ABSTRACT

EFFICIENT DYNAMIC ROUTING IN WIDE-AREA NETWORKS

by
Anees A. Shaikh

Chair: Kang G. Shin

The tremendous growth of the global Internet has given rise to a variety of applications that require quality-of-service (QoS) beyond what is provided by the current best-effort IP packet delivery service. In addition, traditional static shortest-path routing offers little flexibility in managing the distribution of traffic in the network. Dynamic or QoS routing protocols offer the advantage of choosing paths that meet application QoS requirements and balance the load in the network. These benefits come at the cost, however, of additional consumption of computational and bandwidth resources at levels higher than conventional routing protocols. More specifically, dynamic routing performs poorly when the network state used to compute paths is out-of-date, requiring that network information be distributed very frequently to achieve good results.

This dissertation undertakes a detailed study of dynamic and QoS routing with a focus on understanding and controlling the performance-overhead trade-offs inherent in these protocols. We develop a practically motivated model that clearly formulates dynamic routing in terms of a set of route computation algorithms, routing and update policies, and network and traffic configurations. The model is implemented in a custom-designed simulation environment that allows experimentation with a wide variety of configurations.

We use the model and simulation environment to perform in-depth experiments that uncover the influences of a variety of policy and configuration parameters on the performance and resulting overheads. In particular, the evaluation focuses on the effects of out-of-date information about network state on routing performance. A main contribution of this effort is a set of specific guidelines and observations that assist service providers in configuring the network and setting policies for dynamic or QoS routing.

With the insights collected from the evaluation effort, the dissertation develops novel algorithms

for reducing the computational overheads of QoS routing. These mechanisms allow precomputed QoS routes to efficiently capitalize on the latest state information available. As a result, precomputation is able to offer performance competitive with the much more costly on-demand computation policy. Finally, the dissertation introduces hybrid routing, a new routing scheme for IP networks that limits dynamic routing to a subset of the network traffic. By dynamically routing only long-lived flows, hybrid routing is able to improve the stability and efficiency of load-sensitive routing while also providing automated load-balancing capability that reacts to fluctuations in network traffic.

EFFICIENT DYNAMIC ROUTING IN WIDE-AREA NETWORKS

by

Anees A. Shaikh

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
1999

Doctoral Committee:

Professor Kang G. Shin, Chair
Associate Professor Farnam Jahanian
Assistant Professor Sugih Jamin
Assistant Professor Kimberly Wasserman
Jennifer Rexford, Technical Staff Member, AT&T Labs–Research

To my wife
and
my parents

ACKNOWLEDGEMENTS

بسم الله الرحمن الرحيم

In the name of God, most merciful, most beneficent

This dissertation marks a milestone that would have been unreachable without the care and support given to me by many individuals.

Any success I have attained during my academic career I attribute to my parents, Dr. Abdul Quader and Fatema Shaikh. They encouraged me in every endeavor and taught me to value education. My mother is my staunchest promoter and my father remains my consummate role model. I thank my sister Tasneem, who covered for me at home, and kept me (and herself) entertained with late-night phone calls. Thanks to my cousins Muzammil, Taher, Yusuf, and their families, who dropped everything to help me when I needed them. I am also grateful for the support and affection of my wife's parents, Dr. Muhammad Razzak and Momtaz Sarker, and her siblings Shazia and Adil.

My advisor, Professor Kang G. Shin, has continually motivated and supported me through my career at Michigan. I am very grateful to him for creating an outstanding environment in which I was provided with numerous resources, surrounded by talented colleagues, and given the right mix of freedom and guidance.

I wish to thank Professors Farnam Jahanian, Sugih Jamin, Kim Wasserman, as well as Dr. Jennifer Rexford, for serving on my doctoral committee. I gratefully acknowledge financial support of my research from the Defense Advanced Research Projects Agency, the National Science Foundation, and the Office of Naval Research.

Through the course of my doctoral studies, I benefited tremendously from my interaction and collaboration with several individuals. In mentoring and advising me, Jennifer Rexford has shown enormous patience, dedication, and kindness. My collaboration with her at Michigan and AT&T Labs–Research fostered my intellectual growth and edified me in the art of conducting research. I cannot imagine working with someone more gifted.

I am also grateful to Ashish Mehra who gave his time freely to advise, educate, and entertain me. Scott Dawson never turned away my countless systems questions, and taught me the value (and methods) of figuring things out on my own. Thanks also to Farnam who contributed many hours to serve as a collaborator, friend, and source of good advice and great humor.

Thanks to Ashish Mehra, Scott Dawson, Jen Rexford, Wu-chang Feng, Wu-chi Feng, Anurag Bajaj, Todd Mitton, and Steve Vlaovic. I am fortunate to have found such caring and generous friends. I am also grateful to Real-Time Computing Lab members Tarek Abdelzaher, Rob Malan, Mike Bailey, Sung-Whan Moon, and Sunghyun Choi for their friendship and advice, and to B.J. Monaghan for helping to smooth the path.

I was truly blessed to be a part of the Muslim Community Association of Ann Arbor and my small role in serving the community was my privilege. My time with my brothers and sisters in the MCA enriched me and taught me so much that I needed to know.

Finally, I thank my loving wife Zakia, who stood by me selflessly as I pursued my goals. She is my partner and my most ardent supporter. Having her, and now Salma, by my side makes this all the more meaningful.

TABLE OF CONTENTS

LIST OF TABLES

Table

LIST OF FIGURES

Figure

CHAPTER 1

INTRODUCTION

Recent years have witnessed tremendous growth of the global Internet, both in its size and subscriber population, and also in the variety of ways it is used. Consumer applications such as streaming live video, packetized voice, multiplayer games, and World Wide Web-based shopping, are now commonplace. Businesses also depend on the network for providing electronic storefronts, support and service to their customers, and a means to conduct day-to-day operations. The applications that deliver these new services introduce new traffic characteristics and impose new requirements on network performance, reliability, and availability. Yet, the Internet's fundamental service consists only of a packet delivery system that makes no promises regarding reliability, timeliness, or in-order delivery. The network follows a "best-effort" paradigm in which all packets are treated identically, regardless of the user application. Supporting these new types of applications requires more sophisticated mechanisms for link scheduling, buffer management, and route selection, all of which play an important role in meeting the new demands on the network.

Additional burdens are placed on the Internet service provider (ISP) who often must guarantee some aggregate level of performance and availability to customers and peer providers through service level agreements (SLAs). This requires that ISPs improve the quality-of-service (QoS) delivered to customers. Examples of QoS include guarantees on network delay, throughput, or loss, for either individual application flows, or groups of flows. QoS may be "hard" or "soft," implying deterministic or statistical guarantees, respectively. While providing QoS for customers is an important goal, equally important to the ISP is the efficient utilization of the network to reduce costs and improve manageability. To achieve this goal, ISPs must engage in "traffic engineering" to optimize and maintain network resources, assure that SLAs are honored, and respond promptly to problems when they arise.

Further limiting the ability of ISPs to address these QoS and traffic engineering problems is the current static routing used in the Internet. Packets are forwarded on preconfigured paths without regard for application QoS requirements and independent of the current load conditions in the network. Load-sensitive routing was incorporated early in the development of circuit-switched telecommunication networks, and plays a crucial role in its robustness to failures and unexpected changes in load. Though dynamic routing was deployed for a short time in early incarnations of the Internet (e.g., the ARPANET), it was ultimately abandoned due to its overheads and complexity, especially given the absence of applications requiring better than best-effort service. Only recently, due to the arrival of new applications described above, have research and standardization efforts returned to dynamic routing as a key technique to manage traffic and deliver improved application performance. By selecting paths that circumvent congested links, dynamic routing can balance network load and improve network utilization. In addition, variations on dynamic routing (e.g., QoS routing) explicitly identify paths that can provide increased application QoS.

In this dissertation we undertake a detailed investigation of protocols for dynamic routing. While these protocols contribute significantly toward meeting the challenges outlined above, their deployment in IP networks has been hampered, due primarily to their complexity and consumption of network and computational resources. Central to our study, therefore, is an understanding of the performance versus overhead trade-offs of load-sensitive routing in realistic network settings and subject to a variety of routing policies. With these insights, we proceed to develop and evaluate practical approaches for dynamic routing that enhance its deployability by reducing overhead while maintaining its advantages.

The remainder of this chapter reviews routing in the current Internet architecture, and illustrates the effectiveness of dynamic routing in addressing some limitations of conventional shortest-path routing. Finally, Section 1.3 provides an overview of the dissertation.

1.1 Internet Routing Architecture

The contemporary Internet consists of a collection of service providers that allow their customers to connect from various locations called points of presence (POPs) [40]. As shown in the example configuration in Figure 1.1, residential customers may access the network via a bank of dial-in modems while businesses may have (multiple) direct connections to the provider network via leased lines of varying bandwidth. These customers typically connect to the network through

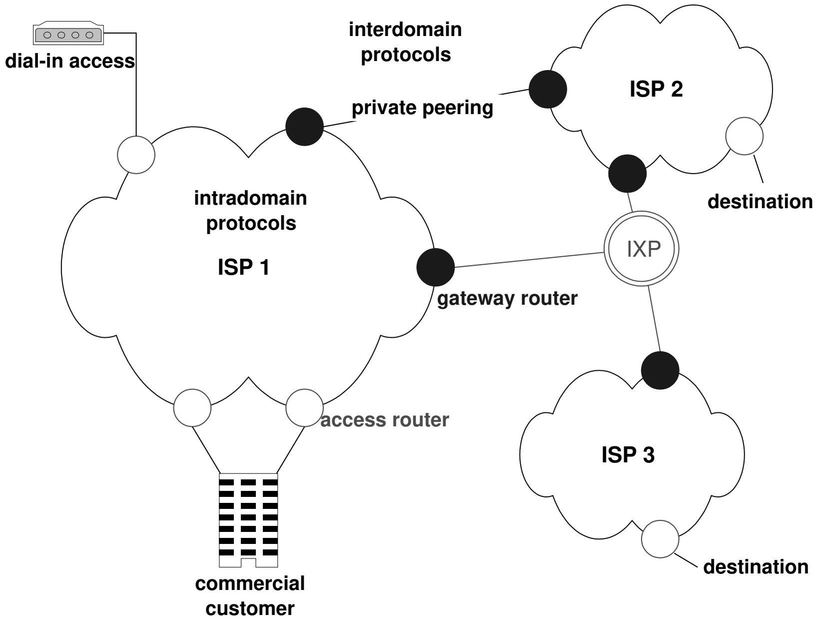


Figure 1.1: Internet routing architecture

access routers. To allow users to reach parts of the Internet outside the ISP, network providers can attach to Internet exchange points (IXPs). IXPs are high-speed switches or networks where ISPs place routers to enable exchange of traffic and routing information with other providers. In addition to connecting to the exchange point, ISPs may form private peering agreements with other providers to improve performance or reliability. These agreements may also specify the ability to send transit traffic (i.e., traffic with both source and destination outside the peer) through the peer network. ISP connections to other networks or IXPs are typically via gateway routers (also called border routers).

For the purposes of routing, the Internet consists of separate administrative domains called autonomous systems (AS) or domains. In Figure 1.1, each ISP network may be comprised of one or more ASs. A company or campus network under a single technical administration is also considered an AS. Within a domain, routing is done with interior gateway protocols (IGPs). Routing between AS domains is done with exterior gateway protocols (EGPs) where each AS appears as a single network entity. The separation into distinct administrative domains improves network manageability and allows ISPs to apply whatever intradomain routing protocol and policy they wish. Communication with other networks is independent of the IGP running within the AS.

As an example of conventional Internet routing, consider a packet sent by a dial-up customer of ISP 1 in Figure 1.1 to a destination attached to ISP 2. It first enters the network via an access router and reaches a gateway router on a path computed by the IGP in use by ISP 1. The choice

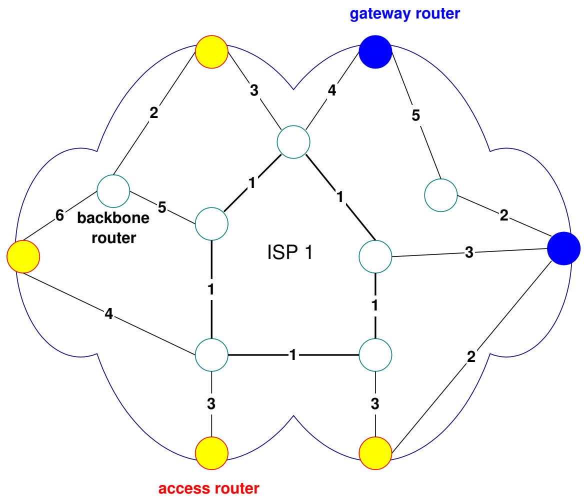


Figure 1.2: Intradomain routing architecture

of which gateway router to use is determined by the EGP. Across domains, the packet is forwarded to a gateway router in ISP 2 according to reachability information computed by the EGP. Once the packet enters ISP 2, it is forwarded to the appropriate destination using routing tables computed by the IGP in ISP 2.

Figure 1.2 shows an example configuration within a single network domain using a link-state protocol (e.g., OSPF or IS-IS), the dominant type of IGP used in the current Internet. Inside the domain, access routers and gateway routers are interconnected via a set of backbone routers. Each link is manually assigned a preconfigured cost that may reflect the capacity or delay, for example. Each router in the network distributes information about the cost of its incident links to all other routers in the domain. Since the costs are relatively static quantities, this link-state information need not be distributed frequently, thus limiting the control and computational overheads. Using the received information, each router computes the shortest path to every other node where distance is in terms of the link costs. Routers recompute paths relatively infrequently, for example only when new link-state information is received. These routes are stored in a next-hop forwarding table so that when a packet arrives, the router simply looks up the destination in the table and forwards the packet on the corresponding interface.

1.2 Advantages of Dynamic Routing

While the conventional intradomain routing described above is relatively simple, and exhibits low overhead, it offers little flexibility in managing network traffic. At best, an ISP may set link costs according to some notion of an expected traffic pattern such that traffic is distributed evenly throughout the network. But when the volume of traffic between particular points shifts unexpectedly, network load may become significantly imbalanced, leading to poor performance and utilization. These fluctuations may arise, for example, due to variations in user demand and changes in the network configuration, including failures or reconfigurations in the networks of other service providers. Even when the network is well-provisioned for the average case traffic pattern, stochastic fluctuations can still cause poor utilization for short time periods. Network providers rely on coarse measurement tools to discover performance problems in the network and when the problem requires adjustment of the network configuration, the ISP typically has to manually reroute traffic.

These challenges have spurred increased interest in dynamic routing as a tool for managing network traffic and providing QoS guarantees. By selecting paths based on link utilization, dynamic routing responds to long and short timescale fluctuations in the traffic pattern, and automates the process of redirecting traffic. In doing so, it balances network load and improves overall network utilization. Furthermore, choosing routes based on resource availability rather than static link weights provides the ability to satisfy per-flow QoS requirements and improve application performance.

Despite these potential advantages, however, most backbone networks still employ static link-state routing (e.g., based on routing protocols such as OSPF). Unlike static routing, load-sensitive routing algorithms require accurate and frequently distributed link-state information to make good routing decisions. Dynamic routing is particularly sensitive to link-state staleness, and suffers *route flapping* when staleness is excessive. These oscillations occur when out-of-date information leads routers to direct most traffic to a seemingly attractive path while an alternative path lies underutilized. A new update arriving to correct the view causes the router to redirect all traffic to the underutilized path, reversing the roles of the routes. In addition, flapping is exacerbated when flows come and go quickly, since the rapid fluctuations degrade the usefulness of link-state information in predicting conditions in the network. Frequent distribution of link-state information prevents oscillation, but runs the risk of overwhelming the network with control traffic. Similar issues apply to route computation. Accurate route selection requires that routers compute paths using the latest link-state information frequently (usually with more sophisticated and complex algorithms), which introduces higher computational overhead.

Recent research has focused primarily on three areas of dynamic and QoS routing. Theoretical work proposes new algorithms for QoS routing that optimize multiple QoS metrics (e.g., delay and throughput), or compute multicast routing trees subject to QoS requirements. Performance evaluation work compares the performance of several route selection algorithms, most often under specific network and traffic configurations. Finally, protocol development efforts consider issues such as link-state distribution policies, path set-up mechanisms, and integration into existing intradomain routing protocols.

This dissertation draws on the work done in each of these communities but distinguishes itself by evaluating and controlling the effects of out-of-date link-state information. Its primary focus is on understanding and managing the fundamental trade-offs in deploying dynamic or QoS routing in a single ISP domain. This requires development of an abstract, yet realistic, model of the behavior of QoS routing, as well as a simulation environment for evaluating its performance and overheads. The dissertation describes results from in-depth experiments that uncover the influences of traffic characteristics, network topology, routing algorithm, and link-state update policies on the cost-performance trade-offs. With the insights collected from the evaluation effort, the dissertation continues with novel algorithms for reducing the computational overheads of QoS routing, and culminates with a new routing scheme that maintains the benefits of dynamic routing, while also making it both stable and efficient.

