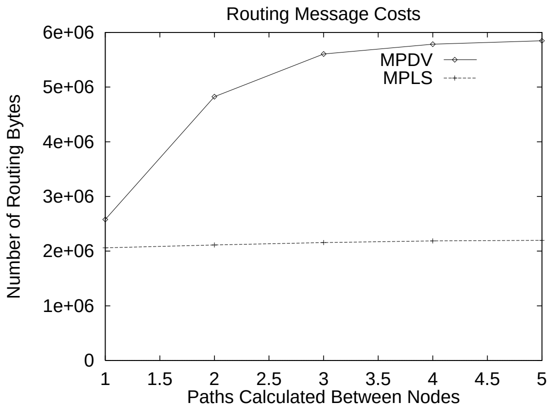


Figure 9.12 : The routing message cost for capacity removal MPLS and MPDV algorithms. The message costs shown are the total number of bytes each routing algorithm transmits for path computation. MPLS uses the explicit routing technique to resolve non-suffix matched paths.

The x-axis in Figure 9.12 denotes the rank of each algorithm, and the y-axis shows the routing message cost (in bytes) needed to compute paths between nodes.

As Figure 9.12 shows, the MPLS message costs increase slightly with the number of paths calculated between nodes. At rank = 1, the MPLS curve shows the cost of LSP broadcasts, which is constant regardless of the rank value. Notice that the MPLS message cost increases slightly with rank. This increase is attributed to the message cost of explicit routing of non-suffix matched paths. The graph shows that the maximum message overhead incurred by source routing (at rank = 5) is approximately 10% of the cost to broadcast LSPs. This means the explicit routing message cost $O(NpL)$ is approximately 10% of the LSP broadcast cost.

For MPDV, notice that the message cost of computing 2 paths per node is approximately the 2 times more than computing one path; however, the cost of computing 3, 4, and 5 paths are only slightly more than computing 2 paths. The reason is that the message cost of MPDV is proportional to the number of paths actually computed. Notice that the curve in Figure 9.4, which shows the number paths MPDV calculates, has the same shape as the MPDV message cost curve in Figure 9.12. This confirms the analysis that MPDV's messaging cost is proportional to the number of paths computed.

In addition to measuring message costs to compute all-pairs paths, another way to gauge message cost is to measure the number of messages needed by a routing algorithm to react to a single topology change (e.g. link failure or recovery). This measure of message cost is given in Figure 9.13. This figure shows the message cost incurred by each routing algorithm to recompute paths when a random link fails and subsequently recovers (i.e. two topology changes).

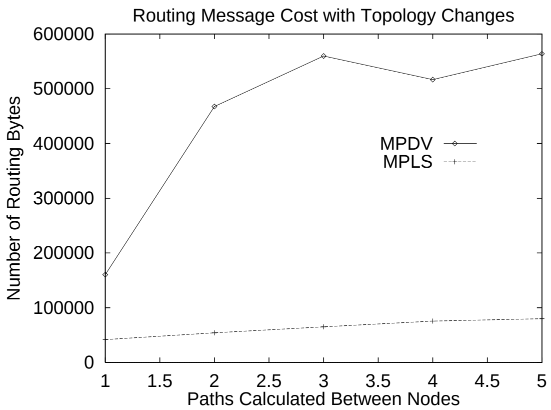


Figure 9.13 : The routing cost of MPDV and MPLS for a single link failure and recovery. The costs show the total number of bytes each routing algorithm transmits to adapt to one link failure and recovery.

In Figure 9.13, the x-axis denotes the rank, and y-axis shows the routing message cost when a link fails and subsequently recovers. The MPDV cost curve shows that routing message cost for an isolated topology change has the same general shape as the cost curve in Figure 9.12. However, the proportion of cost difference between 1-rank MPDV and 2-rank MPDV is higher. This is because the 2-rank MPDV has twice the amount of affected paths per topology change, resulting in twice the MPDV message size and two times the amount of updates

The MPLS cost includes topology broadcasts and source routes to establish non-suffix matched path IDs. Again, MPLS costs increase as rank increases. The cost curve in Figure 9.13 shows that the maximum source routing cost (at rank = 5) is approximately the same as the cost for a single MPLS failure and recovery broadcast.

The experiments in this section show that the message costs of both complete and incremental path computations are relatively low. Figures 9.12 and 9.13 show that MPDV costs are proportional to the number of calculated paths and that capacity-source routes contribute approximately L times more routing message cost compared to SPDV. For MPLS, our experiments show that the additional cost of source routing non-suffix matched paths are comparable to the message cost of broadcasting topology changes.

Given that current link speeds are on the order of Megabits to Gigabits per second and that multiple paths are recomputed only when topology changes, the additional multipath message costs, as shown in Figures 9.12 and 9.13, consume a negligible proportion of link bandwidth. Therefore, with the combination of low message overhead, infrequency of route recomputations, and high link bandwidth, we believe the extra messages incurred by MPDV and MPLS will consume a negligible amount of link bandwidth.

9.4.5 Routing Cost Summary

The various costs of executing MPDV and MPLS routing algorithms can be divided into the following five elements: 1) per packet path specification, 2) per packet forwarding time (which depends on forwarding table storage), 3) router general storage requirement, 4) router CPU usage, and 5) routing messages. The first two cost elements are performance critical because they are incurred on every packet and directly affect packet delivery times. The last three categories, although not performance critical, need to be low in order to make the implementation of multipath routing feasible in large-scale networks. The five cost categories are summarized below.

Using the suffix matched forwarding method, the per packet overhead of specifying suffix matched paths is $O(\lceil lg(K)\rceil)$, where K is the number of paths provided a between node pair. This bound holds for all MPDV based algorithms and for MPLS algorithms that compute suffix matched paths. For MPLS that uses non-suffix matched path calculation algorithms, the per packet overhead to specify paths to a destination D is $O(\lceil lg(K+p)\rceil)$, where p is the average number of non-suffix matched paths to D that passes through a router. In our experiments p is very small, $p \leq 5$.

The second cost category, per packet forwarding cost, is also low when using suffix matched forwarding. Because the path IDs are small integers, forwarding packets on multiple path requires only one additional hash lookup, which has minimal impact on router forwarding speed [48,149,160]. In addition, the suffix matched forwarding allows compact storage of paths in forwarding tables: the MPDV forwarding table complexity is $O(NK)$, K times more than single path DV. For non-suffix matched MPLS (e.g. capacity removal MPLS), a router's forwarding table complexity is $O(NK + NpL))$. In our experiments, the MPLS storage complexity no more than $2NK$. Thus the suffix matched forwarding method's small, fixed-length integer path IDs not only have low per packet overhead but also allow efficient packet forwarding.

The third cost category, general router storage, is also low. General router storage is not performance critical because the information stored in these memories do not directly affect packet forwarding efficiency. In our implementations, the per router storage for MPDV is $O(NKL)$, KL times more than single path DV, and $O(NM + NpL))$ for MPLS routers. From our measurements, we believe the memory requirements for both MPDV and MPLS can be satisfied by today's routers.

Router CPU usage is the fourth cost category. Because more paths are calculated between nodes, the CPU complexities are higher for multipath routing algorithms, compared to their single path counterparts. In MPDV, the CPU complexity usage is proportional to the number of paths examined and their path length ($\Theta(N^2M^3KL(ln(M))^2)$). This is a factor KL more than SPDV. For SPLS, the computation complexity of calculating shortest paths is $O(E * lg(E))$, whereas for MPLS, it is $O(MW)$. Here W is the complexity the multipath calculation algorithm. With the centralized capacity removal algorithm, $W = O(NKE * lg(E))$. Although the actual path computation algorithm has higher complexity, we believe that with the low frequency of path computation, better algorithm design, and advance processor technology, the computation complexity of MPLS will not be a performance bottleneck.

The last cost category is routing message overhead. A routing algorithm's message overhead depends on its path calculation process. For LS based algorithms, message costs are largely independent in the number and type of the paths calculated. This is because LS messages broadcast topology information and are thus independent of path calculation. However, because our capacity removal MPLS implementation source routes non-suffix matched paths, its message complexity is $O(NE + NpL)$. This is compared to SPLS whose message complexity is $O(NE)$. In our simulations, $NE + NpL < 2NE$, for both computing all-pairs paths and for computing paths after a single link failure or recovery.

Unlike LS based algorithms, the message costs of DV based algorithms are closely tied with path calculation because paths are computed through message exchanges. In SPDV, the message overhead is $\Theta(N^2M^3(ln(M))^2)$, and in MPDV, the message overhead is KL more, $\Theta(N^2M^3KL(ln(M))^2)$. MPDV uses a factor of KL more because the capacity removal algorithm needs to examine each path's links, and there are K more paths to examine.

Again, we believe that given current network link speeds and the relative infrequency of path computations, the extra message costs incurred by MPDV and MPLS will consume negligible network bandwidth.

Routing Alg.	Per Pkt Cost	Forwarding Tab.	CPU Usage	Rout. Msgs	Exp. Msgs
SPDV	$O(1)$	$O(N)$	$\Theta(N^2M^3(ln(M))^2)$	$\Theta(N^2M^3(ln(M))^2)$	0.72MB
MPDV	$O(lg(K))$	$O(KNL)$	$\Theta(N^2M^3KL(ln(M))^2)$	$\Theta(N^2M^3KL(ln(M))^2)$	5.6MB
SPLS	$O(1)$	$O(N)$	$O(Elg(E))$	$O(NE)$	2.1MB
MPLS-SM	$O(lg(K))$	$O(NK)$	$O(MW)$	$O(NE)$	2.1MB
MPLS-NSM	$O(lg(K+p))$	$O(N(K+p))$	$O(MW)$	$O(NE+NpL)$	2.2MB

Table 9.1 : A summary of the routing costs incurred by different routing algorithms. This table shows the different routing cost categories for the single shortest path DV (SPDV), capacity removal MPDV (MPDV), shortest path Link State (SPLS), suffix matched MPLS (MPLS-SM), and non-suffix matched, capacity removal MPLS (MPLS-NSM) routing algorithms. The last column shows the actual message cost measured for each algorithm on the 100 node 195 link network (for the multipath algorithms, the costs shown are for computing 3 paths). The variables in the table represent the following: N the number of nodes, E the number of network edges, K number of paths calculated between a node pair, p the average number of non-suffix matched paths to a destination that passes through a router, L the average path length, and W the centralized algorithmic complexity of calculating K paths between nodes.

Table 9.1 summarizes the different cost categories. The processing cost for packet forwarding is omitted in the table because it is the same for all five algorithms.

9.5 Experimental Conclusions

This section presents the cost and performance of the capacity removal MPLS and MPDV routing algorithms. The experiments were conducted on two relatively sparse Internet-like, cluster topologies (100 nodes with 195 and 170 links). The results of these experiments show that the implementations of capacity removal MPDV and MPLS routing algorithms provide additional network resources to end-hosts at low routing costs and that end-hosts can effectively use these additional resources to increase their performance.

Network performance is measured in terms of throughput, round-trip latency, and message drop probability. MPTCP is used to measure throughput, and a multipath ping program is used to measure latency and drop probability. With respect to throughput, our simulations show that MPTCP fully utilizes the paths provided by MPDV and MPLS. This observation confirms two important properties of our multipath architecture. First, both capacity removal MPDV and MPLS algorithms compute paths that provide more network resources. Second, MPTCP effectively uses these resources to increase network throughput.

In addition to throughput, our simulations show that multipath networks also allow end-hosts to successfully increase their performance in terms of lower latency and reduced message drop probability. A multipath ping program is used to measure these two performance metrics. Compared to a single path ping program, the simulations show that the multipath ping program achieves 15% lower round-trip delay and incurs approximately 50% less message drops.

The performance sections in this chapter show that multipath networks can provide quality paths and that these paths can be efficiently utilized to increase end-to-end performance.

The second focus of this experimental section is the runtime cost of multipath routing algorithms. The five cost categories are 1) per packet path specification, 2) packet forwarding, 3) router storage, 4) path computation, and 5) routing messages. Of the five costs, the first two are performance critical because they directly influence the speed of packet delivery. Given that MPDV and MPLS both use the suffix matched forwarding method, these two costs are kept at a minimal: fixed-length per packet path IDs, one additional path ID lookup per packet, and small forwarding tables. For the other three cost categories, our experiments show that their overheads are reasonably low, thereby making the implementation of multipath routing possible for large-scale data networks.

The conclusion of this chapter is that multipath routing can provide substantial end-to-end performance improvement, and we believe that the costs incurred by multipath routing algorithms can be satisfied in modern networks.

Chapter 10

Related Work

The research presented in this thesis builds on a large body of work. The purpose of this section is to compare and present other work relating to dynamic metric and multipath routing models. Section 10.1 presents related research in dynamic metric routing, and Section 10.2 multipath routing.

10.1 Dynamic Metric Routing

Dynamic Metric Routing Algorithms

The ARPANET was one of the earliest testbeds for dynamic metric routing. Dynamic metric routing was implemented in the early 1980's and used the Link State routing algorithm. This traditionally static metric routing algorithm was modified to compute dynamic metric paths in the following manner: every router continually monitors the costs of its outgoing links. Whenever the difference between a link's current cost and its last advertised cost exceeds a preset threshold, the router then broadcasts an LSP to reflect this cost change. The LSP broadcast initiates a global path recomputation that incorporates the new link cost. A dynamic metric Distance Vector algorithm can be implemented in a similar fashion.

As discussed in Chapter 3, two primary disadvantages of this type of routing algorithms were observed and documented by Khanna and Zinky [94]. First, dynamic metric LS/DV algorithms have the tendency to oscillate traffic from one area of congestion to another. These routing oscillations underutilize network resources and exacerbate network congestion. Second, the amount of routing resources used by both algorithms is unpredictable and hard to control. This problem is worsened because dynamic metric LS/DV's resource consumption increases as network traffic increases, thereby competing for network resources at a time when they are most scarce. These two disadvantages limit the wide use of dynamic metric routing in today's Internet.

The hybrid-Scout routing algorithm, to a large extent, overcomes these two limitations (see Chapter 3). Hybrid-Scout reduces route oscillation by independently calculating dynamic metric paths to a selected set of destinations, and its routing costs are easily controlled by Scout generating destinations. The key idea behind the hybrid algorithm is that calculating dynamic metric paths to a selected set of "hot" destinations is sufficient in reducing network congestion, and that such selective calculations can be done efficiently. This idea is made feasible by the high degree of destination locality observed in network traffic today.

Other Measures Of Dynamic Metrics

This thesis develops routing algorithms targeted for large-scale, packet switched wired networks. In these networks, dynamic link metrics are typically used to measure the amount of traffic experienced on a particular link. In other types of networks, however, dynamic link metrics can be used to measure link properties other than traffic.

One such application of dynamic metrics have been applied to wireless, ad hoc mobile networks. In these networks, computers are not physically connected but communicate using wireless channels. The reliability of a connection between neighboring computers depends on factors including distance, the number of interfering signals, etc. Because not all wireless "links" have the same signal strength, Dube et al. in [56] use dynamic link metrics to measure the strength of a wireless connection (measured in terms of signal strength). This link measure is dynamic because a wireless connection's signal strength changes as computers move.

In this context, the routing algorithm provides paths between nodes based not only on path length, but also on the signal strength of each link on that path. In their implementation, packets are routed on paths that use stronger signal links over paths that do not.

The routing algorithm used in [56] is a variation of the dynamic metric Distance Vector algorithm where route recomputations are triggered by link cost changes (Chapter 3).

We believe that the Hybrid-Scout algorithm can be applied in this network environment to efficiently calculate strong signal paths. Like wired networks, effectiveness of hybrid-Scout depends on the amount of destination locality in ad hoc networks. However, unlike wired networks which may consist of tens of thousands of nodes, ad hoc networks are typically small, on the order of less than fifty nodes. Given their size, the reduction in routing cost of hybrid-Scout, compared to dynamic metric LS/DV algorithms, will not be as significant as observed in large networks.

Quality of Service Routing

The current Internet provides a best-effort service model: packets are delivered to their destinations without guarantees on the success of packet delivery nor on delivery times. However, with the emergence of applications such as IP telephony and video conferencing, many advocate that traditional best-effort networks should support mechanisms that provide delivery guarantees [17, 29, 40, 86, 162].

These delivery guarantees are generally refer to as *Quality of Service (QoS)* [34, 91, 162, 169]. The QoS of a path is typically specified as a combination the path's delay bound (i.e., packets sent on this path will reach the destination in time less than t) and the path's available bandwidth (i.e., X amount of throughput can be obtained on this path before packet drops). Before transmitting data, an application that requires QoS needs to specify its QoS requirement and make reservations on the path to its destination (i.e. on every router in the path). Once the reservation is made, the application is assured to receive the requested QoS.

Notice, at any given time, the QoS requirement that a router can satisfy depends on the QoS requests the router is currently supporting. That is, the probability a router can satisfy a certain QoS request is dynamic and depends on traffic. Thus, a dynamic link metric can be formed to expresses the probability that a QoS request can be satisfied, and paths can be computed to optimize this metric.

Indeed, there are several proposals to compute paths that consider QoS metrics [9, 11, 72]. Currently, there are two general approaches to implement QoS routing. The first is to use dynamic metric versions of the Link State or Distance Vector algorithms [9, 72] (refer to Chapter 2 for their implementation). The triggering mechanisms allow the two routing algorithms to calculate paths that maximize the probability of satisfying QoS requests. Recall that in datagram networks, routing instabilities tend to occur in dynamic metric routing because path recomputations can potentially shift a large amount of data from one congested area to another. However, this large traffic shift does not occur in QoS routing because once a QoS path is reserved, packets traveling the path are not re-routed, even though the current path between the source and destination may have been recomputed to use a different path. That is, reserved QoS paths are pinned and packets traversing these paths are not affected by subsequent path computations, thereby reducing large traffic shifts and routing instabilities.

We believe the hybrid Scout algorithm be successfully applied to compute QoS paths. However, because routing instabilities and costs are reduced in QoS based LS and DV algorithms, we believe that the cost-performance benefits of hybrid Scout benefits will be less compared to ones observed in congestion-oriented dynamic metric routing.

The second approach to QoS routing is to dynamically calculate a QoS path given a QoS request [10]. In this approach, whenever a router receives a QoS request from a host, the router dynamically computes a path with the highest probability of satisfying the request. Once the path is calculated, the router then forwards the QoS request on that path. To make this approach feasible, the challenge is to reduce the amount of router processing needed for dynamic path computation.

In addition to explicitly considering QoS constraints in the actual routing algorithm, researchers have studied other methods to increase network QoS. One such approach to examine the impact of multipath networks. That is, given a network that offers multiple paths between nodes, how can end-hosts best utilize these paths to better ensure their network

QoS to their destinations.

Roa and Batsell in [131] proposed a method which guarantees that end-hosts can transmit their messages with minimal delay, given an accurate view of the network (e.g. up-to-date information of router queues and traffic patterns). Cidon and Rom developed end-host reservation algorithms to increase the probability of reserving QoS request on a multipath network [39]. The different algorithms they developed make different tradeoffs between the probability of satisfying a QoS request with the amount of network resources and time needed to satisfy the request.

Algorithms for Highly Dynamic Networks

Another network environment which requires routing techniques similar to dynamic metric routing is in highly dynamic networks. Highly dynamic networks are characterized by frequent changes in network topology. These networks are typically wireless and highly mobile. Abstractly, dynamic networks can be modeled as a dynamic metric networks where a connected link has cost 1 and a unconnected link has cost infinity, and a link's cost changes between these two values to indicate the link's connectedness (value 1) or disconnectedness (value infinity).

Johnson and Maltz proposed a host-initiated routing algorithm for such networks [87] called Dynamic Source Routing (DSR), which is similar to Scout [33]. The assumption of their work is that the network topology changes so rapidly such that maintaining valid routes to all destinations is infeasible. Thus DSR calculates paths on demand: whenever a node wishes to transfer data to another node, the sending node initiates a path computation to that node. DSR is similar to Scout in that small messages are flooded to calculate paths. However, they differ in several key aspects. First hybrid-Scout maintains paths between all nodes; in an environment such as the Internet, DSR's on-demand path calculation is prohibitively expensive. Second, DSR initiates path calculation from the source, as oppose to the destination. Third, Scout is guaranteed to converge on the shortest path, DSR simply computes a feasible path. Fourth, DSR uses source routing, whereas hybrid-Scout uses hop-by-hop forwarding.

Perkins in [127] proposed a modification to DSR called the Ad Hoc On-Demand Distance Vector (AODV). This routing algorithm is a hybrid of DSR and DV for ad hoc networks. AODV uses DSR for on-demand route discovery of hop-by-hop routes and uses the DV mechanism for invalidating the on-demand paths when link failures are detected. Like the hybrid-Scout algorithm, AODV is an hybrid algorithm. However, unlike hybrid-Scout where both the DV/LS and the Scout component compute routes, the DV component in AODV is used only to invalidate routes discovered by DSR (in a similar way the DV component in hybrid-Scout invalidates Scout paths). Again, the AODV algorithm cannot be directly applied to large-scale datagram networks because on-demand route discovery is too expensive.

10.2 Multipath Routing

This section surveys related work on multipath routing. The discussion is organized as follows: the first section describes other designs and/or implementations of multipath networks in comparison with the one proposed in this thesis. Section 10.2.2 discusses other work specifically related to algorithms that calculate multiple paths, and the last section surveys different proposed path forwarding methods.

10.2.1 Multipath Networks and Architectures

IBM SNA

The IBM Systems Network Architecture (SNA) network is a wide area network first deployed in 1974 [70]. This network provides multiple paths between nodes because its designers thought that the traffic generated by routing algorithms that adapt to network failures/recoveries would incur prohibitively high routing costs* [6]. Therefore to account for network failures, multiple paths between nodes are centrally pre-computed so that if one

*This is known to be untrue today.

path fails, other paths would still be operational. SNA also has a notion of multi-service and multi-option paths: the service type are interactive and batch which are used to dictate packet priority.

Conceptually, the SNA multipath model is very different from the one developed in this thesis. First, the purpose of SNA's multiple paths is to provide fault-tolerance without using adaptive routing; whereas in our model, fault-tolerance is provided by the adaptiveness of the routing algorithm, and multiple paths are used solely to increase network performance and services. Second, path services in SNA are predefined. In contrast, the proposed multipath model is designed to accommodate future path services that have yet to be defined.

In practice, SNA also differs significantly from our proposed multipath network. First, SNA's multiple paths are *predefined* and *centrally pre-computed*. That is, path computation is done centrally off-line where routing tables are distributed after this computation is complete. Since SNA routing is not distributed and not adaptive, it cannot be practically deployed in wide-area networks of today's scale. Secondly, the SNA network restricts an application to use only one path at a time, and the path is specified at connection time. On the other hand, our multipath routing model allows full application freedom on the usage of multiple paths.