1.3 Dissertation Overview

Chapter 2 begins with a broad overview of routing techniques in wide area networks. It surveys protocols used in current IP networks, as well as ATM routing. In addition, it reviews early dynamic routing protocols, in particular those used in the ARPANET. Following this is a description of specific models for QoS in internetworks and some examples of recent algorithms and protocols for dynamic routing, both for QoS and traffic engineering.

Chapter 3 develops a detailed and highly parameterized model for evaluating QoS routing under a variety of realistic network conditions. The model draws from features present in current proposals for QoS routing, and provides significant flexibility in conducting experiments that illustrate the impact of inaccurate information about network resource availability on the routing decisions.

Chapter 3 also describes the design and implementation of a custom network routing simulator that implements all aspects of the model and serves as the performance evaluation environment in later chapters.

Chapter 4 applies the model and simulation environment developed in Chapter 3 to a detailed evaluation of the performance and overheads of QoS routing. It illustrates, in particular, the significant bandwidth and processing resources necessary to achieve good routing performance. The evaluation considers a wide array of parameters that affect the performance-overhead trade-off to varying degrees. A major contribution of Chapter 4 is a set of routing policies and network configuration guidelines to aid in the design of QoS routing schemes.

The model developed in Chapter 3 considers traditional QoS routing, in which dynamic routes are computed for each flow request as it arrives in an on-demand fashion. Recent research has proposed route precomputation and path caching as alternatives to reduce the computational overhead of QoS routing. But this prior work either does not consider the effects of link-state staleness, or only evaluates its effect on path precomputation performance. Chapter 5 directly tackles the staleness problem by introducing algorithms and mechanisms for enabling QoS route precomputation in the presence of inaccurate link-state information. These mechanisms are evaluated and shown to significantly reduce processing overheads in comparison to on-demand QoS routing, and improve the routing accuracy relative to other precomputation schemes.

While Chapter 5 addresses the computational overheads of dynamic QoS routing, it does not solve the problem of reducing the overhead due to frequent distribution of network state information. Chapter 6 fills this gap by developing a new hybrid scheme for dynamic routing that reduces computational *and* link-state distribution overheads. Hybrid routing incorporates the observations made in recent research characterizing the properties of Internet traffic. In particular, it capitalizes on the observation that though most Internet transfers are short, consisting of only a handful of packets, the small number of long-lived transfers carry the majority of the packets or bytes. In contrast to conventional QoS routing, hybrid routing applies dynamic routing only to this long-lived subset of the traffic while forwarding the remainder on static preprovisioned shortest-paths used by existing protocols. In doing so, it avoids the problem of basing link-state updates on traffic that fluctuates rapidly, thus improving the predictive quality of link-state information. Hybrid routing is shown to improve overall application performance and network utilization by reacting to load fluctuations without introducing the instability of conventional dynamic routing.

Chapter 7 concludes the dissertation with a summary of the research contributions and suggests topics for further investigation. Appendix A is included to provide further details on the routing simulator in the form of a Unix manual page.

CHAPTER 2

ROUTING IN WIDE-AREA NETWORKS

2.1 Introduction

This chapter presents background material in several areas related to routing in wide-area networks. We first describe how static routing functions in the Internet by summarizing the operation of standard interdomain and intradomain protocols. We also describe the PNNI routing protocol used in ATM networks. Next, we survey early work on dynamic routing protocols, concentrating in particular on the protocols deployed in the ARPANET, a precursor to today's Internet. Experiences with these early protocols lend some insight into the general problems with load- or delay-sensitive routing in packet-switched networks. Since our interest in dynamic routing is motivated to a large extent by its ability to find paths satisfying application QoS requirements, we offer a summary of existing service models for QoS in the Internet and ATM networks. This is followed by an extensive survey of QoS routing issues, algorithms, and protocols.

2.2 Routing in the Internet

In this section we review the details of standard interdomain and intradomain routing protocols currently used in the Internet. In addition, Section 2.2.3 briefly describes ATM routing. Though less widely deployed, ATM was designed as a dynamic, QoS-capable routing protocol.

2.2.1 Interdomain Routing

The current standard exterior gateway protocol is the Border Gateway Protocol version 4 (BGP) [40, 74]. Its primary function is to exchange network reachability information to allow inter-AS communication. In addition to the reachability information itself, the protocol also includes an exchange of the list of ASs that the information traverses. In this sense, BGP is called a path vector protocol. Using the reachability information and the path vector, the AS connectivity may be determined and routing loops are detected and pruned. Version 4 of BGP introduced additional support for classless interdomain routing (CIDR) in which BGP routers send reachability information in the form of variable-length IP address prefixes and subnet mask lengths [31]. It also introduces mechanisms for route aggregation to reduce routing table size.

BGP peers communicate using TCP connections to ensure reliable transmission of routing information. A BGP session is established when a TCP connection is initiated and peers agree on session parameters. After session establishment, BGP routers initially exchange their entire BGP routing tables. Subsequently, the tables are modified incrementally using update messages that may carry reachability information, a list of withdrawn (invalidated) routes, or a set of path attributes. BGP avoids the use of periodic refreshes of the routing table. BGP routers do, however, use short periodic keepalive messages to verify that the session is still live. BGP also associates the routing table with a version number to track instances of the table as it changes.

Though BGP is an exterior gateway protocol, routers within an AS can construct BGP tunnels. These internal BGP connections may be used between routers that carry transit traffic through the AS without affecting the traffic carried by other nontransit routers in the domain.

2.2.2 Intradomain Routing

A common interior gateway protocol (IGP) is the Routing Information Protocol (RIP), a distance-vector protocol [43, 58]. The basic operation of distance-vector algorithms is for each router to broadcast to its immediate neighbors information about all destinations known to it along with their corresponding shortest distances. In RIP, shortest distance implies fewest hops. Each router updates its own routing tables based on this new information and then re-broadcasts the updates. Distances are calculated using a distributed version of the Bellman-Ford shortest path algorithm [22]. The process of routing table exchange may be thought of as passing rumors about the best way to reach destinations. The primary problem with distance-vector routing is that the routing information permeates through the network slowly and thus converges slowly, resulting in loops that can persist for a significant amount of time. Distance-vector algorithms are simple to implement, however, due to their local nature; each router need only send information to its neighbors. RIP is suitable mainly for small networks, and its drawbacks have led to the adoption of other, more sophisticated protocols that do not use distance-vector algorithms.

The other primary class of IP routing protocols is called link-state routing, exemplified by protocols such as Open Shortest-Path First (OSPF) [64, 65] and Intermediate System to Intermediate System (IS-IS) [46]. In these algorithms each router floods the entire autonomous system (AS) with information about its incident links. This information includes their state (according to some link metric, e.g., distance, transmission rate, or cost) and the identity of direct neighbors. Link-state routers take the approach of distributing local information globally as opposed to distance-vector routers which distribute global information locally. The most commonly used link-state IGP is the OSPF protocol. OSPF uses a link-state database at each router that serves as a topology map of the internetwork. The database consists of link state advertisements (LSAs) which describe local portions of the internetwork. These LSAs are distributed in the AS using a reliable flooding protocol in which advertisements are sent in high priority messages to assure that LSAs are processed quickly. Each router, R, uses the link-state database to calculate shortest paths trees rooted at R to other routers in the AS. The well-known Dijkstra single-source shortest-path tree algorithm [22] is used in this step. Here, the shortest-path is defined in terms of the link cost metric, and is not limited to hopcount only as in RIP. Link costs are not specified by the protocol but must be 16-bit positive integers. In OSPF routers may distribute LSAs at most every 5 seconds and the maximum time between refreshes of self-originated LSAs is 30 minutes. An advantage of link-state protocols is their ability to divide the domain into levels or areas. In OSPF, information about what is reachable outside an area is contained in special LSAs called AS-external-LSAs and summary-LSAs.

Both RIP and OSPF are packet-level, hop-by-hop routing protocols. Routes are stored in the form of forwarding tables that list only the next-hop router for a given destination. When a packet arrives, the router simply looks up the next hop for the packet's destination, and forwards the packet on the corresponding interface. Since path selection is done for each packet in a decentralized fashion, packets from the same transfer may follow different paths. This might occur if the underlying route changes, perhaps due recomputation after a link or router failure. An advantage of hop-by-hop routing is that the routers need not keep persistent state about paths or participate in a path set-up procedure before data transfer begins.

2.2.3 Routing in ATM Networks

In contrast to IP routing protocols which operate in a connectionless fashion, routing in asynchronous transfer mode (ATM) networks follows a connection-oriented paradigm. Also, while IP networks use variable-length packets, ATM operates on fixed-size packets called cells, each 53 bytes long. In the ATM model, connection paths are computed at the source switch and then signaled in a set-up phase which installs state in the network to facilitate high-speed hardware switching of cells.

Routing in ATM networks is done using the Private Network-Network Interface (PNNI) protocol [71]. PNNI uses a link state approach in which nodes flood information about reachability and QoS to all other nodes in the network. Another key difference between ATM and protocols such as OSPF, however, is the amount of hierarchy in the network. Whereas OSPF provides two levels (multiple areas within an AS) ATM can support up to 105.

At each ATM level, the network consists of logical nodes interconnected with logical links. Nodes at a given level are organized further into peer groups which exchange link-state information. A peer group is represented in its parent group as a single logical node by an elected peer group leader. The peer group leader summarizes link attribute and reachability information for its peer group and exchanges these summaries with all logical nodes in the parent group. In turn, it receives summaries from its peers at the higher level and transfers these down to the child group. It is important to note that the information fed down to lower levels is more and more aggregated as it travels downward. That is, nodes at the lowest hierarchy level have full information about their peers, summarized information about their parent group, further summarized information about their grandparent group, etc. [2]. To gather reachability information initially, nodes participate in a Hello protocol in which they learn who else is in their peer group and synchronize their topology databases. The Hello packets are exchanged periodically with immediate neighbors so this also serves as a link failure detection mechanism.

Once nodes have full topology information, they can use it to route connection requests. When a network ingress switch receives a connection request via the user-network interface (UNI) protocol, it uses a shortest path algorithm (e.g., Dijkstra) to find paths that connect the source to the destination. It then creates a signaling request with the full source route enumerated. The route is subject to the information available at the source, so it will be a fully specified within the peer group and less detailed in higher level peer groups. When the request arrives in a neighbor peer group (about which the originating node had only aggregated information), the border node replaces some of the source route with a more detailed route through its peer group. In this way the connection request makes its way to the destination. Once the connection is set up and identifiers are in place, operations at the switch are very simple. A cell enters a switch on a known connection identifier, the outgoing port and outgoing identifier are extracted from a lookup table, and the cell is retransmitted on the outgoing port with the new outgoing identifier.

2.3 Origins of Dynamic Routing in Wide-Area Networks

With the exception of ATM, the routing protocols described in Section 2.2 are often considered quasi-static, or simply static [88]. While they do respond to link failures or topology changes, paths are not adapted continuously to changing load and delay conditions in the network. In contrast, dynamic, or adaptive, routing algorithms attempt to find routes around network congestion. In general, network routers that participate in dynamic routing protocols must perform four basic functions [75]:

- report local state (e.g., regarding delay, cost, available bandwidth, etc.) to neighbors or to a central entity that acts as a routing authority
- assemble a global view of network state using the local reports
- compute routes based on the global view
- update routing tables to reflect the new routes (e.g., by recording the next-hop forwarding information for routing packets)

A key property in any dynamic routing scheme that follows this basic framework is that the global characterization of network state is necessarily inaccurate (i.e., based on past information). As reported in [60], investigations of techniques for collecting and distributing the information have revealed some basic observations:

- when routes are allowed to change more frequently (e.g., on a packet-by-packet basis), the information used in computing routes must be more accurate
- when routes are selected based on information from remote nodes (i.e., not only on local information), accuracy of network state collected from remote nodes is affected by propagation delays in the network
- if routes are assigned for a long time (e.g., all traffic between a pair of nodes), route computation may be done at the source or a central authority, where information is less accurate

Early work on dynamic routing for wide-area networks considered a variety of algorithms that differ in implementation complexity and degree of centralization. In the next section we review some early dynamic routing proposals but focus primarily on the evolution of dynamic routing in the ARPANET, the packet-switching research network that grew into today's Internet. Surveys of a number of early routing techniques appear in [60, 75].

2.3.1 Early Dynamic Routing Algorithms

An example of one early form of adaptive packet routing is "hot-potato" routing in which alternate paths are selected when the shortest path is busy or has failed. Alternates may be ranked in the order of distance to the destination to choose the free neighbor that is closest to the destination. Advantages of hot-potato routing are its rapid adaptation to network load or topology changes, and its ability to make decisions based only on local information at each node. When network utilization is high, however, it tends to pick longer paths, which consume additional resources. In "shortest-path plus bias" routing, nodes consider the local packet queue length in addition to distance. When the queue length on the shortest path is long, alternate paths with shorter queues are considered.

Delta routing proposes a hybrid approach that combines centralized and distributed techniques [75]. A central authority receives average delay measurement data from each node, computes a routing strategy for each node, and returns this information back to the nodes. Each individual node, however, makes the ultimate choice of how paths are chosen within the strategy dictated by the central authority. In determining the strategy, the central authority considers a set of alternate paths that have disjoint first hop links. If one of the alternates has significantly shorter delay, the remote node is directed to use only this path. When multiple paths are approximately equal, the remote node may choose the path based on its current local observations of queue length. The "delta" in delta routing refers to the threshold used to determine the amount of decision-making delegated to remote nodes by the central authority.

2.3.2 Dynamic Routing in the ARPANET

The ARPANET routing algorithm, first deployed in 1969, used a completely distributed approach in which nodes exchanged tables of shortest-delay routes [61]. Each router (called Interface Message Processors) participated in the route computation by using a distributed version of the Bellman-Ford shortest-path algorithm [22]. Each node maintained a table of estimated delay to all other nodes in the network, and exchanged this table every 2/3 seconds with its neighbors. Using tables received from its neighbors, along with its own measurements of the delay to neighbors, each node was able to determine the minimum delay path to each destination. Packets could then be forwarded on the current least-delay path to the destination. The delay measurement was simply an instantaneous count of the packets queued for transmission to a given neighbor, plus a bias to avoid reporting zero queue length.

The original ARPANET routing algorithm was in operation for several years, during which several important fundamental drawbacks were observed [49, 62]:

- queue length alone served as a poor indicator of delay, since the queues were constrained by software to have a short maximum length
- instantaneous samples of queue length did not predict average delay due to the high rate of fluctuation in the queue length
- though the distributed Bellman-Ford algorithm will converge given a set of static distance metrics, it was difficult to ensure consistent (i.e., loop-free) routes in the face of the highly volatile metric being used
- the distributed nature of the route computation and delay estimate distribution resulted in slow adaptation to network congestion, as well as overreaction to minor changes, possibly leading to route oscillation
- exchange of complete routing tables resulted in transmission of long packets, that would grow in size as the network grew

Many of these problems were addressed by the new ARPANET routing algorithm, shortest-path first (SPF), deployed in 1979 [62]. While the original algorithm fell into the distance-vector category, SPF was a link-state protocol. That is, each node maintained a database describing the network topology and the delay on each link. Using the link-state database, each node independently computed shortest-delay paths to all destinations, and forwarded packets accordingly. Each node measured the actual delay in sending packets to each neighbor and averaged this delay over a 10 second interval. If the delay values between successive measurements differed by a significant amount (64 ms), the node sent link-state updates to report the new value to all other nodes. To ensure that delay values were periodically refreshed, even if no local delay changes took place, the maximum time between updates was 50 seconds.

The SPF protocol addressed many shortcomings of the original ARPANET protocol. In particular, it produced loop-free and more stable routes, avoided instantaneous measurements of delay, responded to congestion, and limited the amount of information contained in update messages. Despite these advances, however, SPF suffered poor performance under heavy load conditions [49]. Specifically, the effectiveness of using packet delay measurements depends on the correspondence of the measured delay to the delay encountered after rerouting based on these measurements. When network load is high, the ability of measured delays to predict network conditions decreases significantly. Three factors were found to be the primary reasons for this ineffectiveness:

1. the range of delay values was too high, resulting in the possibility of a highly utilized link appearing 20 times more costly than a lightly loaded link

2. the variation in the values reported in successive updates for a single link was unlimited
3. all nodes in the network react to link metric updates simultaneously, due simply to the fact that update processing was a high-priority operation in each router

The most damaging effect arising from these factors was the significant oscillation between candidate routes as the desirability of links alternated. In addition, when the reported delay values experienced large swings (due to the large range of values and lack of restrictions on their variation), the update threshold was frequently satisfied and additional updates were generated. These updates in turn led to additional invocations of the route computation operation at each router. Thus bandwidth and CPU resource consumption were also increased.