Scalable Routing Architecture

Estrin et al. proposed a more recent multipath routing model targeted for today's large-scale Internet environment called the Scalable Inter-domain Routing Architecture (SRA) [59]. This proposed multipath routing scheme supports two types of paths. The first is the static metric single shortest paths calculated by a traditional DV routing algorithm. Packets traveling on these paths are hop-by-hop forwarded as in the current Internet. These "generic" paths are calculated by routers, and it is expected that the majority of network packets will use these generic paths. The second type of path called Type Of Service (TOS) path, is calculated by end-hosts. Packets sent on TOS paths are forwarding using IP loose source routing [57]. Calculation of these paths requires end-hosts to first collect the network topology information from routers and then individually calculate the desired path(s). Since there are no restrictions on path calculation, applications in SRA can send packets on arbitrary paths.

Although SRA and the multipath model considered in this thesis are targeted for the same network environment, they are fundamentally very different. In SRA, packets traveling on non-shortest paths (i.e. on TOS paths) are expected to be few in comparison to packets on generic paths. This assumption is reflected in the difficulty in which TOS paths are calculated and the inefficiency of the packet forwarding method. Whereas, in the multipath model proposed, all paths are calculated with the same efficiency and packets are forwarded the same way irrespective of the path they travel. In short, our multipath model views packets traveling non-generic paths as the general case, whereas in SRA, they are viewed as the exception. This perspective difference is evident in each model's path calculation and packet forwarding methods.

Detour

As mentioned in Chapter 2, Internet routing is decomposed into routing domains for scalability reasons. Routing domains are typically divided by domain ownership, as opposed to domain geographies. This causes routing inefficiency because the path between two geographically close nodes may actually traverse domains that are geographically very far from the source and destination domains. In addition, inter-domain routing algorithms calculate paths based on minimizing the number of domains traversed. This policy also gives rise to sub-optimal routing because the time needed for a packet to traverse a domain depends on the size of the domain, which is highly variable. The result of domain decompositions and inter-domain routing policy frequently causes routing anomalies because minimizing the number of domains in a path does not necessarily minimize path delay. For example, sending a packet "directly" from domain A to B may take longer than sending the packet "indirectly" from A to C and then to B.

The Detour project aims to overcome such routing efficiencies caused by these two factors [8,136]. Detour's approach is to establish a set of "smart" routers that communicate with each other to setup a virtual network. The nodes of the virtual network are Detour routers, and a virtual link connecting two Detour routers is the path in the physical network that connects the two routers. Packets sent by one Detour router are tunneled [134] to its neighboring router. The cost of each virtual link is computed dynamically by sampling the link's delay. In the previous example, Detour hopes that its routers will automatically reroute packets sent from A to B through C using this virtual network.

Detour also plans to support "multipath routing": if Detour routers detect that there are multiple, virtual paths to a destination, the routers will distribute data among the multiple paths. Notice that this type of multipath routing differs from the model proposed in this thesis because end-hosts in Detour do not have control over which path to use. As stated in Chapter 4, the absence of end-host control limits the performance gains of multipath networks.

In short, Detour aims to address inter-domain routing inefficiencies by establishing a virtual network that delivers packets based on actual observed path latencies (i.e. not based on the number of domains traversed). Detour also plans to support a form of multipath routing where end-hosts do *not* have control over which path to send their packets.

We believe the Detour architecture may be used to incrementally deploy the multipath routing model described in this thesis. Using the virtual network, Detour-like routers can implement end-host controlled multipath routing. Details of this approach are given in Chapter 11.

Specialized Networks

Another network environment where using multiple paths have been proposed and implemented is in short-hauled networks [50,106,166]. Short-hauled networks are characterized by tightly coupled network elements and are usually implemented in multiprocessor networks or switches. Examples of short-hauled networks are Banyan networks [140], Omega networks, mesh topology networks [143], cube topology networks [37,55,105,143], general multistage networks [49, 104], and internal ATM switching elements [76, 88, 145, 155, 164].

The goals for providing multiple paths in a short-hauled network are to decrease switch blocking probability (e.g. switch overloading) and to increase fault tolerance [132, 161]. In addition to scale, the main conceptual difference between a short-hauled network and a wide area network is that paths between nodes within short-hauled networks are hardwired. That is, paths between nodes are determined at network design/construction time, as oppose to one that is dynamically determined and adapts to component failures.

Hardwiring paths is possible in short-hauled networks because node addresses are chosen so that the network can determine the path to a node by inspecting the node's address. In this well defined environment, short-hauled network designers are able to assign, *a priori*, multiple paths between nodes.

Unfortunately, the methods employed in these networks are not directly applicable to wide area networks. The three main reasons are 1) wide area networks are not tightly coupled, 2) the wide area network environment is dynamic (i.e. node and links may fail), and 3) it is not possible to pre-assign paths to nodes based on node address.

10.2.2 Multipath Calculation Algorithms

This section presents related work on multipath calculation algorithms. Unlike the multipath networks presented in the last section, the algorithms described here are concerned with path computation and are generally not designed for a distributed network environment. Nevertheless, these algorithms are important because they provide the basis on which to build distributed multipath calculation algorithms.

Introductory algorithm text books such as [19, 42, 108] provide basic path computation algorithms. For more advanced graph algorithms, refer to [5].

Disjoint Paths

The most obvious multipath calculation criteria is to calculate multiple disjoint paths. There are two types of path disjointness, node and link disjointness. A set of paths between two nodes is node (link) disjoint if and only if all paths in the set do not have any node (link) in common [152]. Several algorithms are surveyed below.

The algorithm presented in [118] computes pair disjoint paths between nodes such that the total costs of the paths are minimal. Later work by [36] extended the algorithm to compute K disjoint paths with guaranteed total minimal path cost. Notice that both of these algorithms do not guarantee that the least cost path is in the computed set.

Sidhu et al. developed a distributed multipath calculation algorithm that calculates node disjoint path and guarantees that the shortest path is in the calculated set [146]. This algorithm calculates path from node X to Y by considering the relative position of Y on the shortest path tree rooted at X. Alternate paths are calculated by finding cross edges that connect Y to X. The two major drawbacks of this algorithm are that 1) the algorithm may diverge if the network topology changes during path computation, and 2) the worst case message complexity is exponential.

A variant of disjoint paths is initial link disjoint paths. Initial link disjoint paths, described in [156], is a centralized, dynamic programming algorithm that calculates K paths to every node such that the paths to a particular node has distinct initial links. The algorithm is based on Dijkstra's shortest path algorithm [54].

Shortest K Paths

Another criteria for calculating multiple paths is to calculate unconstrained K shortest paths. The objective of this criteria is to find K distinct paths between two nodes such that they have the minimum sum compared to all other K possible distinct paths. Many algorithms have been developed to compute these paths; refer to [103,141,165] for different approaches. The computational lower bound for computing unconstrained K shortest path is $O(m + n\log(n) + kn)$, where m is the number of edges, n number of nodes, and k the number of path to calculate between nodes [58].

Multi-commodity Flows

Perhaps the most well study of all multipath calculation algorithms are ones that compute multi-commodity flow paths. The constraint of a multi-commodity flow problem is to minimize the total cost of transmitting multiple resources to and from different nodes. This optimization is constrained by the capacities of each link for each resource and the costs of using a link [5, 63]. For more references on this subject, refer to [13, 14, 89].

A variant of multi-commodity is maximum flow. A maximum flow algorithm computes paths between two nodes such that the aggregate capacity of the calculated paths is maximized between the two nodes. These paths can be calculated efficiently and are applicable to routing algorithms that wish to provide high capacity paths between nodes [5]. The capacity removal algorithm presented Chapter 5 also computes paths to increase capacity. But unlike maximum flow algorithms, the capacity removal algorithm considers path length and the number of paths calculated.

10.2.3 Multipath Forwarding Methods

As Chapter 4 showed, forwarding messages on multiple paths is significantly harder than on single shortest paths because paths in a multipath network are not guaranteed to be suffix matched. For paths that are suffix matched, Chapter 6 presented an efficient forwarding method that require fixed-length per packet path ID and router storage linear to the number of paths provided. This section surveys other path forwarding methods.

Source Routing

Source routing is a forwarding technique in which the sender of a message determines the path a message is to travel and tags the message with that path. That is, a message contains an explicit list of nodes the packet is to traverse. Upon receiving such a message, a router simply forwards the message to the neighbor specified in the message's source route.

This method is very flexible because messages can be forwarding on arbitrarily paths. However, source routing has two major drawbacks. First, with each message carrying an explicit path, source routing's message overhead is variable length and high for large networks and long paths. Second, for each router, the storage requirement for each path is $O(LP)$, where L is the average path length, and P is the size to encode each element in a path. This requirement is compared to $O(1)$ in conventional single path routing algorithms and the proposed suffix matched forwarding method.

Label Swapping

Label swapping performs path forwarding by using local path identifiers [139]. In label swapping, a path $X = (x_0, \ldots, x_n)$ is implemented by nodes x_i and x_{i+1} sharing a local path identifier (LPID) such that whenever x_i receives a message with $lpid_{i-1}$, it forwards the message to x_{i+1}, replacing the LPID with $lpid_i$, $0 < i < n$. Examples of label swapping are the suffix matched forwarding and ATM's virtual circuit forwarding methods.

The advantage of label swapping is that the message overhead and the per path router memory overhead are constant. However, the difficulty in label swapping is the establishment of LPIDs such that they globally compose the calculated paths. For example, ATM requires a special pre-connection packet that traverses the intended path to establish LPIDs. Notice that in the suffix matched forwarding method, however, LPIDs are implicitly made consistent by exploiting the suffix matched property and therefore do not need pre-connection packets.

The label swapping technique has also been applied to make source routing more efficient [85]. For example, the solution adopted in Chapter 6 uses source routing with label swapping to establish LPIDs for non-suffix matched paths. Using this optimization, a path is source routed only once, and label swapping is used subsequently. This optimization greatly amortizes the initial source routing overhead.

Compact Addressing

Source routing lists nodes on a path where nodes are identified by a node identifier, which can be very large (in number of bits). For example, a node on the Internet has an ID that is 32 bits long [30] and will increase to 128 bits with IPv6 [26]. Compact addressing reduces the amount of bits needed to enumerate a path by observing that nodes only forward messages to their neighbors. Therefore, if a node has a local name for each neighbor, say $w_1, \ldots, w_M$ for neighbors 1 to M, then a node only needs $\lceil \log(M) \rceil$ bits to forward messages to its correct neighbor [67]. Assume that a node has at most M neighbors, then to encode a source route of length l takes $l * \lceil \log(M) \rceil$ bits. Since M is typically not very large (< 20), this optimization can substantially reduce the overhead of source routing packets.

Chapter 11

Conclusions and Future Directions

The contributions of this dissertation are the methods developed to implement dynamic metric and multipath routing models, and the experimental analysis which demonstrates that the two models can be implemented efficiently and significant performance gains obtained. The two routing models, compared to the current Internet's static metric single path model, better utilize network resources to provide higher network performance.

For the dynamic metric routing model, this thesis develops a novel routing algorithm, hybrid-Scout, that increases network performance by exploiting the destination locality observed in network traffic. For multipath routing, this thesis develops not only efficient multipath routing algorithms, but also a transport protocol that effectively uses multiple paths.

This chapter concludes the research presented in this dissertation and outlines potential future research directions.

11.1 Conclusions

Dynamic Metric Routing

This dissertation began with the development of a dynamic metric single path routing algorithm, the hybrid-Scout. Traditional dynamic metric routing algorithms based on Link State and Distance Vector suffer two primary drawbacks: the tendencies for routing oscillations and uncontrolled routing costs. The hybrid-Scout algorithm addresses these two problems through selective, time staggered path computations that are controlled by destinations.

The algorithm is selective because it calculates dynamic metric paths only to certain "hot" destinations – analysis of Internet traffic locality indicates that a few destinations receive the majority of network traffic (the top 1% destinations receive 60% of network traffic); thus calculating dynamic metric paths to these destinations significantly impacts the performance of the entire network. Moreover, the dynamic metric paths are calculated in a time staggered manner. These two properties of the hybrid-Scout algorithm, selective and time staggered route computation, reduce route oscillations because not all paths are calculated at the same time.

In addition to increasing route stability, the routing costs incurred by hybrid-Scout are also predictable. In traditional dynamic metric routing algorithms, routing traffic are triggered by network traffic, making them unpredictable and traffic dependent. In hybrid-Scout, however, routing traffic are controlled by the destinations whose paths are dynamically calculated. Therefore, routing costs are easily controlled by bounding the routing costs of these destination.

To validate the hybrid-Scout algorithm, extensive simulations were conducted to measure its effectiveness and efficiency and to determine the conditions under which hybrid-Scout is better than dynamic metric LS and DV. Simulation results show that, for an Internet-like topology, 1) hybrid-Scout is effective at rerouting network congestion, under the condition that at least 50% traffic are destined to selected destinations, and 2) hybrid-Scout is more efficient (by 1 to 2 orders of magnitude in routing cost) than both dynamic metric LS and DV if less than 10% of nodes generate Scouts. Since these conditions are present in today's Internet, we conclude that hybrid-Scout, if implemented, can improve Internet performance.

Multipath Routing

The second main contribution of this dissertation is the development of methods that implement and utilize multipath networks. Multipath networks are networks that provide multiple paths between nodes at the same time, and transmitting nodes can select which path(s) to use. With respect to multipath routing, the contributions of this dissertation are the design and evaluation of algorithms that make multipath routing feasible for large scale data networks.

To this end, solutions to solve two main multipath problems are developed: 1) routing algorithms that efficiently support multiple paths between nodes, and 2) an end-host protocol that effectively uses multiple paths to *actually* increase throughput performance.

The first problem, supporting multiple paths efficiently, is primarily concerned with the cost of specifying and forwarding packets on their intended paths. While trivial in single path routing, this process is difficult in a multipath context because there are more than one path to a destination. To solve this problem, this dissertation develops a novel forwarding method, the suffix-matched forwarding method, that uses fixed-length per packet path IDs and requires router storage linear to the number of paths calculated. The method's low overhead is achieved by exploiting path sets with the suffix matched property. To demonstrate the applicability of suffix matched forwarding, this thesis extends the method to multipath versions of the LS and DV routing algorithms.

The second multipath contribution of this thesis is the development of an end-host protocol that effectively uses multiple paths to increase end-to-end and network throughput. This thesis develops a multipath transport protocol, MPTCP, that provides a reliable bit-stream service and effectively uses multiple paths to maximize throughput. Extensive simulations show that MPTCP is able to increase throughput proportional to the amount of available network resources. Furthermore, MPTCP sustains high network performance even during very high levels of network utilization.

In addition to performance, this thesis also measured the costs incurred by two multipath routing algorithms, MPDV and MPLS, based on DV and LS routing algorithms respectively. Because both algorithms use suffix matched forwarding, the performance critical operation of packet forwarding uses fixed-sized, per packet path IDs and adds only one additional lookup in router's packet forwarding procedure. With respect to non-performance critical costs, simulations show that the overhead of router CPU usage, router message cost, and router storage requirements are reasonably low so that we believe the two routing algorithms can be feasibly implemented in today's network routers.

11.2 Future Directions

Two general directions of research can follow from the work presented in this dissertation. The first is the improvement of the hybrid-Scout dynamic metric routing algorithm, and the second is the further development of the multipath routing model.

11.2.1 Hybrid-Scout

The efficiency of the hybrid-Scout algorithm depends on a network's destination locality: experiments show that if at most 10% of the destinations (hot destinations) generate at least 50% of the network traffic, hybrid-Scout is more efficient and effective than traditional dynamic metric routing algorithms. This thesis assumes that the hot destinations are known and that each destination generates Scouts at a rate appropriate for the destination's received traffic. A future research direction for the hybrid-Scout algorithm is to remove the two assumptions: develop a mechanism that determines the set of hot destinations and the Scout generation rate for each hot destination.

To accomplish this, a distributed traffic monitoring protocol can be installed in network routers. The protocol gathers traffic statistics to determine which destination networks are "hot". Once this determination is made, routers directly connected to these destinations can then generate Scouts on each of those destination's behalf. In addition, since routers have traffic statistics of all hot destinations, the appropriate Scout generation rate can be derived for each destination.

The development of mechanisms to determine hot destinations and their Scout generation rate will make hybrid-Scout easier to deploy. With these mechanisms, hybrid-Scout will not require manual configuration of Scout generating destinations and their rates.

11.2.2 Multipath Routing

This dissertation laid the ground work for supporting multipath routing in large-scale networks. Because applications of multipath routing have largely been unexplored, there are many open research issues. Among others, a list of avenues worth pursuing are 1) the support for different path services 2) the investigation of cooperative multipath transport protocols, and 3) the incremental deployment of multipath routing in an Internet-like network. Each of these possibilities are described in detail below.

Supporting Different Path Services

This dissertation examines two path services, low delay and high throughput services. These two services are chosen because of their prevalent use. However, with increasing diversity of network applications and demands, the need for paths with other services has increased. Examples of these new services are QoS, security, real-time, and network pricing. Possible research directions are to develop multipath methods to efficiently support these path services. To give an example of the issues involved in supporting each of the mentioned services, an outline of some of the problems involved in supporting the QoS path service is given below.

To provide QoS path services, routers need to enforce queuing disciplines to provide different types of delivery guarantees [95]. Example QoS services are guaranteed, priority, and bounded services [11, 43]. Current proposals to support QoS have been primarily focused on the single path routing model. In a multipath setting, if an end-host has can make QoS requests on multiple paths to a destination, one expects that QoS requests can be more easily satisfied. Doing this raises two research issues: how should multiple paths be calculated in order to consider router QoS guarantees? And how end-host applications should best use these paths to most efficiently obtain their QoS requirements?

The first issue is concerned with path calculation: designing path calculation algorithms such that the paths calculated between nodes provide the maximum likelihood of satisfying end-host QoS requests. As stated in Chapter 5, end-hosts can benefit from a multipath network only if the paths provided offers higher performance than in a single path environment. The second issue relates to designing protocols that efficiently use network resources to provide QoS guarantees. For example, a QoS protocol should have the ability to combine QoS guarantees of individual paths to provide an overall QoS for an application. Protocols to accomplish this task need more sophisticated path management strategies than the scheme MPTCP uses for dynamic load balancing.

Transport Protocols

Another possible area of research is to design a multipath transport protocol that uses multiple paths in a cooperative manner. Recall that MPTCP uses separate TCP connections on each available path, and each path is solely responsible for transmitting its own packets. In a cooperative protocol, a more flexible form of load balancing can be used and many redundancies may be reduced.

For example, a cooperative protocol can load balance lost segments from one connection to another. As mentioned in Chapter 7, this reduces receiver buffering which can increase protocol throughput. A cooperative protocol can also reduce redundancies. For example, receivers can send one ACK to acknowledge data received on several connections, and the combined ACK should travel the fastest path back to the sender.

Multipath Incremental Deployment

The algorithms and techniques developed in this dissertation are designed and evaluated under realistic network assumptions. Thus, the natural research question is how to deployed the proposed method in an actual large-scale, wide area network.

In practice, deploying algorithms in wide area networks is done in an incremental fashion. The incremental process is necessary because of the network sizes and because different parts of these networks are often under different administrative controls. These two factors make simultaneous routing algorithm updates of every router in a large-scale network impractical. Thus, a practical research subject is developing a method to incrementally deploy the multipath routing algorithms developed here.

As mentioned in Section 10.2.1, a Detour-like architecture can be used to incrementally deploy the proposed multipath model [47,115,136]. Using this approach, a router with multipath capabilities establishes a virtual multipath network with other multipath-routers. In this virtual network, the nodes are multipath-routers, and the links are established between these routers using IP tunneling [134]. With this architecture, only end-hosts whose local router is a multipath router can use the multipath network because the local router is responsible for translating host multipath IDs into suffix matched path IDs. In incremental deployment, the connectivity of the virtual network increases as more multipath routers are placed in the network, resulting in higher performance of the multipath network.

Appendix A

The Scout Routing Algorithm

This chapter describes the Scout routing algorithm. Scout is a destination initiated, selective shortest-path computation algorithm. The key idea underlying Scout is that a destination periodically floods small, fixed-sized messages through the network which explore available paths back to the destination*. These exploratory messages, called *Scouts*, are of a fixed, small size and can be readily piggy-backed onto data packets on a hop-by-hop basis, which largely defrays their cost to the network.

The remainder of this chapter is organized as follows. The next section describes the Scout routing algorithm, followed by Section A.2 which proves its correctness. Concluding remarks are given in Section A.3.

A.1 Flood Based Route Discovery

To facilitate the presentation, the Scout algorithm is presented by initially assuming idealized network characteristics. These assumptions are then progressively removed and additions provided to deal with the relaxed assumptions.

Ideal Network: To feature the basic concepts of the algorithm, assume

- Static network topology where all links have unit cost.
- Message delay along each link is equal.

We define the *shortest-path tree* rooted at node R as a tree which connects all reachable nodes to R with the least cost.

*In general a destination could represent a single host, a subnetwork, or an entire routing domain.

The basic mechanism of route discovery is through message flooding. In the Scout algorithm, each node in the network periodically sends a Scout message to all its neighboring nodes. Let R be the node initiating the Scout message. The period between two consecutive floodings of Scout messages from R is called the *broadcast interval* (BI). A Scout [R, C_R] contains the originating node's address, R, and the cost to reach R, C_R. Initially C_R is zero. When a node P receives a Scout message [R, C_R] from its neighbor Q, P first modifies the Scout's cost C_R to include the cost of sending a message from P to Q, $C'_R = C_R + Cost(P \rightarrow Q)$. C'_R represents the cost a message will take if it is sent by P via Q to R. Under the two assumptions above, if P has already seen a Scout from R in the current broadcast interval, P already has a route to R with cost at least C'_R. In this case, P remembers this alternate path but does not forward the Scout. Otherwise [R,C_R] is the first Scout P has seen in this broadcast interval, then P forwards [R, C'_R] to all its neighbors except Q. Since nodes only forward R's Scout at most once in each BI, the flooding of Scout messages terminates after every node has forwarded a Scout message of the form [R, C_R]. This means the total number of Scouts exchanged in each broadcast interval for a single node is $O(L)$, where L is the number of links in the network.