To address the instability of SPF under high load conditions, a new link metric was proposed which transformed the original delay metric via a series of simple filters [49]. When load was light, the new metric behaved much like SPF, but when the network was heavily utilized it resulted in some traffic taking slightly longer-delay paths, while allowing the remainder to remain on the shortest-delay route. In addition, the new metric imposed hard limits on the range of possible values and also limited how much the reported cost could change between successive updates.

Despite its ability to dampen oscillations in SPF, the new metric can only achieve a limited degree of load balancing since it is subject to the single-path restraint [49, 88]. That is, the routing algorithm considers only a single "best" path no matter how many paths are available from source to destination. Recently proposed multi-path extensions to OSPF and IS-IS, recognize this limitation and support more flexible tie-breaking based on link load [84]. These proposals do not, however, address the other limitations of static shortest paths. In [88] it is shown that, in general, shortest-path dynamic routing algorithms often exhibit oscillatory behavior and lead to poor network utilization. Without additional mechanisms for multi-path routing, global coordination of routing updates, or appropriate metric transformations, achieving stable routing solutions is very difficult.

2.4 Service Models for Quality-of-Service

In this section we briefly review several basic service models currently proposed for delivering quality of service in wide-area networks. These models drive much of the research effort in the design and development of new algorithms and protocols to support various classes of service as defined by the models. In particular, Section 2.5 describes several QoS routing algorithms designed to operate within the frameworks described below.

2.4.1 Integrated Services

The Integrated Services working group of the Internet Engineering Task Force (IETF) has defined the behavior of network elements necessary to provide two new classes of service: controlled-load and guaranteed service.

In controlled-load service [92], an application is provided a service that closely approximates the behavior which would be perceived if the network were unloaded. Note that unloaded does not imply that there is no other network traffic at all, but rather that the network is not experiencing heavy load or congestion. Applications receiving controlled load service may assume that a high percentage of their packets will arrive successfully at the destinations, and that the transit delay experienced by the majority of the packets will not greatly exceed the minimum transit delay. Here, minimum transit delay includes propagation delay plus fixed processing time in routers or other network devices in the path.

Controlled-load service is intended to serve those real-time applications which can adapt to slight variance in network delay but degrade under overload conditions. To receive the service, an application must provide a traffic specification ($TSpec$) in the form of a token bucket specification in addition to a peak rate, minimum policed unit, and maximum packet size. The token bucket has depth b (bytes) and rate r (bytes of IP datagram per second). The minimum policed unit is the smallest datagram size that will be considered by the policing mechanism and the maximum packet size represents the largest packet that will conform to the $TSpec$. Network elements implementing the service are expected to use the $TSpec$ to perform admission control so that they can ensure the availability of adequate bandwidth and other resources. Nonconformant application traffic may expect QoS to be characteristic of overload (e.g., long delay and many losses) [92].

Guaranteed service [78] is intended for applications which cannot tolerate any unbounded delays. These applications require a firm guarantee that packets will arrive at destinations no later than a specified time after transmission by the source. Some audio and video playback applications fall into this category, along with hard real-time applications. The service does not minimize delay jitter, rather it controls only the maximal queuing delay experienced in the network. Applications must still be able to buffer packets that arrive much earlier than the delivery deadline. Network elements supporting guaranteed service provide an assured bandwidth that produces a delay-bounded service with no queuing loss for all conformant traffic. Achieving this level of QoS requires that every element in the path support guaranteed service, however.

As with controlled-load service, guaranteed service expects a $TSpec$ in the form of a token bucket specification, a peak rate, minimum policed unit, and maximum datagram size. Applications also specify a service specification ($RSpec$) which consists of a rate, R (greater than token bucket rate), and a slack term. The slack term represents the difference between the desired delay and the delay achievable using a reservation level R. Authors of the specification reference several suitable packet scheduling algorithms that deliver such a service, including Weighted Fair Queuing, Virtual Clock, and Jitter-EDD.

2.4.2 Differentiated Services

Integrated services require the use of complex scheduling and signaling mechanisms within all network forwarding elements and forces applications to express requirements using a specific traffic specification format. Given the difficulty of deploying such mechanisms on a wide scale, coupled with the need for changes to end-user applications, the Differentiated Services (DS) architecture was proposed as a more scalable, and readily deployable, alternative [12]. Some key examples of departures from the Integrated Services model include:

- operation with existing applications without modifications to various application programming interfaces or host software
- decoupling of traffic conditioning (policing, shaping, etc. according to the service level agreement) from packet forwarding behavior inside the network
- no dependence on application-level signaling
- packet forwarding behaviors for which the implementation cost does not dominate the cost of the network element
- no per-flow (i.e., per application) state kept in the network
- design for interoperability with non-DS-compliant network devices, and accommodation for incremental deployment

The architecture relies on a model in which traffic entering a DS network is classified and conditioned at the network boundary and assigned to a traffic aggregate that determines its network treatment. Within the network, packets are forwarded according to the per-hop behavior associated with the aggregate class. Traffic conditioning consists of various operations performed on traffic to ensure that it conforms to the rules of a specified traffic conditioning agreement. The per-hop behavior in the network is typically implemented by a combination of buffer management and packet scheduling techniques. To date, two per-hop behaviors have been proposed in the IETF, assured forwarding (AF) and expedited forwarding (EF). In AF, packets are marked with drop precedence values according to the class to which they belong. In the event of congestion, DS nodes try to protect packets with lower drop precedence marking from being lost by dropping packets with higher drop precedence values with higher probability. There are no delay requirements associated with AF. Expedited forwarding is designed to look to the endpoints like a dedicated “virtual leased line” at some requested rate, and is often referred to as “premium” service. In EF, the forwarding treatment for a particular traffic aggregate is such that the departure rate from a DS node meets or exceeds a configurable rate. The EF service requires queue management mechanisms that can ensure that packets from a traffic aggregate see virtually empty (or very short) queues in transit.

2.4.3 ATM Service Classes

ATM provides a range of well-defined traffic classes for use by different types of applications. Each service may be thought of as a successive relaxation of the class with the tightest guarantees. An application expresses a traffic contract with a set of parameters describing the traffic (analogous to the IETF $TSpec$) and parameters describing the required QoS. As part of the contract, the application in general specifies its desire to use one of the following traffic classes [79]:

- **constant bit rate (CBR):** This class acts like circuit switching with a constant bit rate. It allows specification of the desired cell loss ratio and provides a service loosely similar to the IETF guaranteed service class in that it is a firm guarantee.
- **variable bit rate (VBR):** This permits transmission at a variable bit rate but with some nonzero random loss. It is divided into an real-time and non real-time subclasses where cell delay variation is specified only for the former. Compressed video is the most frequently cited example of an application that would benefit from VBR.
- **available bit rate (ABR):** ABR does not provide any firm guarantee on cell transfer delay or cell loss ratio but the network tries to minimize delay and loss insofar as possible. The source is expected to control its rate when the network is congested but it may declare a minimum cell rate that is set aside for the ABR virtual circuit. Most VCs request a zero minimum cell rate.
- **unspecified bit rate (UBR):** This class is for applications that are not delay- or loss-sensitive. They are not subject to connection admission control, nor are they policed to assure conformance. When the network is congested cell loss occurs but the sources need not reduce their transmission rate. It is expected that some higher protocol layer will provide loss recovery and retransmission.

When applications desire a best-effort service, they usually will specify ABR or UBR. Hard and soft real-time applications generally will use CBR and VBR depending on their traffic characteristics.

2.5 Quality-of-Service Routing

To accommodate diverse traffic characteristics and quality-of-service (QoS) requirements, emerging networks can employ a variety of mechanisms to control access to shared link, buffer, and processing resources [7, 10, 12, 21, 29, 83, 95]. These mechanisms include traffic shaping and flow control to regulate an individual or groups of traffic flows, as well as link scheduling and buffer management to coordinate resource sharing at the packet or cell level. Complementing these lower-level mechanisms, routing and signaling protocols control network dynamics by directing traffic at the flow or connection level. In particular, QoS routing selects a path for each flow or connection to satisfy diverse performance requirements and optimize resource usage [23, 54, 89]. Most current routing schemes are geared toward connectivity and reachability, and consider only a single metric such as hopcount or cost (e.g., protocols described in Section 2.2.2. QoS routing, on the other hand, must characterize networks with multiple metrics, for example bandwidth, delay, and loss probability, to be useful for QoS-sensitive applications.

In this section, we discuss some of the primary issues in designing effective QoS routing algorithms and protocols, and follow with some examples for packet-switched networks. In addition, we give an overview of QoS routing in ATM networks. The section concludes with a discussion of MPLS, a protocol that combines characteristics of ATM routing and packet-based QoS routing to provide mechanisms for traffic engineering.

2.5.1 Issues in QoS Routing

Interdomain and intradomain QoS routing: The proposed framework for realizing QoS routing in the Internet advocates a clear distinction between intra and interdomain routing [23]. The motivation behind this hierarchical approach is to foster innovative intradomain architectures that are tailored for the specific needs of an AS, while providing simple and stable mechanisms for domains to exchange information about the QoS available on paths. This approach is quite similar to the existing IGP-EGP architecture which performs relatively sophisticated computations within a routing domain and exchanges only aggregated reachability information between domains.

Interdomain QoS routing should be scalable, implying that it cannot be based on highly dynamic information about resource availability, since good performance would require frequent information exchange. At a basic level, interdomain information distribution should enable reachability determination, loop-free paths, address aggregation, and determination of the QoS available on paths.

The QoS information should be relatively static, and based on provisioned capacity for aggregate transit flows rather than ephemeral load fluctuations [23]. It is expected that the available QoS through a transit domain will not fluctuate significantly, since provisioning should account for the expected traffic from neighboring domains and its associated QoS requirements (perhaps based on traffic contracts).

Intradomain routing requirements are left intentionally vague in the framework to allow as much flexibility as possible. The remainder of this section, including the examples in Section 2.5.2 relate to intradomain routing.

QoS routing metrics: Given the need to perform routing based on QoS metrics, one fundamental question to be resolved is which metric to choose. It is tempting to try a single mixed metric; many well-known algorithms exist and are widely deployed for the single metric path optimization problem. As discussed in [89], it is quite difficult to compose such a metric. One such possibility is $M(l) = \frac{B(l)}{D(l) \cdot P(l)}$, where B, D, and P represent bandwidth, delay, and loss probability for link l, respectively. Thus a desirable path would have a large M value. An important consideration, though, is how such a metric could be aggregated along a path to determine if it could meet an end-to-end QoS requirement. Clearly, adding or taking the minimum along the path are invalid. At best the mixed metric may be an indicator of a link or path's QoS capability [89].

Next consider the case of multiple metrics. Some suitable QoS routing metrics might be delay, delay jitter, bandwidth, hop count, loss probability, or cost. With multiple metrics, however, computational complexity becomes an issue. In fact finding an optimal path subject to constraints on two metrics is in general an NP-complete problem [32]. One determining factor in the complexity of the multiple metric problem is the set of composition rules of the metrics [89]. Three common composition rules applicable to most metrics of interest are additive (e.g., delay, cost), multiplicative (e.g., loss probability), and minimum (e.g., available bandwidth).

Tackling this problem requires heuristics or some other methods. We review several QoS routing algorithms in the next section that, in general, take the approach of giving one QoS metric precedence over another to simplify the problem. Other recently developed heuristics address the problems of finding paths subject to a combination of end-to-end delay and cost constraints [17], or delay and buffer requirements [98].

Link-state distribution: Related to which metrics to use is the issue of how to distribute the information. Unlike currently used link metrics like hopcount, QoS attributes are highly dynamic. If link-state updates were flooded every time available bandwidth changed, for example, the network would be overwhelmed with update traffic. The overhead may be worsened if reliable flooding is used as with OSPF. An alternative lower-cost distribution technique is to compute a spanning tree for the network and forward updates only along the tree. Updates may also be distributed only to nearby neighbors, within some given hopcount. This technique is motivated by the observation that communication in a large network exhibits spatial locality [41].

Another way to reduce update overhead is to quantize metric information to avoid generation unless the new value crosses a range boundary. Such coarse information may impact routing performance, however [23].

Source vs. hop-by-hop routing: Another fundamental issue is whether to use source-directed or distributed hop-by-hop routing. Hop-by-hop is the prevailing method in packet-switched IP networks while source routing is widely used for policy routing and ATM networks. Source-directed routes are flexible since a source may tailor the route to each individual request and can use any algorithm it wishes. Additionally, loop-free routing can be guaranteed. Using global network state in the route computation also aids in identifying alternate paths [13]. Its performance depends, however, on the accuracy of link-state information which can be poor in large networks. In the absence of path set-up, each packet must also carry the complete route, increasing the size of packet headers. Using a signaling phase can alleviate this problem, but at the expense of additional set-up delay.

Hop-by-hop routing, on the other hand, may be fully distributed and requires only a table lookup at each intermediate router. Packet headers can be kept short since they need carry the entire forwarding path. However, the distributed nature of the routing decision can lead to routing loops when the state available at different nodes is inconsistent. Also, retrying a set-up attempt on an alternate path requires the intermediate nodes to maintain additional state to avoid using the same path as the original request. Finally, though hop-by-hop routing can capitalize on fully distributed algorithms, developing efficient distributed algorithms for computing QoS routes is difficult without full detailed topology or link-state information [18].

2.5.2 Algorithms for QoS Routing

Below we present the details of several unicast QoS routing algorithms that represent a variety of approaches, including hop-by-hop and source-directed, as well as link-state and distance-vector. A recent survey of other examples, including algorithms for multicast may be found in [18].

Bandwidth- and delay-constrained QoS routing: Wang and Crowcroft propose several algorithms for QoS routing, one centralized and two distributed [89]. The first is a centralized generic source routing algorithm in which the metrics of interest are bandwidth and delay. The network is modeled as a directed graph with each arc having an associated bandwidth and delay parameter. They define the width of a path as the minimum of all the link bandwidths along the path, and the length of the path as the sum of all the link delays along the path. So given QoS constraints on minimum bandwidth, B, and maximum delay, D, the goal of the algorithm is to find a path with width B and length D. The algorithm proceeds by removing all links from the network with bandwidth less than B. Then a shortest path algorithm is run to find the shortest path from source to destination based on delay alone; consider this path to have delay d. If $d < D$, then the algorithm is successful, otherwise no path satisfying the QoS requirement is available.

Next, the authors propose distance-vector and link state versions of a hop-by-hop routing algorithm. Hop-by-hop routing requires precomputation of forwarding entries for every destination but QoS hop-by-hop routing must have forwarding entries for each destination for every desired QoS level. Clearly this is impossible so the authors propose to precompute only the "best" route to each destination. The best route in terms of bandwidth and delay may not exist, however, so they define a precedence of bandwidth over delay, claiming that bandwidth deficiency is more likely to degrade QoS-sensitive applications. The search strategy, then, is to find a path with maximal bandwidth and when multiple such paths are available, to choose the one with shortest delay. These "shortest-widest" paths are shown to be loop-free, making them attractive for use in distributed computations. The distance-vector version of the algorithm is similar to a Bellman-Ford shortest paths algorithm in that at each iteration, h, all widest paths are found to each node within h hops. If multiple such paths are found, the minimum delay one is chosen and the width and length of the shortest-widest path is updated for each node. The hop count is incremented and the operations repeat until the link count is exceeded by h. The authors propose a similar algorithm based on link states that follows a Dijkstra path optimization procedure. It is claimed that these two algorithms scale well since, per iteration, they perform a number of operations proportional to that in conventional shortest path algorithms.

QoS extensions to OSPF: Guerin *et al.* recently proposed mechanisms to extend the OSPF intradomain routing protocol with the goal of improving performance of QoS flows [3, 39] (see [65] for details on the OSPF protocol). A primary goal here is simplicity, thus trading off optimality for ease of deployment is considered desirable. The algorithms assume an environment in which all routers support QoS. Additionally, only unicast is considered. Metrics of interest are bandwidth and hop count. That is, it is assumed that most QoS requirements are derivable from a rate-based quantity, i.e. bandwidth. Hop count is taken to represent the path cost to the network, implying that fewer hops require fewer network resources. The authors also introduce a policy mechanism which is used to identify network links that should be included or ignored when performing path computation. An example would be to remove (prune) all high-latency links when computing a path for a delay-sensitive flow. Since these algorithms are meant to work in an OSPF environment, they assume that a current link state database is present listing available bandwidth of each link. QoS routing will be much more effective if these link states are kept as accurate as possible but the frequent updates necessary for such accuracy is not scalable or practical. The authors propose a combined scheme in which periodic updates are distributed every T seconds for any change greater than some Δ, and event-driven updates for bandwidth changes greater than another δ. Moreover, they suggest a minimum interval between any update to prevent overload.

Guerin *et al.* present three algorithms each with the same objective, namely to find a route with minimum hops that supports the requested bandwidth [39]. The network model is the same as that used by OSPF, i.e., a directed graph with routers and networks as nodes. The topology is assumed to be pruned according to whatever policy may be in effect and whenever multiple feasible minimum-hop paths are available, the algorithms pick the one with maximal bandwidth. The motivation for this policy is to increase the probability that the requested bandwidth will indeed be available when resources are sought.

The first algorithm precomputes a minimum hop path with maximal bandwidth for all destinations. It uses a Bellman-Ford approach, taking advantage of the property that at each iteration, h, it knows the optimal path between source and all destinations among paths having h or fewer hops. So at the hth iteration, the maximum bandwidth available to each destination on a path of no more than h hops is stored along with forwarding information. Upon termination the stored information allows identification, for any destination and bandwidth requirement, a minimum hop path with sufficient bandwidth. Note that although a Bellman-Ford algorithm is used, this is not a distance-vector algorithm. Experiences and evaluation of the implementation of this algorithm on actual router hardware appears in [4].

Another alternative algorithm computes QoS routes on demand, allowing the use of the most up-to-date link state information and the absence of a QoS routing table. The algorithm performs a basic Dijkstra minimum hop path calculation on a graph in which all links with insufficient bandwidth have been removed. As it proceeds, it keeps track of only the path with maximal bandwidth among equal hop count paths. Note that this is very similar to Wang and Crowcroft's centralized algorithm described above.

The final algorithm precomputes “approximate” routes for all destinations and bandwidth values. The approximation arises from the fact that computed paths may not have the minimum number of hops. The algorithm computes minimum hop paths using a standard Dijkstra calculation but for a set of quantized bandwidth values. That is, the range of bandwidth request possibilities is mapped to a fixed number of classes. When a path is calculated for a particular bandwidth value, b, links having bandwidth less than b are removed and then the minimum hop count algorithm is run.

Distributed probe-based QoS routing: Chen and Nahrstedt propose variations of a completely distributed algorithm that is able to capitalize on the accurate local state available at individual nodes to find a path satisfying the QoS requirements. In the selective probing algorithm [16], QoS routes are found by sending probe packets along multiple paths from source to destination. Intermediate nodes forward probes only along paths that meet the requested QoS requirement, according to a local admission test. A probe reaching the destination serves as an indication that a path with sufficient resources exists. Upon receiving a probe, the destination returns an acknowledgment along the reverse path that reserves resources along the route. Several forwarding conditions (admission tests) for probes are developed for a number of QoS metrics, as well as calculations for suitable probe waiting times that are necessary for additive QoS metrics such as delay or cost. Another similar distributed algorithm, presented in [50], uses a bounded flooding approach to find least-loaded, shortest-paths.

Ticket-based probing [15] is another distributed algorithm that attempts to find QoS routes subject to delay constraints but also with low (but not necessarily strictly bounded) cost. Each node keeps estimates of the end-to-end delay and cost to every other node, and an estimate of the variation in the end-to-end delay. This information is assumed to be updated periodically using conventional link-state or distance-vector protocols. As with selective probing, the source sends out probes toward the destination, but in this case the probes carry two types of tickets. Probes carrying one type of ticket follow least-delay paths while probes carrying the other type attempt to find least-cost paths that may have longer delay. The authors propose rules for distributing tickets based on the likelihood of finding a feasible path. Again, probes proceed only when the path appears to meet the delay requirement, otherwise they are invalidated (but still forwarded). If a valid probe reaches the destination, a feasible path exists. If multiple valid probes arrive, the least cost path is selected.

Note that the work described above focuses primarily on algorithmic issues, such as computational or message complexity, or the optimization of various combinations of QoS metrics. Protocol issues such as heterogeneity and dealing with failures are not addressed in these initial proposals. The extensions to OSPF, however, are designed with rapid deployment in mind, as the authors describe in detail possibilities for integration with current OSPF routing and RSVP-based resource reservation [96] on the global Internet.

2.5.3 QoS Routing in ATM Networks

Unlike the global Internet, ATM networks were designed with guaranteed QoS in mind. As such, ATM incorporates a number of features allowing a node to request a certain QoS and receive assurance from the network that it will be delivered. The signaling, QoS routing, and admission control are integrated into a single approach.

As mentioned in Section 2.5.1 above, one important feature of any QoS routing approach is the metrics it uses to find paths that can meet the requested application QoS. ATM uses two types of link parameters [2]: non-additive link attributes which determine whether a link can meet some QoS requirement, and additive link metrics with may be aggregated to determined if a chosen path can meet an end-to-end QoS requirement.

Link attributes include available cell rate (per traffic class), cell rate margin (difference between effective and allocated bandwidth per class), and a relative variance factor. Link metrics include maximum cell transfer delay, maximum cell delay variation, and maximum cell loss ratio (CBR and VBR only). Delay and jitter are specified per traffic class and it is assumed that each node can provide sufficient resource isolation between traffic classes.

ATM uses source routing for finding QoS routes but faces the obstacle of the high degree of metric aggregation due to the extensive routing hierarchy. Therefore, when examining the topology database, the source can only make a guess about the actual resources available. Just as important, connection admission control (CAC) at each node is a customized algorithm that may be different for various vendor implementations. To address this problem, ATM uses a Generic CAC (GCAC) algorithm which is meant to model the expected behavior of the CAC at a remote node given its advertised link metrics.

Using the GCAC, nodes receiving a request for a new connection perform the following procedure [2]:

1. prune all links from the topology that cannot provide the requested cell rate or cell loss ratio using the GCAC

2. perform a shortest path computation to find a set of paths to the destination

3. prune the set of paths based on their ability to provide requested end-to-end delay and perhaps other constraints
4. after choosing a path, create a signaling request with the transit list (source route) inserted

Of course it is still possible that the chosen route will fail due to CAC failure at some intermediate node; this may be caused by aggregation-induced inaccuracy or staleness in the QoS information. Rather than return failure all the way back to the source, however, ATM supports crankback to restart the request at an intermediate node that is the ingress to a peer group. The intermediate node then attempts to find a path using hopefully newer or more accurate information. Another mechanism used is fallback in the case step 2 above fails to produce any feasible routes. Fallback relaxes some of the request attributes until a path is found that can support some minimal set of requirements.

2.5.4 MPLS and Constraint-Based Routing

Multiprotocol Label Switching (MPLS) represents an effort by the Internet Engineering Task Force (IETF) to standardize a set of concepts into a unified set of protocols [29, 86]. MPLS is expected to improve network layer (e.g., IP) scalability and provide a framework for delivering new routing services (including QoS routing). The idea of label switching or label swapping was motivated primarily by the limited scalability of the current routing and forwarding paradigm in the Internet. Routers examine packet headers to determine the destination address and dispatch the packet to the corresponding outgoing interface for forwarding to the next hop. A longest-match prefix rule is used to resolve the next-hop address, requiring that routers find the best, or most specific, match between the destination address in the packet and the variable-length address prefixes stored in the routing table. Given the very large number of variable-length prefixes in the current Internet (approximately 64,000 [45]), and signs that it will continue to grow, the time and computational resources necessary to perform route lookups alone are significant.

Label switching relieves the the burden of longest-match operations by allowing routing and forwarding decisions based on a fixed-length (20 bit) label inserted between network layer and data-link layer packet headers. Label switched routers (LSRs) are able to forward packets quickly and simply by exact matching on these short labels. When a packet enters an MPLS-capable network, the ingress router performs a conventional route lookup, but also associates the packet with an assigned label. The packet is then forwarded to the next hop with the label. Subsequently, other LSRs in the path use the label as an index into a table which specifies an outgoing label and interface. The incoming label is replaced with the outgoing label, and the packet is forwarded, much like in

ATM. Similarly, at the egress of the MPLS-capable network, the last LSR must bind the label to a network-layer address prefix. This scheme eliminates the need for route lookups except at the edge routers. The path defined by the series of labels is called a label switched path (LSP). Label assignment is done with a separate Label Distribution Protocol (LDP) that allows neighboring LSRs to exchange label information. Key to MPLS is that labels may be associated with traffic at varying levels of granularity. For example, traffic flows between the same source and destination IP address may have their own label, or all traffic destined for a particular network exit point may be associated with a single label.

Although label switching was originally developed to improve the performance of route lookup and forwarding, today the performance benefits are less compelling. New algorithms for prefix matching, along with high-performance hardware, have enabled recently developed routers to perform traditional packet forwarding at gigabit speeds comparable to that of switches [26, 87]. Even so, MPLS delivers additional capabilities for providing QoS and engineering network traffic. By decoupling the forwarding mechanisms from routing, MPLS enables the deployment of specialized routing services or polices without changes to the packet forwarding path [86]. In particular, MPLS contains support for setting up explicitly routed LSPs in which the entire path is determined at the source. This allows administrators, for example, to reroute certain classes of traffic to balance network load independent of the underlying conventional shortest-path routing protocol [9]. MPLS also specifies a routing framework called constraint-based routing that enables demand-driven routing capable of satisfying resource and policy requirements. The network traffic engineer identifies classes of traffic and their endpoints, along with associated QoS metrics and policies. A constraint-based routing protocol is then expected to find suitable paths that meet the requirements. Constraint-based routing operates much the same as source-directed link-state QoS routing protocols. The value of constraint-based routing is in its ability to assist in traffic engineering while reducing the amount of manual route configuration.

CHAPTER 3

FLEXIBLE MODELS FOR QUALITY-OF-SERVICE ROUTING

3.1 Introduction

In this chapter, we develop a model of QoS routing that enables investigation of the fundamental trade-off between resource requirements and the quality of the routing decisions. We focus on link-state routing algorithms where the source router selects a path based on the flow traffic parameters and the available resources in the network. During periods of transient overload, link failure, or general congestion, these schemes are able to find QoS paths for more flows. However, QoS routing protocols can impose a significant bandwidth and processing load on the network, since each router must maintain its own view of the available link resources, distribute link-state information to other routers, and compute and establish routes for new flow requests.

Despite the apparent complexity of QoS routing, these path-selection and admission control frameworks offer network designers a considerable amount of latitude in limiting overheads. In particular, the network can control the complexity of the routing algorithm itself, as well as the frequency of route computation and link-state update messages. Link-state information can be propagated in a periodic fashion or in response to a significant change in the link-state metric (e.g., utilization). For example, a link may advertise its available bandwidth metric whenever it changes by more than 10% since the previous update message; triggering an update based on a change in available capacity ensures that the network has progressively more accurate information as the link becomes congested. In addition, a minimum time between update messages would typically be imposed to avoid overloading the network bandwidth and processing resources during rapid fluctuations in link bandwidth. However, large periods and coarse triggers result in stale link-state information, which can cause a router to select a suboptimal route or a route that cannot accommodate the new connection. Hence, tuning the frequency of link-state update messages requires a careful understanding of the trade-off between network overheads and the accuracy of routing decisions.

By capturing each of these policy and configuration variations, the model developed in this chapter is necessarily complex, and precludes a closed-form analytic expression for quantities of interest such as the flow blocking probability. Therefore, we have developed `routesim`, a simulation environment that allows us to efficiently evaluate a diverse collection of network and routing configurations. Section 3.5 of this chapter describes the high-level design of `routesim`.

3.2 Routing and Signaling Model

The following four sections describe the primary components of the QoS routing model, namely route computation, signaling, link costs, and link-state updates. These components, along with a rich network and traffic model, allow us to study a wide variety of policies and configurations.

3.2.1 Route Computation

Since predictable communication performance relies on having some sort of throughput guarantee, our routing model views bandwidth as the primary traffic metric for defining both application QoS and network resources. Although application requirements and network load may be characterized by several other dynamic parameters, including delay and loss, initial deployments of QoS routing are likely to focus simply on bandwidth to reduce algorithmic complexity. Hence, our model expresses a flow's performance requirements with a single parameter b that represents either a peak, average, or effective bandwidth, depending on the admission control policy. In practice, the end-host application may explicitly signal its required bandwidth, or network routers can detect a flow of related packets and originate a signaling request. Each link i has reserved (or utilized) bandwidth u_i that cannot be allocated to new flows. Consequently, a router's link-state database stores (possibly stale) information u_i' about the utilization of each link i in order to compute suitable routes for new flows. Each link also has a cost c_i (c_i') that is a function of the utilization u_i (u_i'), as discussed in Section 3.2.2.

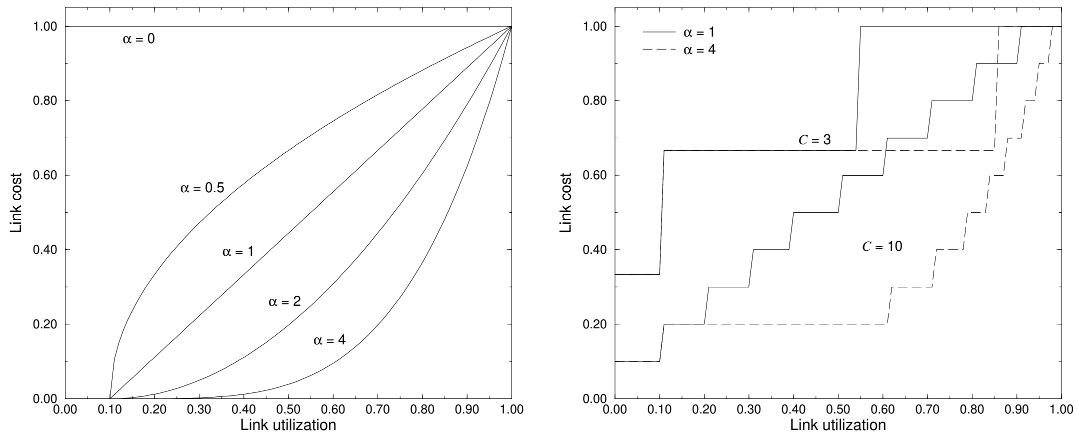
Although networks can employ a wide variety of QoS routing strategies, previous comparative studies have demonstrated that algorithms with a strong preference for minimum-hop routes almost always outperform algorithms that do not consider path length [1, 33, 34, 57, 59, 72]. For example, selecting the widest shortest path (i.e., the minimum-hop route with the maximum value of $\min_i\{1-$

$u_i\}$) increases the likelihood of successfully routing the new flow. Similarly, the network could select the minimum-hop path with the smallest total load (minimum value of $\sum_i u_i$) to balance network utilization. In contrast, non-minimal routing algorithms, such as shortest widest path, often select circuitous routes that consume additional network resources at the expense of future flows, which may be unable to locate a feasible route. Biasing toward shortest-path routes is particularly attractive in a large, distributed network, since path length is a relatively stable metric, compared with dynamic measurements of link delay or loss rate [34].

In our model, the source selects a route based on the bandwidth requirement b and the destination node in three steps:

1. (Optionally) Prune infeasible links (i.e., links i with $u'_i + b > 1$)
2. Compute shortest paths to the destination based on hopcount
3. Extract a route with the minimum total cost $\sum_i c'_i$.

This process effectively computes a "cheapest-shortest-feasible," or a "cheapest-shortest" path, depending on whether or not the pruning step is enabled. By pruning any infeasible links (subject to stale information), the source performs a preliminary form of admission control to avoid selecting a route that cannot support the new flow. In an N-node network with L links, pruning has $O(L)$ computational complexity and produces a sparser graph consisting entirely of feasible links. Then, the router can employ the Dijkstra shortest-path tree algorithm [22] to compute a the shortest path with the smallest total cost. This is achieved through a careful assignment of link weights, w_i, that permits a single invocation of the Dijkstra algorithm to produce a shortest-path route with the minimum cost (in terms of link costs c_i). In a network with N nodes and $0 < c_i \leq 1$, the link weights $w_i = N + c_i$ ensure that paths with h links always appear cheaper than paths with $h + 1$ links. In particular, h-hop routes have a maximum cost of $h(N + 1)$, while any $(h + 1)$-hop route has a cost that exceeds $(h + 1)N$, where $h \leq N$. The Dijkstra shortest-path calculation has $O(L \log N)$ complexity when implemented with a binary heap. Although advanced data structures can reduce the average and worst-case complexity [19], the shortest-path computation still incurs significant overhead in large networks. Extracting the route introduces complexity in proportion to the path length.



(a) Continuous cost functions (b) Discrete cost functions

Figure 3.1: Link-cost metric functions

3.2.2 Link-Cost Metrics

The routing algorithm uses link cost metrics $\{c_i\}$ to distinguish between paths of the same length. Previous studies suggest several possible forms for the path metric, including sum of link utilizations, maximum link utilization on the path, or sum of the link delays. Defining the path cost as the sum of link utilization reduces flow blocking probability and results in less route oscillation by adapting slowly to changes in network load [59]. Other studies have shown that assigning each link a cost that is exponential in its current utilization results in optimal blocking probability [70].

For a general model of link cost, we employ a function that grows exponentially in the link utilization ($c_i \propto u_i^\alpha$), where the exponent α controls how expensive heavily-loaded links look relative to lightly-loaded links. An exponent of $\alpha = 0$ reduces to load-independent routing, whereas large values of α favor the widest shortest paths (selecting the shortest-path route that maximizes the available bandwidth on the bottleneck link). We define the parameter $u_{\min}$ to be the minimum-cost utilization level; any link utilization below $u_{\min}$ is considered to have the minimum cost. Setting $u_{\min} = 0.5$, for example, results in a routing policy in which all links with less than 50% utilization look the same with regard to cost.