After the termination of R's BI, every node in the network knows the minimum path to R. This knowledge is represented in the form of a next-hop address: node P the minimal path to R if P knows the minimal cost to R and which of its adjacent nodes is on the minimal path. In the Scout algorithm, node P knows the shortest cost to R and the shortest next-hop to R when P receives the first Scout message in a BI. Nodes decide the shortest next-hop to R as the neighbor from whom they received the least cost Scout in the current BI. The forwarding tree constructed is a *sink* tree rooted at R. Since nodes only advertise (or forward) the least cost route to R, the Scout algorithm guarantees that all nodes will know the minimum path to R when flooding terminates. All pairs shortest path is computed once every node in the network floods its Scout message; thus the worst case time complexity of the Scout algorithm is $O(NL)$ messages, where N is the number of nodes in the network.

To disambiguate Scout messages from one broadcast interval with another, assume for the time being that before R floods, all messages of R's previous flood are terminated. This requires the BI to be at least the time it takes for a message to traverse the longest path in the network. This assumption is removed in the general algorithm.

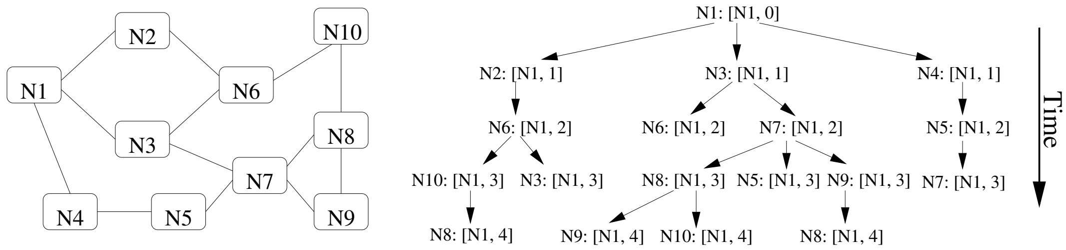


Figure A.1 : Example network topology and the first round broadcast tree from N1. All link costs in the network are 1.

Figure A.1 shows the *broadcast tree* for Scout message flooding initiated by N1 in the above network under our idealized assumptions. A broadcast tree represents an event ordering of sending and receiving messages.

The figure illustrates a sequence of events in which Scouts are sent and received after N1 initiates a broadcast interval. A node Ni: [Nx, k] in the tree represents an event in which node Ni receives a Scout [Nx, $C'_{\mathrm{N}x}$] . Edges show Scout message transmissions and time flows downward in the figure. For events on the same level of the tree, we adopt the convention that events on the left happen earlier than events on the right. For example, event N6: [N1,2], the child of N2: [N1,1] happens before N6: [N1,2], the child of N3: [N1,1]. That is, node N6 receives a Scout from N2 before receiving one from N3, therefore as the figure shows, N6: [N1,2] child of N2: [N1,1] forwards the Scout while the other does not. Leafs of the tree correspond to events which occurred on nodes who have already received a Scout from N1. As Figure A.1 shows, the shortest path is computed when flooding terminates.

Non-uniform link cost and delay: The Scout algorithm described above relies heavily on the assumption that the first Scout message of a BI is the one with minimal cost. Next, this assumption is removed by allowing non-unit link costs and non-uniform (but bounded) link delays. For now, the assumption that all Scouts of R are terminated before R's next flood and that the network topology is static is preserved.

We define two terms. Node P's *designated neighbor* to R in the broadcast interval i is defined as the neighbor that gave P the least cost Scout to R in broadcast interval $i-1$. P's *upstream node* to R in broadcast interval i is the neighbor which provided the Scout that P forwarded to its neighbors (after modifying its cost).

As with the previous algorithm, every node R periodically floods its Scout message [R, C_R]. Upon receiving [R, C_R] from a neighbor, P computes C'_R as before. In the first broadcast interval, immediately after receiving the first Scout message from R, node P forwards [R,C'_R] to all neighbors except R. Node P might receive more of R's Scouts in the same BI, indicating different paths and path costs to R. P remembers these Scouts and adjusts its data forwarding table to reflect the new paths, but P does not forward the Scout messages. In the next broadcast interval, P waits to receive a Scout message from its designated neighbor before flooding. When P receives the Scout from this designated neighbor, P computes C'_R and examines all other Scout messages it has received in the current BI (including the one from its designated neighbor), to find the Scout with the least cost C''_R to R[†]. P forwards [R, C''_R] to all neighbors except the neighbor from which it received the Scout with the least cost. This neighbor is now P's upstream node for this BI.

As an example, consider the network in Figure A.1 with the cost of edge (N1, N2) and (N1, N3) set to 5 and all other edge weights being 1. In the first broadcast interval of N1, the broadcast tree is exactly the same as in Figure A.1, except the cost of the N2 and N3 subtree is increased by 4. Figure A.2 shows the broadcast tree for the second broadcast interval of N1.

Notice in Figure A.2, N7 does not forward the first Scout it receives (one from N3) but waits for N5 (its designated neighbor) and then forwards the Scout with the least cost. The shortest-path tree is computed in the third broadcast interval when N3, N8 and N9 wait for

[†]In general, with a dynamic network, the least cost Scout may not come from the designated neighbor, since link costs may change between broadcast intervals.

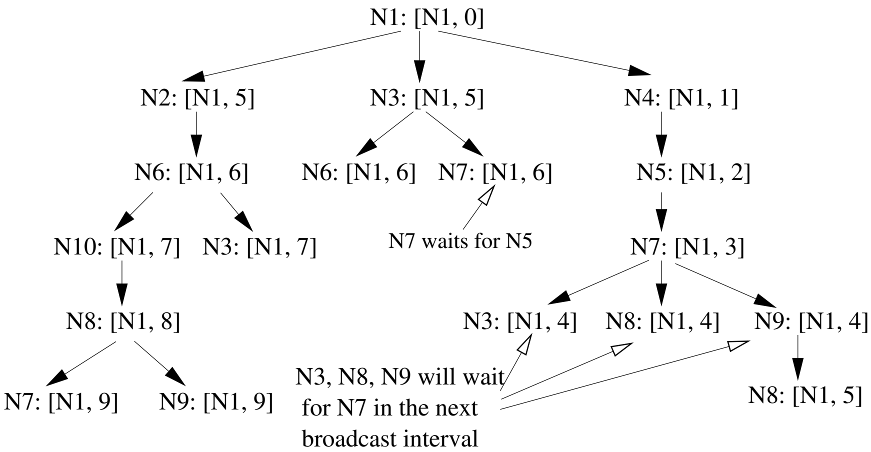


Figure A.2 : Broadcast tree after the second round of flooding from N1 in the network of Figure 1 with unequal link costs.

N7 and forward the least cost Scouts to N6 and N10.

Under these network assumptions, the shortest path computation to any node R is bounded by K broadcast intervals, where K is the depth of the shortest-path tree rooted at R. In the example above, the depth of the shortest-path tree is 5 and the Scout algorithm converged in 3 broadcast intervals. It is worth mentioning that although the shortest-path computation is bounded by K broadcast intervals, all nodes know *a* path to R after R's first broadcast interval. The quality of the path progressively improves with additional broadcast intervals.

Unrestricted Network: This is the generalized network where nodes/links can fail and recover and link costs can change.[‡]. The static network assumption guarantees that if node P receives a Scout message in the previous broadcast interval from neighbor Q, P will also receive a Scout message from Q in the current broadcast interval. If links are allowed to fail and recover, P might never receive another message from Q. To resolve this problem, Scout is modified by requiring P to flood the first Scout message from R if P did not flood any of R's Scout messages in the previous BI. In other words, if P waited for a Scout from

[‡]Notice that convergence of a shortest path computation applies only when the network is not changing for some period.

its designated neighbor Q in the previous BI but never received a Scout, instead of waiting for Q, or any other neighbor in the current round, P immediately floods the first Scout it sees in the current broadcast interval and recalculates its designated neighbor.

P is not allowed to wait for a neighbor in the current BI, in particular, the designated neighbor, because if there are multiple failures, waiting for the best information might cause cascading waits. This decision was motivated by the observation that propagating more recent, perhaps sub-optimal, information is more useful than trying to wait for the best information which entails the risk of not propagating any information at all.

To handle overlapping broadcast intervals from the same source, Scout messages are simply tagged with a sequence number indicating the current broadcasting interval. The sequence number ensures that the algorithm only makes routing decisions based on the information from the most recent broadcasting interval. Figure A.3 summarizes the general Scout algorithm.

1. Destinations periodically generate a Scout with an increasing sequence number.
2. On receiving a Scout, the router discards the Scout if
   the Scout sequence number is not current
   OR the Scout advertises a path to the current node.
3. The router adds the cost of the incoming link to the cost of the Scout and
   A. If the router has not forwarded a Scout (from the same source) in the last BI, flood the Scout.
   B. Else if Scout is from the designated neighbor, forward the least cost Scout (from the same source) received in the current BI.
   C. Else store the Scout.
4. Update forwarding table to reflect the shortest path.

Figure A.3 : Scout Algorithm Summary for Unrestricted Networks

Scout and Distance Vector Routing Algorithms

Both the Scout and the Distance Vector routing algorithms compute shortest paths between nodes by exchanging messages that convey path cost information. In Scout, if a router receives a Scout message from neighbor Q originated from node A with cost C, the router knows that it can reach A via Q with cost C. Similarly, in DV, if a router receives a DV packet from neighbor Q containing an entry to A with cost C, the router knows that it can reach A via Q with cost C.

However, the method in which each algorithm initiate path computation and propagate routing messages are very different. In Scout, path computation to node A is always initiated by A, via flooding Scout messages. However, in DV, path computation to a node can be initiated by any router. For example, in the event of a link failure, routers connected to the failed link will initiate a path computation to every node whose path used the failed link.

In addition to the differences in initiating path computations, the two algorithms also differ in how routing messages are propagated. In Scout, a Scout message contains path information only to the initiating node. Whereas in DV, a DV packet may contain path information to many nodes. The combination of path aggregation and computation initiation allows DV to efficiently compute all-pairs shortest path. As noted in Chapter 3, computing all-pairs shortest path is not as efficient using Scout.

A.2 Scout Proof of Correctness

The proof of Scout shortest-path convergence for the unrestricted Scout algorithm proceeds by showing that sink trees rooted at R are iteratively transformed into shortest-path trees rooted at R in a number of rounds bounded by the structure of the network. Intuitively this follows because every node eventually receives a Scout from the neighbor with the lowest cost to R, and therefore every node eventually knows the shortest path to R. As stated in Chapter 2, the Scout convergence and its proof apply only in situations where the network state (e.g. topology) changes at a slower rate than Scout's convergence time.

We define SPT as the shortest-path tree rooted at R. SPT has depth K, and SPT_i as the subtree of SPT, also rooted at R, such the depth of SPT_i is i. T_i is the sink tree built from R's broadcast tree built by the Scout algorithm in broadcast interval i. Tree A contains tree B iff B is a subtree of A and A and B share the same root. We prove that $\forall i \in [0, \ldots, K]$, T_i contains SPT_i.

Lemma 1: Assuming T_i contains SPT_i, then for all nodes P $\in T_i \cap SPT_i$, P will not change its forwarding tables in subsequent trees $T_j, j > i$.

Proof: Since node P $\in SPT_i$, P already has the shortest path (in cost and next-hop) to the sink R. Thus in subsequent broadcast rounds of R, P cannot receive a Scout with a lower cost, in fact, P will always receive the same lowest cost from the same neighbor (its designated neighbor). Therefore P will not alter its forwarding table (both cost and next-hop) in subsequent broadcast rounds.