Figure 3.1(a) plots link-cost functions for several values of α with $u_{\min} = 0.1$. When $\alpha = 0$ the path-selection scheme reduces to load-independent routing, while $\alpha = 1$ selects a shortest-path with the minimum sum of link utilizations. Large values of α loosely correspond to widest shortest-path routing, since the large exponent virtually eliminates heavily-loaded links from consideration. We represent link cost with C discrete values:

$$c_i = \begin{cases} \dfrac{\left[\left(\frac{u_i - u_{\min}}{1-u_{\min}}\right)^{\alpha} \cdot (C-1)\right] + 1}{C} & u_i > u_{\min} \\ 1/C & \text{otherwise,} \end{cases}$$

as shown in Figure 3.1(b). Small values of C limit the computational and storage requirements of the shortest-path computation. However, coarse-grain link-cost information can degrade performance by limiting the routing algorithm's ability to distinguish between links with different available resources, though the presence of multiple minimum-cost routes provides efficient opportunities to balance load through alternate routing.

3.2.3 Hop-by-Hop Signaling

When a new flow request arrives, the source router applies the three-step routing algorithm to select a suitable path. However, the optional step of pruning the (seemingly) infeasible links may actually disconnect the source and the destination, particularly when the network is heavily-loaded. When a feasible route cannot be computed, the source rejects the flow without trying to signal it through the network. Stale link-state information may contribute to these *routing failures*, since the source may incorrectly prune a link that could actually support the new flow (i.e., the link has $u_i + b \leq 1$, although the source determines that $u_i' + b > 1$). Routing failures do not occur when pruning is disabled. In the absence of a routing failure, the source initiates hop-by-hop signaling to reserve bandwidth b on each link in the route. As the signaling message traverses the selected path, each router performs an admission test to check that the link can actually support the flow. If the link has sufficient resources, the router reserves bandwidth on behalf of the new flow (i.e., $u_i = u_i + b$) before forwarding the set-up message to the next link in the route.

Once the bandwidth resources are reserved on each link in the route, the network admits the flow, committing bandwidth b on each link in the path for the duration of the flow. However, a *set-up failure* occurs if a link does not have enough resources available when the set-up message arrives.

To deploy QoS routing with reasonable network overheads, the delays for propagating and processing these set-up messages must be much smaller than the link-state update periods and flow durations. In assuming that propagation and processing delays are negligible, our model focuses on the primary effects of stale link-state information on establishing connections the long-lived traffic flows. Finally, we generally model at most one attempt to signal a flow, though in some cases a single flow may attempt path set-up multiple times. Although we do not evaluate alternate routing (or crankback) after a set-up failure in Chapter 4, the flow blocking probability does provide an estimate of the frequency of crankback operations. In practice, a "blocked" request may be repeated at a lower QoS level, or the network may carry the offered traffic on a preprovisioned static route. The experiments in Chapter 5 do consider alternate routing in the context of path precomputation.

3.2.4 Link-State Update Policies

Every router has accurate information about the utilization and cost of its own outgoing links, and potentially stale information about the other links in the network. To extend beyond the periodic link-state update policies evaluated in previous performance studies [13, 56, 57, 59], we consider a three-parameter model that applies to the routing protocols in PNNI and the proposed QoS extensions to OSPF. In particular, the model includes a trigger that responds to significant changes in available bandwidth, a hold-down timer that enforces a minimum spacing between updates, and a refresh period that provides an upper bound on the time between updates. The link state is the available link bandwidth, beyond the capacity already reserved for other QoS-routed traffic (i.e., $1 - u_i$). This is in contrast to traditional best-effort routing protocols (e.g., OSPF) in which updates essentially convey only topology information. We do not assume, or model, any particular technique for distributing this information in the network; two possibilities are flooding (as in PNNI and OSPF) or broadcasting via a spanning tree.

The periodic update messages provide a refresh of the link utilization information, without regard to changes in the available capacity. Still, the predictable nature of periodic updates simplifies the provisioning of processor and bandwidth resources for the exchange of link-state information. To prevent synchronization of update messages for different links, each link introduces a small random component to the generation of successive updates [30]. In addition to the refresh period, the model generates updates upon detection of a significant change Δ_i in the available capacity since the last update message, where

$$\Delta_i = \frac{|u_i' - u_i|}{1 - u_i'}.$$

These changes in link state stem from the reservation (release) of link bandwidth during connection establishment (termination). By updating link load information in response to a change in available bandwidth, triggered updates respond to smaller changes in utilization as the link nears capacity, when the link may become incapable of supporting new flows. Similarly, flows terminating on a heavily-loaded link introduce a large relative change in available bandwidth, which generates an update message even for very large trigger thresholds. In contrast to periodic updates, though, triggered messages complicate the provisioning of network resources since rapid fluctuations in available capacity can generate a large number of link-state updates, unless a reasonable hold-down timer is used.

3.3 Network and Traffic Model

A key challenge in studying protocol behavior in wide-area networks lies in how to represent the underlying topology and traffic patterns. The constantly changing and decentralized nature of current networks (in particular, the Internet) results in a poor understanding of these characteristics and makes it difficult to define a "typical" configuration [67]. Adding to the challenge are observations that conclusions about algorithm or protocol performance may in fact vary dramatically with the underlying network model. For example, random graphs can result in unrealistically long paths between certain pairs of nodes, "well-known" topologies may show effects that are unique to particular configurations, and regular graphs may hide important effects of heterogeneity and non-uniformity [94]. Consequently, our simulation experiments consider a range of network topologies with differences in important parameters such as average path length, number of equal-hop paths between nodes, and network diameter. We comment on similarities and differences between the trends in each configuration.

As our study focuses on backbone networks, we consider topologies with relatively high connectivity, an increasingly common feature of emerging core backbone networks [37, 94], that support a dense traffic matrix (with significant traffic between most pairs of core nodes) and are resilient to link failures. Each node can be viewed as a single core router in a backbone network that sends and receives traffic for one or more sources and carries transit traffic to and from other routers. In addition to studying a representative "well-known" core topology (an early representation of the MCI backbone that has appeared in other routing studies [56, 57]), we also evaluate both random graphs and regular topologies in order to vary important parameters like size, diameter, and node degree in a controlled fashion. Most of our graphs show results for the MCI and random topologies, though we use a set of regular graphs with different degrees of connectivity to evaluate the effects of having multiple shortest-path routes between pairs of nodes. The MCI and random graphs in general have relatively few (if any) multiple equal-hop paths between nodes but paths are shorter

Topology	Nodes	Links	Degree	Diameter	Mean path length
Random graph	100	492	4.92	6	3.03
MCI backbone	19	64	3.37	4	2.34
Regular topology	125	750	6	6	3.63

Table 3.1: Topologies used in experiments

in the smaller MCI topology. The regular topology has significantly more equal-hop paths but at the expense of having more links and, consequently, higher link-state update distribution overhead. The random graph is an instance generated using Waxman's model [90]. We further assume that the topology remains fixed throughout each simulation experiment; that is, we do not model the effects of link failures. Table 3.1 summarizes the characteristics of the topologies used for experiments in Chapters 4 and 5.

Each node generates flow requests according to a Poisson process with rate λ, with uniform random selection of destination nodes. This results in a uniform traffic pattern in the regular graphs, and a non-uniform pattern on the MCI and random topologies, allowing us to compare QoS routing to static shortest-path routing under balanced and unbalanced loads. In addition, we consider the effect of bursty arrivals where flow interarrival times follow a Weibull distribution [27]. We model flow durations using a Pareto distribution with shape parameter $a = 2.5$ since that empirical studies of wide-area network traffic have shown that flow duration distributions are long- or heavy-tailed [67]. This shape parameter still produces a distribution with finite variance making it possible to gain sufficient confidence on the simulation results. We use a standard form of the Pareto distribution with shape parameter a, scale parameter β, and cumulative distribution function $F_X(x) = 1 - (\beta/x)^a$. For comparison we also conducted experiments with exponentially distributed durations. We denote the mean duration as ℓ. Flow bandwidths are uniformly-distributed within an interval with a spread about the mean $\bar{b}$. For instance, flow bandwidths may have a mean of 5% of link capacity with a spread of 200%, resulting in $b \sim U(0.0, 0.1]$. Most of the simulation experiments in Chapters 4 and 5 focus on mean bandwidths from 2–5% of link capacity. Smaller bandwidth values, albeit perhaps more realistic, would result in extremely low blocking probabilities, making it almost impossible to complete the wide range of simulation experiments in a reasonable time; instead, the experiments consider how the effects of link-state staleness scale with the $\bar{b}$ parameter to project the performance for low-bandwidth flows. With a flow arrival rate λ at each of N nodes, the offered network load is $\rho = \lambda N \ell \bar{b} \bar{h} / L$, where $\bar{h}$ is the mean distance (in number of hops) between nodes, averaged across all source-destination pairs.

3.4 Evaluation Metrics

In order to evaluate the performance and costs of QoS routing, we identify a set of metrics that serve to uncover a wide variety of primary and secondary effects. Many of the results in later chapters focus on standard metrics for performance and overhead: flow blocking probability and link-state update rate, respectively. The overall flow blocking probability gauges the ability of the QoS routing algorithm and update policies to find feasible paths for flow requests. The link-state update rate, meanwhile, gives an indication of the required bandwidth and processing overhead necessary to achieve the corresponding blocking probability. Below we describe some of the numerous other performance and overhead statistics collected by the simulator.

3.4.1 Performance Metrics

In addition to the standard overall flow blocking metric, we consider several alternate notions of blocking. For example, we record blocking relative to requested bandwidth (first defined in [57]), and relative to source-destination distance (hopcount blocking). We also consider blocking for flows between nodes that are a given hopcount apart, (e.g., blocking probability for 5-hop paths). Each of these is defined more precisely as:

$$\text{blocking} = \frac{|\mathcal{B}|}{|\mathcal{C}|} \qquad \text{bandwidth blocking} = \frac{\sum_{c \in \mathcal{B}} b_{req}(c)}{\sum_{c \in \mathcal{C}} b_{req}(c)}$$

$$\text{hopcount blocking} = \frac{\sum_{c \in \mathcal{B}} hops_{min}(c)}{\sum_{c \in \mathcal{C}} hops_{min}(c)} \qquad h\text{−hop blocking} = \frac{|\mathcal{B}_h|}{|\mathcal{C}|}$$

where $\mathcal{C}$ is the set of all flow, $\mathcal{B} \subseteq \mathcal{C}$ is the set of blocked flows, and $hops_{min}(c)$ is the shortest distance (in hops) between the source and destination of flow c. $\mathcal{B}_h$ is a subset of $\mathcal{B}$ denoting blocked flows with hopcount h.

We introduce the idea of separating overall blocking probability into its components due to routing and set-up failures, as defined in Section 3.2.3. Chapter 4 demonstrates that the distinction has important implications on overhead. Routing failures are purely local to the source during route computation, and thus are less costly than set-up failures. In particular, no bandwidth or processing resources are consumed within the network when a feasible route cannot be determined. On the other hand, when the absence of sufficient resources is discovered during signaling, as in the case of set-up failures, network resources are consumed in attempting to reserve resources along the selected path.

In addition, we measure the residual bandwidth available for sharing among all flows across a link. Higher residual bandwidth may indicate success in balancing load in the network or may arise from additional blocking that reduces the average amount of traffic admitted. This statistic is similar to that presented in [13], and is computed by recording the available bandwidth on the bottleneck link during set-up set-up for each flow, and averaging over all flows.

3.4.2 Overhead Metrics

The primary overheads in QoS routing protocols arise from three sources: link-state updates, route computation, and signaling operations. As with the performance metrics, we collect these basic statistics as well as some others that provide additional insight.

For instance, we classify link-state updates into those generated by periodic or triggered mechanisms, and those deferred due to the hold-down timer. In experiments with multiple update mechanisms, this allows us to determine which type of update is generated most frequently. We also allow examination of the additional overhead introduced by extraneous updates generated during failed set-up attempts. Finally, we record the average elapsed time between the last update generated and a successful or blocked flow request.

In the traditional on-demand routing model, the average route computation rate is identical to the flow arrival rate. When multiple path computations are allowed for a single request, or in the context of precomputed routes (see Chapter 5), we provide additional information about the average number of computation operations per successful or blocked request. We also consider the effect of source-destination distance on route computation rate, by including metrics that reflect the computation rate for flows traversing a given number of hops. This allows us to gauge whether the distance between the source and destination plays a role in inflating the route computation rate when multiple computations per request are allowed. Similar to route computation, signaling operations may be repeated for a single flow, in particular when a new route is extracted. Hence, we include similar metrics reflecting the number of signaling operations per successful or blocked request, and the per-hopcount set-up rate.

Another important measure of overhead is the amount of additional resources consumed by flows that are routed over non-minimal paths. When a large proportion of flows take paths longer than necessary (e.g., due to link-state inaccuracy), the overall routing performance may be significantly degraded. To measure the pervasiveness of non-minimal routes, we compute a ratio of the route length for each flow to the minimum hopcount possible for the source-destination pair. We average this ratio over all flow requests. When non-minimal routing is not allowed (e.g., no pruning) this ratio is always 1.

We also record the average hopcount of successful flow requests for comparison to the overall average hopcount to determine if closer source-destination pairs are successfully routed more often. Other, similar, metrics include the proportion of flows taking non-minimal routes and the average number of additional hops taken by each admitted flow.

3.4.3 Representing Performance and Overhead Metrics

Since we are interested in understanding how performance and overhead scale under stale link-state information, we avoid translating the parameters in our model to specific units. This allows us to decouple the traffic model from assumptions about signaling capacity at network routers or about the type of traffic (e.g., how short- or long-lived) that is assigned to QoS routes. Most network resource requirements are affected by two or more of our performance metrics, where the relative contribution depends on the specific configuration. For example, CPU load is affected by both set-up failures and link-state update messages, where the exact cost depends on the details of the protocol and the implementation. Our approach allows us to locate the operating regions where QoS routing can provide added benefit under reasonable overheads of signaling and link-state updates.

Similarly, we report link-state update overhead in terms of the number of updates generated per link per unit time. While it may be appealing to express this instead as bytes per second or as a fraction of network capacity, the actual overhead is dependent on the link speed, the routing protocol, and the mechanisms for exchanging link state. For example, a QOSPF advertisement for a single link requires approximately 50 bytes [97], whereas a PNNI topology state packet may require up to a few hundred bytes depending on the resource advertisement format and bundling of updates [71]. The portion of capacity consumed by these updates depends also on the link speed, and whether updates are distributed via flooding or a spanning tree. In plotting unitless metrics for overhead, we are able to examine the general scaling trends while still permitting estimation of actual overhead for a particular network and protocol configuration. In Chapter 6, however, we conduct experiments with traffic characteristics derived from real network traces, and hence report actual units with most metrics (e.g., updates per second).

3.5 Simulation Environment

To evaluate the QoS routing model we developed `routesim`, a flow-level event-driven simulator written in C with complete support for the model parameters and evaluation metrics described in Sections 3.2–3.4. In this section, we describe the high-level design of `routesim` and some of its features.

3.5.1 Simulator Design

One of our primary goals in designing `routesim` was to facilitate the study of relatively large networks of over 100 nodes while limiting the memory overheads of the simulation. These overheads are created primarily by the potentially large number of events required to capture session arrivals and departures, and individual link-state updates. Although other general-purpose simulation packages were available, we found that studying detailed routing policies and algorithms, link cost, and link-state update policies with these would require substantial customization or addition of some key missing features. Moreover, due to their focus on low-level modeling (i.e., at the packet level), the performance of these packages suffers when studying large networks. Examples of such tools include `ns-2` from the Virtual InterNetwork Testbed project [85] or OPNET from MIL 3, Inc.

In trying to reduce simulation event overhead, we specifically targeted link-state updates since they are the most unpredictable and numerous, especially when using triggered updates in highly connected topologies. We employ a "lazy" approach in evaluating updates so that network link-state is updated only when a particular link is examined by a flow. Whenever a flow request is signaled or terminated, only the links in its route are updated, if necessary, according to the current time and any change in available bandwidth. Using this approach we are able to avoid devoting any events to link-state updates. The cost is some extra bookkeeping in the link data structures, and the extra computation during signaling operations.