This implies nodes of distance i or less from the root R in T_i will stay in the same position in T_{i+1}.

Theorem 1: Broadcast tree containment: $\forall i \in [0, \ldots, K]$, SPT_i is contained in T_i.

Proof: by induction on the depth of the broadcast tree T_i.

Base: $i = 0$. $SPT_0 = R$, and the root of T_0 is R by definition of sink trees, therefore T_0 contains SPT_0.

Induction: Assume $\forall w < z < K, T_w$ contains SPT_w, proof that T_{z+1} also contains SPT_{z+1}.

Let the set m_z = leaf nodes of(SPT_i). By the induction hypothesis and the algorithm structure, the set of nodes m_z know that in round $z + 1$, they should wait for their parent node in T_z, who will offer them the shortest cost to R. Thus in broadcast interval $z + 1$, nodes in m_z forward only R's least cost Scout to their neighbors. This implies neighbors of node $n \in m_z$ are guaranteed to receive n's shortest path to R. In particular, leaf nodes L = leaf nodes of(SPT_{z+1}) will receive the minimal path cost from nodes in m_z. After the broadcast interval $z + 1$, nodes $l \in L$ deduces the minimal path to R is through its parent node in m_z (by the definition of SPT_{z+1}). So the leaf nodes of T_{z+1} are attached in to the same nodes in SPT_{z+1}. And from lemma 1, interior nodes in T_{z+1} preserve the same connections as nodes in T_z; Therefore, T_{z+1} contains SPT_{z+1}.

From the induction when $z = K - 1$, T_K will contains SPT_K, and $SPT_K = SPT$. Since SPT contains all nodes in the network, by the tree containment definition, T_K must equal SPT. Thus after K broadcast rounds of R, the Scout algorithm is guaranteed to have computed the shortest path to R. When $z > K$, T_z obviously contains SPT from lemma 1.

Intuitively, non-uniform link costs and link delays cause the Scout algorithm more broadcast intervals to convergence because a link can advertise a high cost in the reverse direction (to R) but have very fast forward propagation (away from R). When this occurs, node P will initially flood the costly Scout. The extra number of broadcast intervals needed to send the minimal cost Scout to P is **exactly** the shortest distance (in links) from P to an ancestor of P on the shortest-path tree who currently knows the minimal cost to R, which is always bounded by K.

Theorem 2: Shortest Path Convergence. The unrestricted Scout algorithm in Figure A.3 computes the single shortest path between every pair of nodes in the network.

Proof: This follows directly from Theorem 1. We proved that the broadcast tree eventually contains the shortest path tree for any given node sending Scouts. Thus when every node in the network generates Scouts, then in a bounded number of rounds, every node will know the shortest path to every other node.

Theorem 3: Scout message bound on convergence. The unrestricted Scout algorithm in Figure A.3 converges to the single shortest path on $O(K)$ BI's, with no more than $O(L)$ messages per round per node. Where L is the number of links in the network.

Proof: The first bound on the number of BI's for shortest path convergence is proven above. The second follows directly from the algorithm. Since every nodes can only forward one Scout per round, at most $O(L)$ Scouts can be forwarded per round per node.

A.3 Scout Summary

This chapter presented and proved the correctness of Scout, a destination initiated, selective shortest path routing algorithm. A destination initiates Scout path computation to itself by flooding Scout messages, which discovers path back to the initiating destination. Furthermore, Scout messages are small, fixed-sized and therefore can be hop-by-hop piggybacked onto data packets, largely defraying their costs to the network.

The main features of the Scout routing algorithm are

1. Independent and uncorrelated path computation

2. Destination controlled and initiated Scout messages

Scout is independent because Scout messages initiated by a destination calculate paths ONLY to that destination and is uncorrelated because the time in which a destination initiates its Scouts is determined only by the destination. Thus, Scout path computations to different destinations will occur at different times. This is in contrast to traditional LS and DV routing algorithms where all-pairs path computation occur at the same time.

Because Scout is destination initiated, the amount of Scouts injected into the network is determined by destinations. On the other hand, LS and DV's routing traffic is triggered by network changes. The advantage of this triggering property is that in a relatively static network environment, very little routing overhead is incurred. The disadvantage is that during high rates of network changes, the LS and DV routing overhead are hard to predict and control. With Scout, the amount of routing messages are not triggered and thus do not depend on network changes. This non-triggering in Scout implies that during low rates of network change, many Scout messages are sent unnecessarily (because the shortest paths did not change); On the other hand, it also means that during high rates of change, Scout messages are predictable and can be easily controlled.

The principal disadvantage of the Scout algorithm is that individual Scout messages do not aggregate path computation; thus computing all-pairs shortest paths in a relatively network static environment (such as the Internet) requires more routing message overhead than LS or DV routing algorithms. Again, the reason LS and DV performs better in this scenario is that 1) path computation is aggregated (the two algorithms always computes all-pairs shortest path) and 2) path computation are triggered by network changes (a relative static network has infrequent network changes).

Bibliography

[1] Intermediate System to Intermediate System intra-domain routing exchange protocol for use in conjunction with the protocol for providing the connectionless-mode network service, ISO 8473. ISO DP 10589, February 1990.

[2] Internet traffic archive traces. http://ita.ee.lbl.gov/html/traces.html.

[3] PNNI draft specification. ATM Forum 94-0471R13, 1994.

[4] B. Acevedo, L. Bahler, E. N. Elnozahy, V. Ratan, and M. E. Segal. Highly available directory services in DCE. In *In Proceedings of the Twenty-Sixth Annual International Symposium on Fault-Tolerant Computing (FTCS-26)*, pages 387–391, June 1996.

[5] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows*. Prentice-Hall Inc., Englewood Cliffs, NJ, 1993.

[6] V. Ahuja. Routing and flow control in systems network architecture. *IBM Systems Journal*, 18(2):298–314, 1979.

[7] G. Alkin and F. Baker. RIP version 2 MIB extension. RFC 1724, Xylogics, Inc., Cisco Systems, November 1994.

[8] Tom Anderson and John Zahorjan. Detour research homepage, 1999. http://www.cs.washington.edu/research/networking/detour/.

[9] G. Apostolopoulos, R. Guerin, and S. Kamat. Implementation and performance measurements of QoS routing extensions to OSPF. In *Proceedings of IEEE INFOCOM*, April 1999.

[10] G. Apostolopoulos, R. Guerin, S. Kamat, and S. K. Tripathi. On reducing the processing cost of on-demand QoS path computation. *Journal of High Speed Networks*, 7(2):77–98, 1998.

[11] George Apostolopoulos, Roch Guerin, Sanjay Kamat, and Satish K Tripathi. Quality of Service routing: A performance perspective. In *Proceedings of ACM SIGCOMM*, pages 17–28, September 1998.

[12] Mohit Aron. Analysis of TCP Performance over ATM Networks. Master's thesis, Rice University, December 1997. http://www.cs.rice.edu/˜aron/papers/ms-thesis.ps.

[13] A. A. Assad. Models for rail transportation. In *Transportation Research*, pages 205–220, 1980.

[14] A. A. Assad. Solving linear multicommodity flow problems. In *Proceedings IEEE International Conference on Circuits and Computers*, pages 157–161, 1980.

[15] S. C. Baade. SNA route generation using traffic pattern. *IBM Systems Journal*, 30(3):250–258, 1991.

[16] Saewoong Bahk and Magda El Zarki. Dynamic multi-path routing and how it compares with other dynamic routing algorithms for high speed wide area networks. In *Proceedings of ACM SIGCOMM*, pages 53–64, 1992.

[17] Sandeep Bajaj, Lee Breslau, and Scott Shenker. Uniform versus priority dropping in layered video. In *Proceedings of ACM SIGCOMM*, pages 131–143, September 1998.

[18] Hari Balakrishnan, Venkata N. Padmanabhan, and Randy H. Katz. The Effects of Asymmetry on TCP Performance. In *Proceedings of 3rd ACM Conference on Mobile Computing and Networking*, September 1997.

[19] Sara Basse. *Computer Algorithms.* Addison-Wesley Publishing Company, Reading, Mass., 1988.

[20] R. Beckers, J. L. Deneuborg, and S. Goss. Trails and U turns in the selection of a path by the ant *lasius niger*. *Journal of Theoretical Biology*, 159:397–415, 1992.

[21] Jochen Behrens and J. J. Garcia-Luna-Aceves. Distributed, scalable routing based on Link-State vectors. In *Proceedings of ACM SIGCOMM*, pages 136–147, 1994.

[22] R. E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, N.J., 1957.

[23] Dimitri Berksekas and Robert Gallager. *Data Networks, Second Edition*. Prentice-Hall, Englewood Cliffs, N.J., 1992.

[24] D. Bertsekas and R. Gallagher. *Data Networks*. Prentice-Hall, Englewood Cliffs, CA., 1987.

[25] D. P. Bertsekas and J. N. Tsitsiklis. Parallel and distributed computation: Numerical methods. Prentice-Hall, Englewood Cliffs, N.J., 1989.

[26] S. Bradner and A. Mankin. The recommendation for the next generation IP, January 1995. Internet Request for Comments (RFC) 1752.

[27] Lawrence Brakmo and Larry Peterson. TCP Vegas: End-to-end congestion avoidance on a global Internet. *IEEE Journal on Selected Areas in Communications*, 13(8):1465–1480, October 1995.

[28] Lawrence S. Brakmo, Andrew C. Bavier, and Larry L. Peterson. *x*-Sim user's manual, 1996. http://www.cs.arizona.edu/classes/cs525/xsim/xsim.html.

[29] Lee Breslau and Scott Shenker. Best-effort versus reservation: A simple comparative analysis. In *Proceedings of ACM SIGCOMM*, pages 3–16, September 1998.

[30] V. Cerf and R. Kahn. A protocol for packet network intercommunication. *IEEE Transactions on Communications*, 22(5):637–648, May 1974.

[31] Chi-Tsong Chen. *Linear System Theory and Design*. The Dryden Press, Saunders College Publishing, 1984.

[32] Johnny Chen, Peter Druschel, and Devika Subramanian. A new approach to routing with dynamic metrics. Technical report, Rice University. CS-TR 98-321, 1998.

[33] Johnny Chen, Peter Druschel, and Devika Subramanian. A simple, practical distributed multi-path routing algorithm. Technical report, Rice University. CS-TR 98-320, 1998.

[34] S. Chen and K. Nahrsted. An overview of Quality of Service Routing for next-generation high-speed networks: problems and solutions. *IEEE Networks*, pages 64–79, November/December 1998.

[35] C. Cheng. A loop-free extended Bellman-Ford routing protocol without bouncing effect. *ACM Computer Communication Review*, 19(4):224–236, 1989.

[36] C. Cheng, S. P. R. Kumar, and J. J. Garcia-Luna-Aceves. A distributed algorithm for finding K disjoint paths of minimum total length. In *In Proceedings of the 28th Annual Allerton Conference on Communication, Control, and Computing, Urbana, Illinois*, Oct. 1990.

[37] A. A. Chien and J. H. Kim. Planar-adaptive routing : low-cost adaptive networks for multiprocessors. *Journal of the Association of Computing Machinery*, 42(1):91–123, Jan 1995.

[38] B. Chinoy. Dynamics of Internet Routing Information. In *Proceedings of ACM SIGCOMM*, pages 45–52, September 1993.

[39] Isreal Cidon and Raphael Rom. Multi-path routing combined with resource reservation. In *Proceedings of IEEE INFOCOM*, pages 92–100, 1997.