3.5.2 `routesim` Structure

Figure 3.2 illustrates the functional components of the `routesim` simulator. The functional divisions are influenced by the model components described in Section 3.2. The simulator "engine" is implemented by the event handlers depicted as ovals in the diagram. Algorithms and policies for routing, signaling, and link-state updates are implemented as separate functional blocks (shown as rectangles). Each handler or functional block performs specific tasks during the simulation as

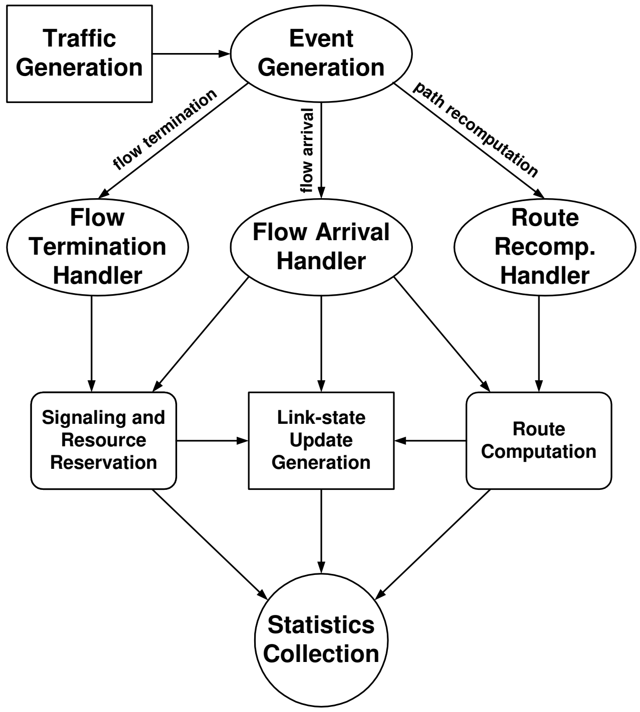


Figure 3.2: `routesim` functional components

described below.

Traffic generation: This module is responsible for determining characteristics of each flow request according to the specified traffic parameters. The traffic generation module supplies each new flow request with a source address, destination address, interarrival time relative to the last request from a given source, flow duration, and bandwidth requirement.

Event generation: All events, including flow arrivals, flow terminations, and periodic route re-computations, are generated by this module. During initialization, the event generation module bootstraps the simulation by creating an initial flow request from each source. Then, for each flow request that arrives and is handled, two new events are generated: a flow termination event (in the case of successful routing), and the next new flow request for the same source. Hence, each source typically has two outstanding events in the event queue during the simulation.

Event handlers: These handlers coordinate the routing process by performing specific functions in response to flow arrival, termination, or route computation events. The primary event handler handles each new flow request arrival event, and is further divided into a route computation handler and a signaling handler.

Signaling and resource reservation: Signaling is implemented by the flow signaling handler and the flow termination handler. The signaling handler implements resource reservation, admission control, and the signaling policies, in particular the actions to take in the case of a set-up failure. The main task of the termination handler is to release bandwidth resources on each link in the path. Both set-up and termination operations require interaction with the link-state update module to perform "lazy" updates as each link in the path is traversed.

Route computation: This module serves as a controller for all routing policy and algorithmic functions. Depending on the routing parameters, it steers each flow request to the appropriate route computation and route extraction algorithms. It also implements various routing policies, including on-demand versus precomputed routing, pruning and non-minimal routing, alternate routing, and link costs. The route computation module uses the link-state update module to perform updates as links are considered in the route computation.

Link-state update generation: Link-state updates are generated according to the three-parameter model described in Section 3.2.4. Triggered updates are invoked when available link bandwidth changes during flow set-up or teardown. When link state cannot be updated due to a hold-down timer constraint, a flag is set (with the time that the update may occur) so that later operations involving the link will reflect the most current update if the hold-down timer has expired. Also, the pending updates are reevaluated to check if the bandwidth change that invoked the original triggered update still persists. As mentioned above, the route computation module also invokes link-state update processing so that each link reflects the latest update available during path computation. In this case update processing handles only periodic or pending updates that should have occurred before the computation begins.

Statistics collection: `routesim` tracks a large number of statistics during the simulation. The statistics collection module is invoked as part of routing, signaling, and link-state update as shown in Figure 3.2. Though not shown in the figure, the traffic generation module also interfaces to the statistics collection module to measure traffic characteristics for later comparison with input traffic parameters as a means of validating that flows are correctly generated.

A major structural component in `routesim` is the network abstraction, illustrated in Figure 3.3. It consists of four parts: topology and associated parameters, distance table, connectivity information, and routing table. The available topology types include regular graphs (k-ary n-cubes), random graphs, fully-connected, or user-specified with a specification file. The network retains the same functional and data access interface regardless of the topology being used. For example, topology parameters, such as number of links, average node degree, or network diameter, are ac-

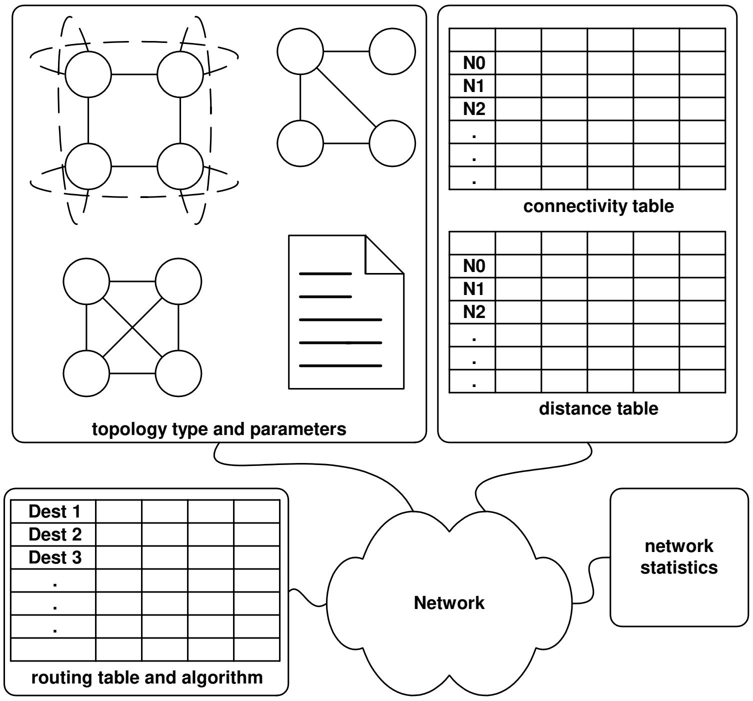


Figure 3.3: `routesim` network abstraction

cessed through a common set of function calls. For each type of topology, however, these functions are implemented in a type-specific manner. Similarly, the embedded distance table (for computing minimum-hop distance between nodes) and connectivity table (for checking connectivity between nodes) export a common interface but are initially constructed according to the topology type.

Many of the simulation statistics are associated with the network structure for for ease of access and updating. Since many operations are performed on the network, it is convenient to update statistics regarding these operations using the network structure directly . For example, since route computation, signaling, and link-state updates all perform operations on the network links, it is a simple matter to update statistics via the network structure. Examples of state kept in the network link structure include update generation time, advertised link-state metric values, current available bandwidth, and the number of flows currently occupying the link.

The routing table need not be contained in the network abstraction, but was included for convenience when studying hybrid routing (see Chapter 6) in which multiple networks with identical topologies may coexist in the simulator but use different routing algorithms. The routing table is implemented differently depending on the routing algorithm and its associated path representation, and is accessed in algorithm-specific ways.

3.5.3 `routesim` Features

The design of `routesim` has evolved as new models for traffic characteristics (in particular, traffic exhibiting high variability), routing algorithms, and network topologies were introduced. During its evolution, new problems in configuration management, simulation execution and monitoring, and debugging arose, dictating the need for additional features. Below we discuss some important `routesim` features aside from the routing model and performance metrics described in Sections 3.2–3.4.

Flexible configuration: One of the goals of the design was to to retain the ability to evaluate fundamentally different routing policies with a single simulation program. This goal requires the ability to flexibly manage a large set of configuration parameters. With such a large number of input parameters, it is important that `routesim` configuration be easily manageable and maintain its flexibility. We rely on a combination of a single configuration file to set most parameters with the ability to override individual values on the command line. In addition to some standard topology and traffic configurations, `routesim` provides a high degree of flexibility through the use of specification files. For example, users may specify the topology and individual link capacities, discrete distributions of flow durations, and a traffic matrix with entries for individual source-destination pairs, along with the ability to specify values for different times-of-day.

Multiple stopping criteria: A primary challenge in simulation of network effects in the face of high degrees of underlying variability lies in establishing stopping criteria for the simulation. To address this challenge, `routesim` employs several techniques. The simulator imposes a warm-up period before collecting any final performance statistics. When the measured blocking probability is within a 99% confidence interval subject to a user-specified threshold, we begin collecting actual statistics. We terminate the simulation (i.e., stop creating new flow requests) when the new blocking probability is within a 99% confidence interval subject to a tighter threshold (again, user-specified). Note that statistics are collected before the period when the network is "draining" due to flow terminations. Though we could end the simulation after collecting statistics, but running until all flows terminate (i.e., until the network drains) provides an additional error check on the number of termination events.

In addition to stopping criteria based on confidence intervals over a single parameter, `routesim` allows other explicit constraints to be specified. For example, it is possible to specify that the warm-up and data collection intervals are each at least several times larger than the link-state update period, even if the confidence intervals initially suggest that the simulation has converged. Similarly, stopping criteria for both warmup and actual simulation may be controlled in terms of minimum number of requests for each phase. These additional time- or request-based stopping criteria are useful in cases where traffic exhibits very high variability (e.g., infinite variance). In these situations, the confidence interval alone may not be sufficient since a "steady-state" condition may not be achievable.

Simulation monitoring: Since a `routesim` simulation on a large network is often long-running, several simple monitoring facilities are provided to gauge its progress. A monitor file is generated and updated during simulation displaying the simulation time, number of flow requests generated, mean and confidence interval endpoints for the blocking probability, and an indication of which phase (warmup or actual) is executing. In addition, the simulator may be configured to notify the operator via electronic mail when error or completion events occur.

Event tracing: `routesim` provides two essential event tracing mechanisms with different purposes. First is a layered tracing mechanism for debugging and precise examination of all simulation events. Depending on the trace level, varying degrees of information are available, ranging from only fatal error messages, to details on every simulation event. The second mechanism is a facility to generate a trace of the link-state for a number of links. The trace file tracks the link utilization over time, as well as all of the signaling and link-state update events that occur on the link during the course of the simulation.

Simulation validation: Ideally, `routesim` results should be validated by comparing with an actual implementation of QoS routing algorithms running in a real network. In the absence of such a testbed, we rely on analytical techniques to validate that various portions of the simulation give expected results. A closed-form solution for overall performance metrics such as flow blocking is intractable; instead, we develop analytic expressions for several simpler quantities. The validation approach was to compare `routesim` statistics to off-line computations of these quantities. For example, using the expression for offered load, ρ, shown in Section 3.3 it is possible to check that traffic generation is working correctly. Similarly, by setting the link-state update trigger to 0, it is possible to develop an expression for the expected update rate and compare it to the generated statistics. The route computation algorithm can be checked by comparing statistics on average path hopcount with the expected hopcount for the topology. Admittedly, these quantities (in addition to others) do not validate the overall operation of `routesim` but taken together they offer considerable evidence of correct results.

Category	Parameters
Path selection	Pruning/not-pruning, cost levels C, and exponent α
Link-state updates	Period, trigger, and hold-down timer
Network topology	Random graph, regular topology, or MCI backbone
Flow characteristics	Uniform random bandwidth b, Pareto or exponential duration, ℓ
Traffic pattern	Poisson or Weibull arrivals, λ, uniform random destinations
Performance metrics	Blocking, routing vs. set-up failures, and link-state update rate

Table 3.2: QoS routing model parameters

3.6 Summary

In this chapter we have developed a parameterized model of QoS routing, where routes depend on flow throughput requirements and the available bandwidth in the network. When a new flow request arrives, the source router computes a minimum-hop path that can support the throughput requirement, using the sum of link costs to choose among feasible paths of equal length. To provide every router with a recent view of network load, link information is distributed in a periodic fashion or in response to a significant change in the available capacity. Table 3.2 summarizes the parameters in our model.

This chapter also described the design of `routesim`, a custom simulator that implements the routing model and provides a rich environment for evaluating the performance-overhead trade-offs of QoS routing. In the next chapter we use this model to characterize the effects of stale link-state information under a wide variety of network and traffic configurations. In later chapters, we extend the model and simulator to evaluate new techniques for improving the efficiency of dynamic routing in large backbone networks.

CHAPTER 4

EVALUATING OVERHEADS OF QUALITY-OF-SERVICE ROUTING

4.1 Introduction

The performance and implementation trade-offs of QoS routing depend on the interaction between a large and complex set of parameters. For example, the underlying network topology not only dictates the number of candidate paths between each pair of nodes, but also affects the overheads for computing routes and distributing link-state information. The effects of inaccurate link-state information depend on the amount of bandwidth requested by new flows. Similarly, the frequency of link-state updates should relate to flow interarrival and holding times. Routing and signaling overheads, coupled with the presence of short-lived connectionless traffic, limit the proportion of traffic that can be assigned to QoS routes; this, in turn, affects the interarrival and holding-time distributions of the QoS-routed flows. Although a lower link-state update rate reduces network and processing requirements, stale load information incurs set-up failures, which may require additional resources for computing and signaling an alternate route for the flow. In addition, controlling overhead in large networks may require strict limits on the frequency of link-state updates and route computation, even though inaccurate information makes it very difficult to successfully reserve resources on long routes.

In this chapter, we investigate these performance issues through a systematic study of the scaling characteristics of QoS routing in large backbone networks. In contrast to recent simulation studies that compare different routing algorithms under specific network configurations [13, 33, 34, 47, 56, 57, 59, 69, 72, 73], we focus on understanding how routing performance and implementation overheads grow as a function of the network topology, traffic patterns, and link-state update policies. In Chapter 3, we constructed a detailed model of QoS routing that parameterizes the path-selection algorithm, link-cost function, and link-state update policy, based on the proposed QoS extensions to OSPF and the PNNI standard, as well as the results of previous performance studies. It should be emphasized that our study focuses on the interaction between link-state staleness and the cost-performance trade-offs of QoS-routing protocols. We consider a mixture of representative topologies, and operating regimes where flow durations and the time between link-state updates are large relative to propagation and signaling delays. The model permits a realistic evaluation of large backbone networks and the routing of the longer-lived traffic flows that are likely to employ QoS routing.

Several recent studies consider the effects of stale or coarse-grained information on the performance of QoS routing algorithms. For example, analytical models have been developed to evaluate routing in hierarchical networks where a router has limited information about the *aggregate* resources available in other peer groups or areas [38]. To characterize the effects of stale information, comparisons of different QoS-routing algorithms have included simulation experiments that vary the link-state update period [13, 56, 57], while other work considers a combination of periodic and triggered updates [69]. In particular, the work in [6] evaluates several variants of triggered updates coupled with hold-down timers. However, these studies have not included a detailed evaluation of how the update policies interact with the traffic parameters and the richness of the underlying network topology. Finally, new routing algorithms have been proposed that reduce computation and memory overheads by basing path selection on a small set of discrete bandwidth levels [39, 56]; these algorithms attempt to balance the trade-off between accuracy and computational complexity.

Based on the simulation model, Section 4.2 examines the effects of periodic and triggered link-state updates on the performance and overheads of QoS routing. The experiments evaluate several topologies to explore the impact of inaccurate information on how well a richly-connected network can exploit the presence of multiple short routes between each pair of nodes. Section 4.3 studies the impact of stale load information on the choice of link metrics for selecting minimum-cost routes for new flows. The experiments suggest guidelines for tuning link-state update policies and link-cost metrics for efficient QoS routing in high-speed networks. Section 4.4 concludes the chapter with a list of guidelines for designing efficient quality-of-service routing policies in large backbone networks.

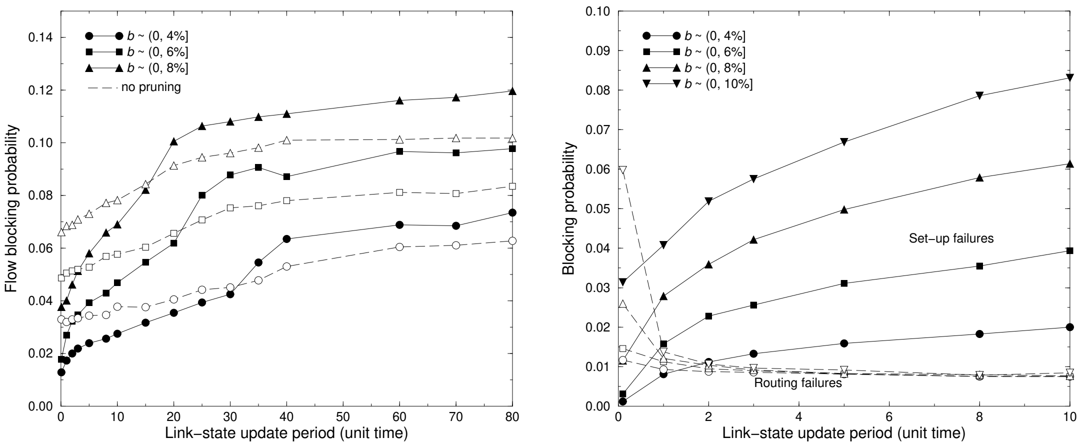
4.2 Link-State Update Policies

The initial simulation experiments focus on the effects of inaccurate link-state information on the performance and overheads of QoS routing by evaluating periodic and triggered updates in isolation. With a periodic update policy, large periods substantially increase flow blocking, ultimately outweighing the benefits of QoS routing. In contrast, experiments with triggered updates show that coarse-grain triggers do not have a significant impact on the overall blocking probability, although larger triggers shift the type of blocking from routing failures to more expensive set-up failures. The experiments also show that this shift between routing and set-up failures degrades the performance of QoS routing in richly-connected network topologies.

4.2.1 Periodic Link-State Updates

The blocking probability increases as a function of the link-state update period, as shown in Figure 4.1(a). The experiment evaluates three bandwidth ranges on the random graph with an offered load of $\rho = 0.75$; the flow arrival rate remains fixed at $\lambda = 1$, while the Pareto scale parameter, β, is used to adjust the mean duration to keep load constant across the three configurations (Pareto shape parameter, $a = 2.5$). For comparison, the graph shows results with and without pruning of (seemingly) infeasible links. We vary the update periods from almost continuous updates to very long periods of 200 times (graphs show up to 80 times) the average flow interarrival time. Due to their higher resource requirements, the high-bandwidth flows experience a larger blocking probability than the low-bandwidth flows across the range of link-state update rates. The blocking probability for high-bandwidth flows, while higher, does not appear to grow more steeply as a function of the update period; instead, the three sets of curves remain roughly equidistant across the range of update periods.

Pruning vs. not pruning: In experiments that vary the offered load, we see that pruning reduces the blocking probability under small to moderate values of ρ by allowing flows to consider nonminimal routes. However, pruning degrades performance under heavy load since these nonminimal routes consume extra link capacity, at the expense of other flows. Stale link-state information reduces the effectiveness of pruning, as shown in Figure 4.1(a). With out-of-date information, the source may incorrectly prune a feasible link, causing a flow to follow a nonminimal route when a minimal route is available. Hence, the staleness of the link-state information narrows the range of offered loads where pruning is effective, though other techniques can improve the performance. Experiments with



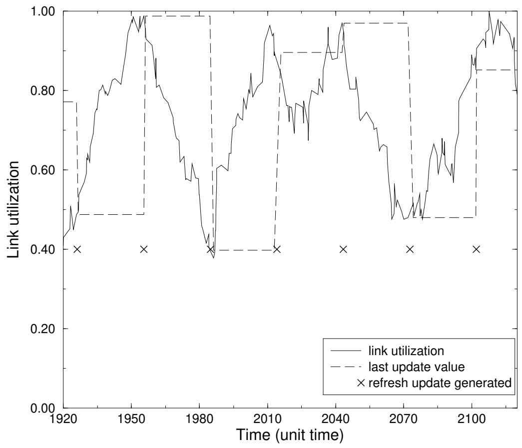
(a) Flow blocking vs. update period (b) Routing and set-up failures (with pruning)

random topology, $\lambda = 1$, $\alpha = 1$, and $\rho = .75$

Figure 4.1: Staleness due to periodic updates

different α values and topologies show the same basic trends as Figure 4.1(a), though the results with pruning disabled show a weaker dependence on the link-state update period in the MCI topology. Since the MCI topology has relatively low connectivity, most source-destination pairs do not have multiple minimum-length routes; hence, when pruning is disabled, the route computation does not react much to changes in link load. In general, the network can control the negative influence of nonminimal routes by limiting the number of extra hops a flow can travel, or reserving a portion of link resources for flows on minimal routes. To address staleness more directly, the pruning policy could more conservative or more liberal in removing links to balance the trade-off between minimal and nonminimal routes [39].

Route flapping: Although QoS routing performs well for small link-state update periods (significantly outperforming static routing [76]), the blocking probability rises relatively quickly before gradually plateauing for large update periods. In Figure 4.1(a), even an update period of five time units (five times the average flow interarrival time) shows significant performance degradation. By this point, set-up failures account for all of the call blocking, except when the update period is very small (e.g., for update periods close to the interarrival time), as shown in Figure 4.1(b) which focuses on a small region of the experiments with pruning in Figure 4.1(a). When pruning is disabled, routing failures never occur, and set-up failures account for all blocked flows. In general, periodic updates do not respond quickly enough to variations in link state, sometimes allowing substantial changes to go unnoticed. This suggests that inaccuracy in the link-state database causes the



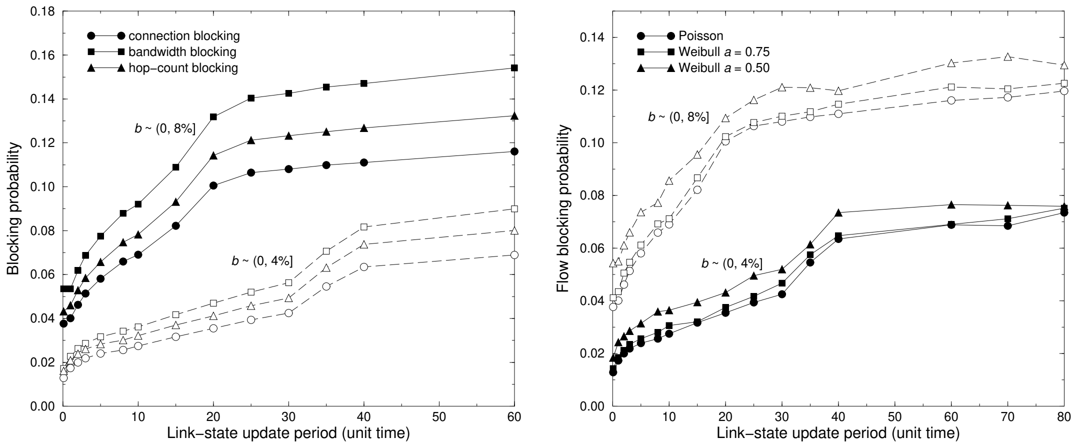
parameters as in Figure 4.1 with $b \sim (0.0, 0.06]$, update period $= 30$

Figure 4.2: Link-state fluctuations with periodic updates

source router to mistake infeasible links as feasible; hence, the source selects an infeasible path, even when there are other feasible routes to the destination. We see that routing failures occur only with very accurate information since the source learns about link infeasibility very quickly. When link-state can fluctuate significantly between updates the source is virtually certain to find at least one seemingly feasible path, thus avoiding a routing failure.

Under large update periods, relative to the arrival rates and durations, the links can experience dramatic fluctuations in link state between successive update messages, as shown in Figure 4.2. Such link-state flapping has been observed in packet routing networks, as discussed in Chapter 2, where path selection can vary on a packet-by-packet basis; the same phenomenon occurs here since the link-state update period is large relative to the flow arrival rates and durations. When an update message indicates that a link has low utilization, the rest of the network reacts by routing more traffic to the link. Blocking remains low during this interval, since most flows can be admitted. However, once the link becomes saturated, flows continue to arrive and are only admitted if other flows terminate. Blocking stays relatively constant during this interval as flows come and go, and the link remains near capacity. For large update periods, this "plateau" interval dominates the initial "climbing" interval. Hence, the QoS-routing curves in Figure 4.1(a) flatten at a level that corresponds to the steady-state blocking probability during the "plateau" interval.

Eventually, QoS routing starts to perform worse than static routing, because the fluctuations in link state begin to exceed the random variations in traffic load. In searching for (seemingly) underutilized links, QoS routing targets a relatively small set of links until new update messages arrive to correct the link-state database. In contrast, under static routing, the source blindly routes



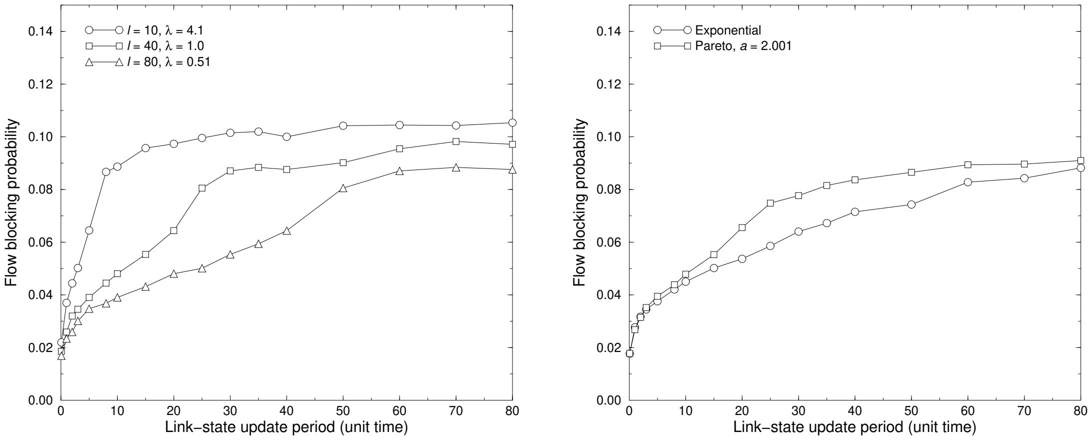
(a) Blocking metrics vs. update period (b) Blocking for varying arrival burstiness

parameters as in Figure 4.1

Figure 4.3: Blocking for different traffic types

to a single group of links, though this set is typically larger than the set identified by QoS routing. Thus, when the update period grows quite large, static routing is more successful at balancing load and reducing flow blocking. The exact crossover point between the two routing algorithms is very sensitive to the distribution of traffic in the network. For example, in the presence of "hot-spots" of heavy load, QoS routing can select links that circumvent the congestion (subject to the degree of staleness). Under such a non-uniform load, QoS routing continues to outperform static routing even for large update periods. For example, experiments with the non-homogeneous MCI backbone topology show that QoS routing consistently achieves lower blocking probability than static routing over a wide range of update rates.

Path length, high-bandwidth requests, and non-Poisson arrivals: Fluctuations in link state have a more pernicious effect on flows between distant source-destination pairs, since QoS routing has a larger chance of mistakenly selecting a path with at least one heavily-loaded link. This is especially true when links do not report their new state at the same time, due to skews in the update periods at different routers. Figure 4.3(a) illustrates this effect by comparing the flow blocking probabilities from Figure 4.1(a) to several alternative measures of blocking. Recall from Chapter 3 the hopcount blocking probability is defined as the ratio of the hopcount of blocked flows to the hopcount of all flows. The number of hops derives from the shortest-path distance between the source and destination nodes, independent of the actual (possibly longer) path selected for the flow. Bandwidth blocking is defined analogously relative to requested bandwidth. Compared to conventional flow



(a) Blocking for different mean durations (b) Blocking for different distributions ($\ell = 40$)

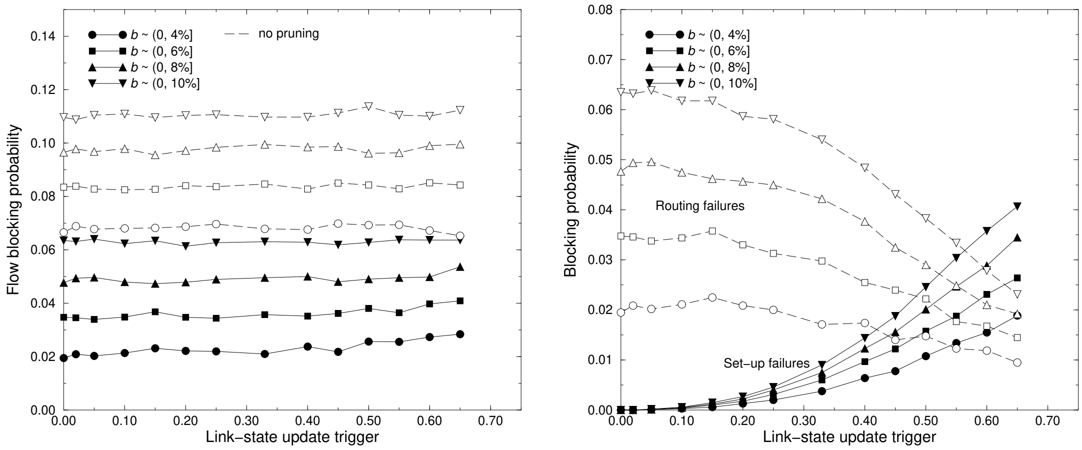
random topology, $\rho = 0.75$, $b \sim (0, .06]$, $\alpha = 1$, pruning enabled

Figure 4.4: Effects of flow duration characteristics

blocking, these metrics grow more quickly in the presence of stale information. In general, bandwidth blocking exceeds hopcount blocking, suggesting that high-bandwidth flows are even harder to route than high hopcount flows, though link-state staleness does not seem to affect one metric more than the other. Figure 4.3(b) shows that increased burstiness in the arrival process also increases blocking probability over the range of update periods. For higher bandwidth flows, the effect of burstiness is exacerbated by the nonminimal routes introduced by pruning. Thus, even for the smallest update period, the blocking for bursty traffic is significantly higher than for nonbursty traffic. The effect is not so pronounced for lower-bandwidth flows since they consume fewer resources even when allowed to take nonminimal routes.

Flow durations: Despite the fact that staleness due to periodic updates can substantially increase flow blocking, the network can limit these effects by controlling which types of traffic employ QoS routing. For example, Figure 4.4(a) shows the blocking probability for flows with different mean duration; the arrival rate λ varies to keep offered load fixed. The graph demonstrates that longer durations allow the use of larger link-state update periods to achieve the same blocking probability. Short-lived flows cause link-state to fluctuate rapidly, particularly for high-bandwidth requests, and thus require frequent updates to maintain good routing performance.

In Figure 4.4(b), we find that exponentially distributed flow durations produce a more gradual rise in blocking probability over the same range of update periods (nearly half as fast for some mean holding times) than with the Pareto distribution. The long-tailed Pareto distribution introduces more overall variability in the network load by creating some very long-lived flows (relative to the



(a) Flow blocking vs. update trigger (b) Routing and set-up failures (with pruning)

MCI topology, $\rho = 0.70$, $\lambda = 1$, $\alpha = 1$

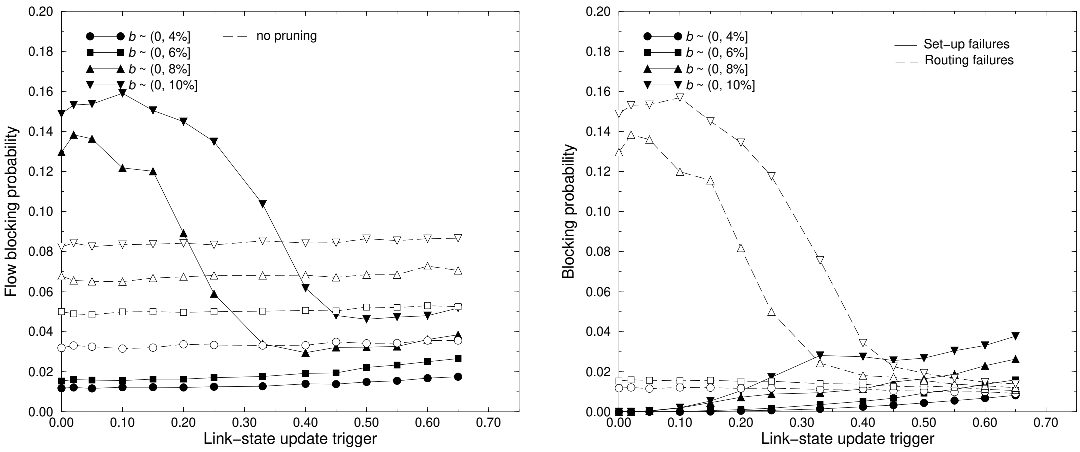
Figure 4.5: Blocking insensitivity to triggers

mean). The increased load fluctuation decreases the effectiveness of periodic link-state updates in identifying feasible paths. The heavier tail of the Pareto distribution also results in more shorter-lived flows than an exponential distribution with the same mean, implying that these shorter flows require very frequent updates to achieve acceptably low blocking probability.

These results suggest that the network could limit QoS routing to the longer-lived traffic that would consume excessive link resources if not routed carefully, while relegating short-lived traffic to preprovisioned static routes. With some logical separation of resources for short-lived and long-lived traffic, the network could tune the link-state update policies to the arrival rates and holding times of the long-lived flows. With appropriate mechanisms to identify or detect long-lived traffic, such as flow detection schemes for grouping related packets [20, 28], the network can assign this subset of the traffic to QoS routes and achieve good routing performance with a lower link-state update rate. Chapter 6 exploits these observations to introduce a new dynamic routing scheme for a portion of the traffic.