[40] David D. Clark, Scott Shenker, and Lixia Zhang. Supporting real-time applications in an integrated services packet network. In *Proceedings of ACM SIGCOMM*, pages 14–26, August 1992.

[41] R. Coltun. OSPF: An Internet routing protocol. *ConneXions*, 3(8):19–25, 1989.

[42] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, Mass., 1991.

[43] E. Crawley, R. Nair, B. Rajagopalan, and H. Sandick. A framework for QoS-based routing in the Internet. Internet Draft, QoS Routing Working Group, Internet Engineering Task Force, expires October 1998.

[44] J. D. Day. The (un)revised OSI reference model. *Computer Communication Review*, 25:39–55, Oct. 1995.

[45] Jose Augusto de Azevedo, Joaquim Joao E. R. Silvestre, Madeira Ernesto Q. Vieira Martins, and Filipe Manuel A. Pires. A computational improvement for a shortest paths ranking algorithm. *European Journal of Operational Research*, 73:188–191, 1994.

[46] Stephen E. Deering. IP multicast and the MBone: Enabling live, multiparty, multimedia communication on the Internet. Internet-Draft, draft-ietf-manet-aodv-00.txt, November 1997. Work in progress.

[47] Stephen E. Deering. Multicast routing in internetworks and extended LANs. In *Proceedings of ACM SIGCOMM*, pages 89–101, 1988.

[48] Mikael Degermark, Andrej Brodnik, Svante Carlsson, and Stephen Pink. Small forwarding tables for fast routing lookups. In *Proceedings of ACM SIGCOMM*, pages 3–14, September 1997.

[49] A. DeHon, T. Knight, and H. Minsky. Fault-tolerant design for multistage routing networks. In *Proceedings of the International Symposium on Shared Memory Multiprocessing*, pages 60–71, 1991.

[50] Andrer DeHon, Frederic Chong, Matthew Becker, Eran Egozy, Henry Minsky, Samuel Peretz, and Tomas Knight Jr. METRO: A router architecture for high-performance, short-haul routing networks. *Proceedings the 21st Annual International Symposium on Computer Architecture*, pages 266–77, 1994.

[51] M. DeMarco and A. Pattavina. Distributed routing protocols for ATM extended banyan networks. *IEEE Journal on Selected Areas in Communications*, 15(5):925–37, June 1997.

[52] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queuing algorithm. In *Proceedings of ACM SIGCOMM*, pages 1–12, 1989.

[53] Roy C. Dixon and Daniel A. Pitt. Addressing, bridging and source routing. *IEEE Networks*, 2(1):25–32, Jan 1988.

[54] E. W. Djikstra. A note on two problems in connection with graphs. *Numerische Mathematik*, 1, 1959.

[55] J. T. Draper and J. Ghosh. Multipath E-cube algorithms (MECA) for adaptive wormhole routing and broadcasting in K-ary N-cubes. In *Sixth International Parallel Processing Symposium*, pages 470–10, 1992.

[56] Rohit Dube, Cynthia D. Rais, Kuang-Yeh Wang, and Satish K. Tripathi. Signal stability based adaptive routing (SSA) for ad-hoc mobile networks. *IEEE Personal Communications*, February 1997.

[57] J. B. Postel (editor). Internet Protocol. Internet Request For Comments (RFC) 729, September 1981.

[58] David Eppstein. Finding the k shortest paths. In *Proc. 35th Symp. Foundations of Computer Science*, pages 154–165. Inst. of Electrical & Electronics Engineers, November 1994.

[59] Deborah Estrin, Yakov Rekhter, and Steven Hotz. Scalable inter-domain routing architecture. In *Proceedings of ACM SIGCOMM*, pages 40–52, 1992.

[60] Michalis Faloutsos, Anindo Banerjea, and Rajesh Pankaj. QoSMIC: Quality of Service Sensitive Multicast Internet ProtoCol. In *Proceedings of ACM SIGCOMM*, pages 144–153, September 1998.

[61] Domenico Ferrari, Anindo Banerjea, and Hui Zhang. Network support of multimedia: A discussion of the tenet approach. *Computer Networks and ISDV Systems*, 10:1267–1280, July 1994.

[62] L. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, N.J., 1962.

[63] L. Fratta, M. Garla, and L. Kleinrock. The flow deviation method: An approach to store-and-forward communication network design. *IEEE Networks*, 3(2):97–133, 1973.

[64] L. Fratta, M. Gerla, and L. Kleinrock. Computer communication network design - experience with theory and practice, 1972. AFIPS conference proceedings SJCC, Atlantic City, New Jersey.

[65] V. Fuller, T. Li, J. Yu, and K. Varadhan. Classless interdomain routing (CIDR): An address assignment and aggregation strategy. RFC 1519, September 1993.

[66] J. J. Garcia-Luna-Aceves. A unified approach for loop-free routing using link states or distance vectors. *ACM Computer Communication Review*, 19(4):212–223, 1989.

[67] Marianne L. Gardner, Ina S. Loobeek, and Stephen N. Cohn. Type of service routing with loadsharing. In *IEEE/IECE Global Telecommunications Conference*, pages 1144–50, 1987.

[68] P. Georgatsos and D. Griffin. A management system for load balancing through adaptive routing in multiservice ATM networks. In *Proceedings of IEEE INFOCOM*, pages 863–870, 1996.

[69] R. Govindan and A. Reddy. An analysis of interdomain topology and route stability. In *Proceedings of IEEE INFOCOM*, April 1997.

[70] J. P. Gray and T. B. McNeill. SNA multiple-system networking. *IBM Systems Journal*, 18(2):263–297, 1979.

[71] R. Guerin, S. Kamat, V. Peris, and R. Rajan. Scalable QoS provision through buffer management. In *Proceedings of ACM SIGCOMM*, pages 29–40, September 1998.

[72] R. Guerin, A. Orda, and D. Williams. QoS routing mechanisms and OSPF extensions. In *Proceedings of the IEEE GLOBECOM*, November 1997.

[73] R. Guerin, D. Williams, and A. Orda. QoS routing mechanisms and QSPF extensions. In *Proceedings of the IEEE GLOBECOMM*, November 1997.

[74] C. Hedrick. Routing information protocol, 1988. Internet Request for Comments (RFC) 1058.

[75] C. Hedrick. Use of OSI IS-IS for routing in TCP/IP and dual environments, 1988. Internet Request for Comments (RFC) 1195.

[76] M. A. Henrion, G. J. Eilenberger, G. H. Petit, and P. H. Parmentier. A multipath self-routing switch. *IEEE-Communications Magazine*, 31(4):46–52, April 1993.

[77] W. C. Lee M. G. Hluchyj and P. A. Humblet. Routing subject to Quality of Service constraints in integrated communications networks. *IEEE Networks*, pages 46–55, July 1995.

[78] Janey C. Hoe. Improving the start-up behaviour of a congestion control scheme for TCP. In *Proceedings of ACM SIGCOMM*, 1996.

[79] B. Holldobler and E. O. Wilson. *Journey to the Ants*. Bellknap Press/Harvard University Press, 1994.

[80] International Standards Organization. Intra-domain IS-IS routing protocol. ISO/ICE JTC1/SC6 WG2 N323, Sept, 1989.

[81] K. Ishida, Y. Kakuda, and T. Kikuno. A routing protocol for finding two node-disjoint paths in computer networks. In *Internation Conference on Network Protocols*, pages 340–347, Nov. 1992.

[82] A. Itah and M. Rodeh. The multi-tree approach to reliability in distributed networks. In *Proceedings of the 25th Symposium on FOCS*, 1984.

[83] Van Jacobson. Congestion avoidance and control. In *Proceedings of ACM SIGCOMM*, pages 314–32, August 1988.

[84] Van Jacobson. Berkeley TCP evolution from 4.3-Tahoe to 4.3-Reno. In *Proceedings of the Eighteenth Internet Engineering Task Force*, August 1990.

[85] J. M. Jaffe, F. H. Moss, and R. A. Weingarten. SNA routing: Past, present, and possible future. *IBM Systems Journal*, 22(4):417–434, 1983.

[86] Sugih Jamin, Peter Danzig, Scott Shenker, and Lixia Zhang. A measurement-based admission control algorithm for integrated services packet networks. *IEEE/ACM Transactions on Networking*, 5(1):56–70, February 1997.

[87] David B. Johnson and David A. Maltz. Dynamic source routing in ad hoc wireless networks. In Mobile Computing, edited by Tomasz Imielinski and Hank Korth, Chapter 5, pages 153-181, Kluwer Academic Publishers, 1996.

[88] Youn Chan Jung and Chong Kwan Un. Banyan multipath self-routing ATM switches with shared buffer type switch elements. *IEEE-Transactions on Communications*, 43(11):2847–57, Nov 1995.

[89] J. L. Kennington and R. V. Helgason. *Algorithms for Network Programming*. Wiley-Interscience, New York, 1980.

[90] S. Keshav and R. Sharma. On the efficient implementation of fair queuing. *Journal of Internetworking: Research and Experience*, 2(3), September 1991.

[91] S. Keshav and R. Sharma. Achieving quality of service through network performance management. In *Proceedings of NOSSDAV*, July 1998.

[92] S. Keshav and R. Sharma. Issues and trends in router design. *IEEE-Communications Magazine*, May 1998.

[93] Srinivasan Keshav. *An Engineering Approach to Computer Networking*. Addison-Wesley Publishing Company, Reading, Mass., 1997.

[94] Atul Khanna and John Zinky. The revised ARPANET routing metric. In *Proceedings of ACM SIGCOMM*, pages 45–56, September 1989.

[95] E. Knightly and H. Zhang. D-BIND: an accurate traffic model for providing QoS guarantees to VBR traffic. *IEEE/ACM Transactions on Networking*, 5(2):219–231, April 1997.

[96] Ram Krishnan and John A. Silvester. Choice of allocation granularity in multipath source routing schemes. In *Proceedings of IEEE INFOCOM*, pages 322–329, 1993.

[97] Satish Kumar, Pavlin Radoslavov, David Thaler, Cengiz Alaettinoglu, Deborah Estrin, and Mark Handley. The MASC/BGMP architecture of Inter-domain multicast routing. In *Proceedings of ACM SIGCOMM*, pages 94–104, September 1998.

[98] Craig Labovitz, G Robert Malan, and Farnam Jahanian. Internet routing instability. Technical report, University of Mechigan. CSE Technical Report 322-97, 1997.

[99] Craig Labovitz, G. Robert Malan, and Farnam Jahanian. Internet routing instability. In *Proceedings of ACM SIGCOMM*, pages 53–64, 1997.

[100] W. S. Lai. Bifurcated routing in computer networks. *Computer Communication Review*, 15(3):28–49, 1986.

[101] T. V. Lakshman and D. Stiliadis. High speed policy-based packet forwarding using efficient multi-dimensional range matching. In *Proceedings of ACM SIGCOMM*, pages 203–214, September 1998.

[102] E. J. Lawler. *Combinatorial optimization: networks and matroids*. Holt, Rinehart and Winston, 1976.

[103] E. L. Lawler. A procedure for computing the K best solutions to discrete optimization problems and its applications to the shortest path problem. *Management Science*, 18:401–405, 1972.

[104] Eun Seol Lee and Chae Tak Lim. A study on the DRF multipath multistage interconnection network. *Journal of the Korea Institute of Telematics and Electronics*, 27(10):1605–12, Oct 1990.

[105] F. C. Lin and F. H. Wang. Message pattern routing in hypercubes: a distributed-concentrate approach. *Journal of Parallel and Distributed Computing*, 29(1):27–42, Aug 1995.

[106] Neng Pin Lu and Chung Ping Chung. A fault-tolerant multistage combining network. *Journal of Parallel and Distributed Computing*, 34(1):14–28, April 1996.

[107] Q. Ma and P. Steenkiste. On path selection for traffic with bandwidth guarantees. In *Proceedings of IEEE International Conference on Network Protocols*, October 1997.

[108] Udi Manber. *Introduction to Algorithms*. Addison-Wesley Publishing Company, Reading, Mass., 1989.

[109] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. Request for comments (RFC) 2018: TCP selective acknowledgment options, October 1996.

[110] J. M. McQuillan, I. Richer, and E. C. Rosen. The new routing algorithm for the ARPANET. *IEEE Transactions on Communications*, COM-28(5):711–719, 1980.

[111] J. M. McQuillan and D. C. Walden. The ARPANET design decisions. *Computer Networks*, 1, 1977.

[112] D. Mills. Exterior Gateway Protocol formal specification. Request for Comments (RFC) 904, April, 1984.

[113] S. P. Morgan. Queuing disciplines and passive congestion control in byte-stream networks. *IEEE Transactions on Communications*, 39(7):1097–1106, 1991.

[114] J. Moy. The OSPF specification. RFC 1131, October 1989.

[115] J. Moy. Multicast routing extensions for OSPF. *Communications of the ACM*, 37(8):61–66, August 1994.

[116] Andrew Myles, David B. Johnson, and Charles Perkins. A mobile host protocol supporting route optimization and authentication. *IEEE Journal on Selected Areas in Communications, special issue on "Mobile and Wireless Computing Networks"*, 13(5):839–849, June 1995.

[117] Paul Newman and G. Karlsson. IP switching and gigabit networks. *IEEE-Communications Magazine*, January 1997.

[118] R. Ogier and N. Shacham. A distributed algorithm for finding shortest pairs of disjoint paths. In *Proceedings of IEEE INFOCOM*, 1989.

[119] Teunis J. Ott and Neil Aggarwal. TCP over ATM: ABR or UBR? In *Proceedings of the ACM SIGMETRICS*, Seattle, WA, June 1997.

[120] Venkata N. Padmanabhan and Randy H. Katz. TCP Fast Start: A Technique For Speeding Up Web Transfers. In *Proceedings of the IEEE GLOBECOM*, November 1998.

[121] C. Parris, S. Keshav, and D. Ferrari. A framework for the study of pricing in integrated networks. ICSI Technical Report TR-92-016 and AT&T Bell Labs Technical Memorandum TM-920105-03, January 1992.

[122] Vern Paxson. Growth trends in wide area TCP connections. *IEEE Networks*, 8(4):8–17, 1994.

[123] Vern Paxson. End-to-end Internet packet dynamics. In *Proceedings of ACM SIGCOMM*, pages 139–152, September 1997.

[124] Vern Paxson. Measurements and analysis of end-to-end Internet dynamics, April 1997. Ph.D. dissertation, University of California, Berkeley.

[125] Vern Paxson. End-to-End routing behavior in the Internet. *IEEE/ACM Transactions on Networking*, 5(5):43–51, 1998.

[126] Vern Paxson and Sally Floyd. Wide-area traffic: the failure of poisson modeling. In *Proceedings of ACM SIGCOMM*, pages 257–268, August 1994.

[127] Charles Perkins. Ad Hoc On Demand Distance Vector (AODV) routing. Internet-Draft, draft-ietf-manet-aodv-00.txt, November 1997. Work in progress.

[128] Michael Perloff and Kurt Reiss. Improvements to TCP Performance in High-Speed ATM Networks. *Communications of the ACM*, 38(2):90–100, February 1995.

[129] Larry L. Peterson and Bruce S. Davie. *Computer Networks: a Systems Approach*. Morgan Kaufmann Publishers, Inc., San Francisco, CA., 1996.

[130] M. De Prycker. *Asynchronous Transfer Mode: solution for broadband ISDN*. Ellis Horwood, Chichester, England, 1991.

[131] N. S. V. Rao and S. G. Batsell. QoS routing via multiple paths using bandwidth reservation. In *Proceedings of IEEE INFOCOM*, pages 11–18, 1998.

[132] S. M. Reddy and V. P. Kumar. On multipath multistage interconnection networks. In *Proceedings of the 5th International Conference on Distributed Computing Systems*, pages 210–17, 1985.

[133] Y. Rekhter and T. Li. A Border Gateway Protocol 4 (BGP-4). RFC 1771, T.J. Watson Research Center, IBM Corp., Cisco Systems, March 1995.

[134] E. C. Rosen, A. Viswanathan, and R. Callon. Multiprotocol label switching architecture. *draft-itef-mpls-arch-04.txt* Internet Draft, February 1999, work in progress.

[135] H. Saran, S. Keshav, and C. R. Kalmanek. A scheduling discipline and admission control policy for Xunet 2. In *Proceedings of NOSSDAV*, November 1993.

[136] Stefan Savage, Tom Anderson, Amit Aggarwal, David Becker, Neal Cardwell, Andy Collins, Eric Hoffman, John Snell, Amin Vahdat, Geoff Voelker, and John Zahorjan. Detour: a case for informed Internet routing and transport. Technical report, University of Washington. CS Technical Report UW-CSE-98-10-05, 1998.

[137] M. Schwartz. *Telecommunication Networks, Protocols, Modeling and Analysis*. Addison-Wesley, Reading, Mass., 1987.

[138] Mischa Schwartz and Thomas E. Stern. Routing techniques used in computer communication networks. *IEEE Transactions on Communications*, 28(4):539–552, 1980.

[139] A. Segall and J. M. Jaffe. A reliable distributed route set-up procedure. In *IEEE/IECE Global Telecommunications Conference*, 1983.

[140] Seung Woo Seo and Tse Yun Feng. The composite banyan network. *IEEE-Transactions on parallel and distributed systems*, 6(10):1043–1054, Oct 1995.

[141] D. R. Shier. On algorithms for finding the k sortest paths in a network. *Networks*, 9:195–214, 1979.

[142] R. Siamwalla, R. Sharma, and S. Keshav. Discovering Internet topology. Technical report, Cornell University. submitted to Infocom, 1999.

[143] F. Sibai and S. Kulkarni. Performance of multicast wormhole routing algorithms in fault-tolerant 2D meshes. *Seventh International Conference on Parallel and Distributed Computing Systems*, pages 610–613, 1994.

[144] F. N. Sibai and A. A. Abonamah. Parallel path assignment of multicast connections in multi-path networks. In *IEEE Fifteenth Annual International Phoenix Conference on Computers and Communications*, 1994.

[145] F. N. Sibai, N. K. Sharma, and A. A. Abonamah. A simulation study of four reconfiguration algorithms for a multi-path cube-based network. *Transactions of the Society for Computer Simulation*, 10(1):1–21, March 1993.

[146] Deepinder Sidhu, Raj Nair, and Shukri Abdallah. Finding disjoint paths in networks. In *Proceedings of ACM SIGCOMM*, pages 43–51, 1991.

[147] Josep Soel-Pareta, Debapriya Sarkar, Jorg Liebeherr, and Ian F Akyildiz. Adaptive multipath routing of connectionless traffic in an ATM network. In *Proceedings of IEEE INFOCOM*, pages 1626–1630, 1995.

[148] J. Sole-Pareta, D. Sarkar, J. Liebeherr, and I. F. Akyildiz. Adaptive multipath routing of connectionless traffic in an ATM network. *Journal of Network and Systems Management*, 3(4):355–370, 1995.

[149] V. Srinivasan, George Varghese, Subash Suri, and Marcel Valdvogel. Fast scalable level four switching. In *Proceedings of ACM SIGCOMM*, pages 191–202, September 1998.

[150] Martha Steenstrup. *Routing in communications networks*. Prentice-Hall, Fort Collins, CO., 1995.

[151] Ion Stoica, Scott Shenker, and Hui Zhang. Core-stateless fair queuing: A scalable architecture to approximate fair bandwidth allocations in high speed networks. In *Proceedings of ACM SIGCOMM*, pages 118–130, September 1998.

[152] J. W. Surballe and R. E. Tarjan. A quick method of finding shortest pairs of disjoint paths. *Networks*, 4:43–51, 1984.

[153] Hiroshi Suzuki and Fouad A. Tobagi. Fast bandwidth reservation scheme with multi-link & mult-ipath routing in ATM networks. In *Proceedings of IEEE INFOCOM*, pages 2233–2240, 1992.

[154] Andrew S. Tanenbaum. *Computer Networks*. Prentice-Hall, Englewood Cliffs, NJ., 1981.

[155] S. Tiarawut, T. Saito, and H. Aida. A connection-level design of multistage nonblocking ATM switches. *IEICE-Transactions on Communications*, E77-B(10):1203–8, Oct 1994.

[156] D. M. Topkis. A k shortest path algorithm for adaptive routing in communications networks. *IEEE Transactions on Communications*, 36, 1988.

[157] J. N. Tsitsiklis and G. D. Stamoulis. On the average communication complexity of asynchronous distributed algorithms. MIT LIDS-report LIDS-P-2238, May 1986.

[158] K. Varadhan, R. Govindan, and D. Estrin. Persistent routing oscillations in Inter-domain routing. USC/ISI, Available at Routing Arbiter project's home page.

[159] C. Villamizer, R. Chandra, and R. Govindan. *draft-IETF-idr-route-dampen-00-preview*, 1996. Internet Engineering Task Force Draft, July 21, 1995.

[160] Marcel Waldvogel, George Varghese, Jon Turner, and Bernhard Plattner. Scalable high speed IP routing lookups. In *Proceedings of ACM SIGCOMM*, pages 25–36, September 1997.

[161] Mu Cheng Wang, H. J. Siegel, M. A. Nichols, and S. Abraham. Using a multipath network for reducing the effects of hot spots. *IEEE-Transactions on Parallel and Distributed Systems*, 6(3):252–68, March 1995.

[162] Z. Wang and J. Crowcroft. Quality of Service routing for supporting multimedia applications. *IEEE Jounal Selected Areas in Communications*, 14(7):1228–1234, 1996.

[163] Zheng Wang and Jon Crowcroft. Analysis of shortest-path routing algorithms in a dynamic network environment. In *ACM SIGCOMM Computer Communication Review*, pages 63–71, 1992.

[164] I. Widjaja and Leon Garcia. The helical switch: a multipath ATM switch which perserves cell sequence. *IEEE-Transactions on Communications*, 42(8):2618–29, Aug 1994.

[165] J. Y. Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17:712–716, 1971.

[166] Zhou Yigquan and Min Yinghua. A kind of multistage interconnection networks with multiple paths. *Journal of Computer Science and Technology*, 11(4):395–404, July 1996.

[167] Lixia Zhang, Steve Deering, Deborah Estrin, Scott Shenker, and Daniel Zappala. RSVP: A new resource reservation protocl. *IEEE Network Magazine*, 7(5):8–18, September 1993.

[168] Lixia Zhang, Scott Shenker, and David D. Clark. Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic. In *Proceedings of ACM SIGCOMM*, pages 133–148, 1991.

[169] Z. Zheng and J. Crowcroft. QoS Routing for supporting resource reservation. In *IEEE JSAC*, September 1996.