4.2.2 Triggered Link-State Updates

Although periodic updates introduce a predictable overhead for exchanging link-state information, triggered updates can offer more accurate link-state information for the same average rate of update messages. The graph in Figure 4.5(a) plots the flow blocking probability for a range of triggers and several bandwidth ranges in the MCI topology. In contrast to the experiments with



(a) Flow blocking vs. update trigger (b) Routing and set-up failures (with pruning)

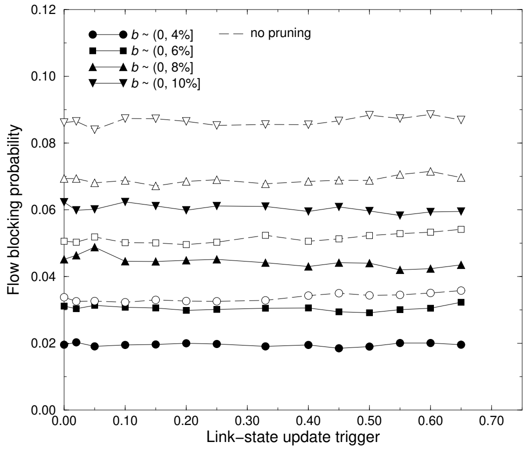
random topology, $\rho = 0.75$, $\lambda = 1$, and $\alpha = 1$

Figure 4.6: Blocking probability with triggers in random topology

periodic link-state updates, we find that the overall blocking probability remains relatively constant as a function of the trigger, across a wide range of flow bandwidths, cost metrics, and load values, with and without pruning, and with and without hold-down timers. Additional experiments with the well-connected regular topology show the same trend [76].

Blocking insensitivity to update trigger: To understand this phenomenon, consider the two possible effects of stale link-state information on the path-selection process when pruning is enabled. Staleness can cause infeasible links to appear feasible, or cause the router to dismiss links as infeasible when they could in fact support the flow. When infeasible links look feasible, the source may mistakenly choose a route that cannot actually support the flow, resulting in a set-up failure. However, if the source had accurate link-state information, any infeasible links would have been pruned prior to computing a route. In this case, blocking is likely to occur because the source cannot locate a feasible route, resulting in a routing failure. Instead of increasing the flow blocking probability, the stale information changes the nature of blocking from a routing failure to a set-up failure. Figure 4.5(b) highlights this effect by plotting the blocking probability for both routing and set-up failures. Across the range of trigger values, the increase in set-up failures is offset by a decrease in routing failures.

Now, consider the other scenario in which staleness causes feasible links to look infeasible. In this case, stale information would result in routing failures because links would be unnecessarily pruned from the link-state database. Although this case can sometimes occur, it is very unlikely,

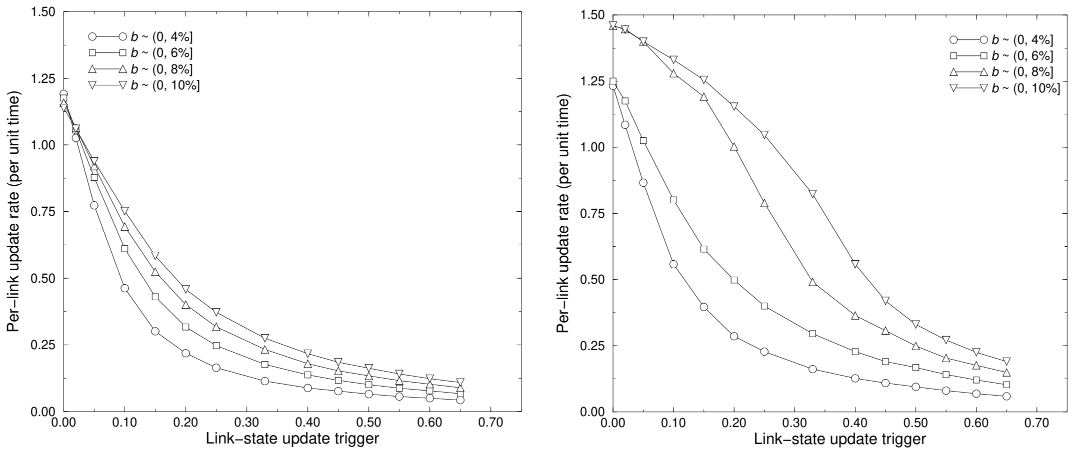


parameters as in Figure 4.6(a) with hold-down timer $= 2.0$

Figure 4.7: Blocking probability in random topology with hold-down timer

since the triggering mechanism ensures that the source router has relatively accurate information about heavily-loaded links. For example, a flow terminating on a fully-utilized link would result in an extremely large change in available bandwidth, activating most any trigger. Moreover, a well-connected topology often has more than one available route between any two nodes; the likelihood of pruning links incorrectly on *all* of the feasible routes is quite low. Hence, the blocking probability is dominated by the previous scenario, namely mistaking infeasible links as feasible. Additional experiments (not shown) illustrate that the trade-off between routing and set-up failures persists even in the presence of hold-down timers, though the hold-down timer increases the overall blocking probability and rate of set-up failures.

In loosely-connected networks, however, an incorrect pruning decision can cause the source to erroneously consider nonminimal routes. For example, Figure 4.6 shows that the random topology has *higher* blocking rates with *smaller* trigger values when trying to route high-bandwidth flows. Unlike the other two topologies, the random graph typically does not have multiple equal-length paths between a pair of nodes. As a result, pruning an infeasible link along the shortest path results in the selection of a nonminimal route. In the end, this increases the overall blocking probability, since these nonminimal routes consume extra resources. If, instead, the source chose not to prune this infeasible link (say, due to stale link-state information), then the flow would attempt to signal along the shortest path. Although the flow would block upon encountering the infeasible link, the network would benefit by deciding not to accept it. Having slightly out-of-date information has a throttling effect in this case; in fact, the use of a small hold-down timer has a similar effect, resulting



(a) Pruning disabled (b) Pruning enabled

parameters as in Figure 4.6

Figure 4.8: Link-state update rate for different trigger values

in much flatter curves for blocking as a function of the trigger, as shown in Figure 4.7.

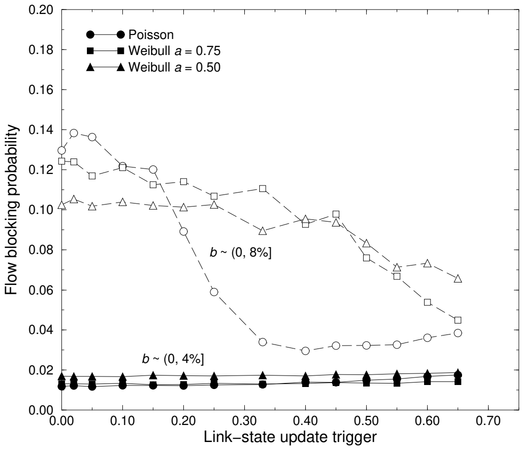
Link-state update rate: Despite the increase in set-up failures, large trigger values substantially reduce the number of update messages for a given blocking probability, as shown in Figure 4.8(a). For very fine-grained triggers, every connection establishment and termination generates an update message on each link in the route, resulting in an update rate of $2\lambda N\bar{h}/L$ in a network with N routers, L links, and an average path length of $\bar{h}$ hops. For the parameters in this experiment, the expression reduces to 1.23 link-state update messages per unit time, which is close to the y-intercept in Figure 4.8(a); additional experiments show that the link-state update rate is not sensitive to the flow holding times, consistent with the $2\lambda N\bar{h}/L$ expression. In Figure 4.8(a), the larger bandwidth values have a slightly smaller link-state update rate for small triggers because high-bandwidth flows experience a higher blocking rate, which decreases the proportion of flows that enter the network and generate link-state messages. When triggers are coarse, however, more flows are signaled in the network (due to fewer routing failures), and the high-bandwidth flows trigger more updates since they create greater fluctuation in link state.

Unlike routing failures, set-up failures can generate link-state update messages, since reserving and releasing link resources generates changes in link state, even if the flow ultimately blocks at a downstream node. The increase in set-up failures for larger triggers slows the reduction in the update rate in Figure 4.8(a) as the trigger grows. The exact effect of set-up failures depends on the number of successful hops before the flow blocks. Also, if the network supports retry (crankback) operations, the attempt to signal the flow on one or more alternate routes could generate additional link-state update messages. Finally, as a secondary effect, pruning infeasible links at the source router can inflate the update rate by selecting nonminimal routes that reserve (and release) resources on extra links, as shown in Figure 4.8(b). Overall, though, modest trigger values are effective at reducing the link-state update rate by about a factor of three to four. Also, for a fixed update rate, triggers are able to significantly reduce the proportion of set-up failures when compared with periodic updates. For instance, setting the trigger to around 0.30 results in an average update interarrival of 3 (for $b \sim (0, 0.06]$) and 17% of the blocking occurs in signaling. When using periodic updates at the same frequency, set-up failures account for 74% of the blocked flows, and the blocking rate is much higher.

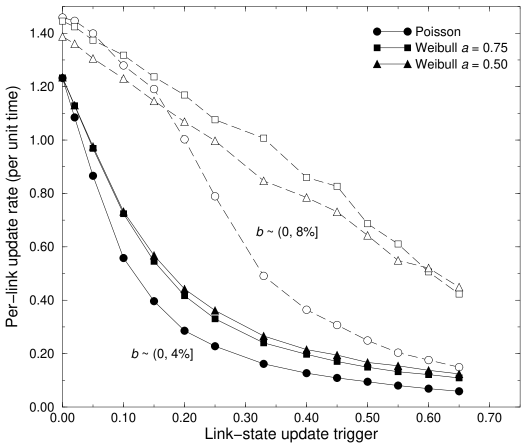
Impact of non-Poisson arrivals: Figure 4.9(a) further illustrates the problem with nonminimal routes when flow requests arrive in bursts. For both Poisson and non-Poisson arrivals of high-bandwidth requests, blocking is higher when the trigger is smaller. When link-state information is more inaccurate, the throttling effect is evident for Poisson arrivals as shown by the rapid reduction in blocking probability. This does not occur for the non-Poisson traffic, however. When a source chooses nonminimal routes for groups of requests, the network is not able to sufficiently recover from poor allocation of resources. As a result, throttling does not have an effect except for very large triggers, as shown by the gradual decline in blocking probability relative to Poisson arrivals. When pruning is not permitted, we find that blocking is insensitive to the update trigger, but bursty arrivals suffer a higher blocking probability relative to Poisson traffic.

A burst of flow requests may be thought of as a single, high-bandwidth request that, when signaled, results in more link-state fluctuation relative to non-bursty traffic. Figure 4.9(b) illustrates this through a comparison of the link-state update rate for bursty and non-bursty arrivals. With small update triggers, the bursty traffic has a lower update rate due simply to its higher blocking probability, particularly for the higher-bandwidth requests. That is, fewer flows are admitted to the network and thus link-state triggers are not set off as often. When link-state triggers become coarse, however, the update rate for bursty traffic remains high due to a combination of a higher number of nonminimal routes and increased link-state fluctuations.

In general, bursty traffic increases the blocking due to routing failures, and it remains high even as the update trigger is increased. That is, larger triggers do not shift blocking to set-up failures as in Figure 4.5. The poor resource allocation caused by bursty arrivals hampers the ability of the source to find feasible routes, especially in the random topology where the probability of having



(a) Flow blocking



(b) Link-state update rate

parameters as in Figure 4.6

Figure 4.9: Bursty arrivals with triggered updates

multiple equal-length paths is low. We find, for example, that routing failures remain high in the random topology as the update trigger increases, but they decline somewhat in the uniform topology where multiple equal-length routes are available. It is generally unwise to apply pruning for high-bandwidth flows when the topology does not have multiple routes of equal (or similar) length. The detrimental effect of nonminimal routes may be limited, however, by explicitly controlling the degree of nonminimality (e.g., at most one extra hop) rather than disabling pruning altogether.

Ultimately, the choice of link-state periods and triggers depends on the relative cost of routing failures, set-up failures, and update messages, as well as the importance of having a predictable link-state update rate. Coarse triggers and large periods can substantially decrease the processing and bandwidth requirements for exchanging information about network load. But the benefit of larger triggers and periods must be weighed against the increase in flow blocking, particularly due to more expensive set-up failures. By blocking flows inside the network, set-up failures consume processing resources and delay the establishment of other flows [35]. In addition, a failed connection temporarily holds resources at the upstream links, which may block other flows in the interim [44, 63]. In contrast, routing failures are purely local and do not consume any resources beyond the processing capacity at the source router. These trade-offs suggest a hybrid policy with a moderately large trigger value to provide load-sensitive information when it is most critical, as well as a relatively small hold-down timer to bound the peak link-state update rate without suppressing these important messages. For example, for the experiment shown in Figure 4.8(b), imposing a hold-down timer

Topology	Nodes	Links	Degree	Diameter	Mean path length
10-ary 2-cube	100	400	4	10	5.05
5-ary 3-cube	125	750	6	6	3.63
4-ary 3-cube	64	384	6	6	2.95
5-ary 2-cube	25	100	4	4	2.50

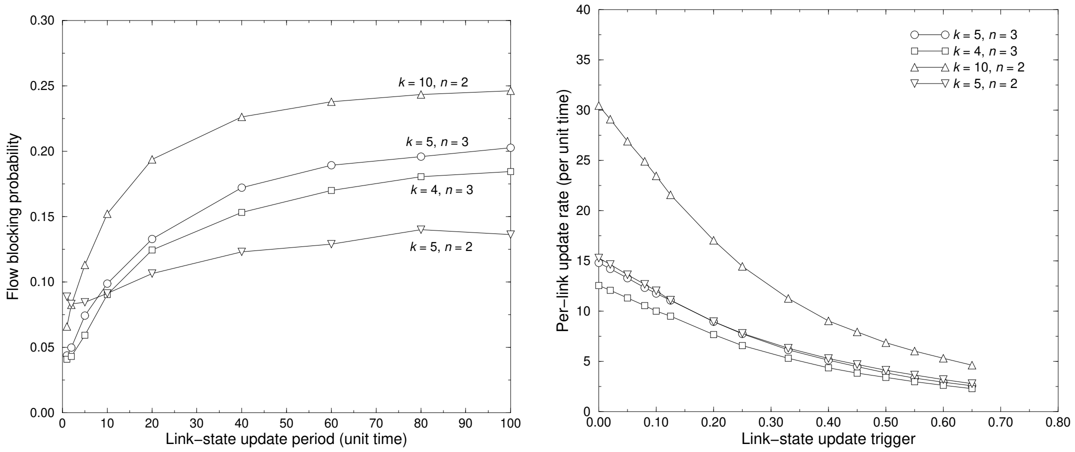
Table 4.1: Characteristics of k-ary n-cubes

two times the flow interarrival time reduces the number of link-state updates by nearly a factor of 3 for fine-grained triggers (around 5%). With coarser triggers, the hold-down timer still reduces the update rate but the decrease is not as dramatic.

4.2.3 Network Topology

The impact of stale link-state information depends on the underlying network topology. To study the effects of staleness under a range of topologies, we evaluate a set of regular graphs with similar size and different degrees of connectivity under the same uniform traffic load. The experiments focus on the class of k-ary n-cube graphs with k nodes along each of n dimensions ($N = k^n$ nodes of degree $2n$, with $L = 2nk^n$ links and diameter $D = \lfloor k/2 \rfloor n$), as shown in Table 4.1. A higher dimension (n) typically implies a "richer" topology with more flexibility in selecting routes, whereas a large number of nodes (k), with a fixed n, increases the average length of routes, which requires flows to successfully reserve resources on a larger number of links. These differences between the topologies have a significant influence on how well the QoS-routing algorithm's performance scales with the staleness of link-state information, as shown in Figure 4.10(a). The graph plots the flow blocking probability over a range of update periods, with offered load kept constant between the four configurations by changing the mean (exponentially distributed) holding time.

Under accurate link-state information, the 10-ary 2-cube and 5-ary 3-cube topologies exhibit good performance, despite the longer average distance between pairs of nodes. For small update periods, the higher connectivity of the 5-ary 3-cube and 4-ary 3-cube results in a large number of possible routes, which reduces the likelihood of a routing failure. Similarly, the 10-ary 2-cube has a large number of routes, though fewer than the 5-ary 3-cube. However, the performance of these richer topologies degrades more quickly under stale load information. For larger link-state update periods, blocking stems mainly from set-up failures, which are more likely when a flow has a longer path through the network. Once the routing algorithm selects a single path, based on stale information, the new flow can no longer capitalize on the presence of other possible routes.



(a) Blocking vs. update period ($\lambda = 1$) (b) Update rate vs. trigger level ($\lambda = 12.5$)

$\rho = 85\%$, $b \sim (0, 0.1]$, and $\alpha = 2$, pruning enabled

Figure 4.10: Interaction of topology and link-state staleness

The performance of the 5-ary 2-cube degrades more slowly, since the shorter route lengths increase the chance that the routing algorithm selects a feasible path. Similarly, though they have identical connectivity, the 4-ary 3-cube outperforms the 5-ary 3-cube, due to its smaller average path length.

Direct comparisons between the four topologies are somewhat difficult, due to differences in the number of routers and links. For example, the crossover in Figure 4.10(a) occurs because the 5-ary 2-cube has a lower average path length, despite the topology's poorer connectivity. Still, varying k and n lends insight into the effects of stale information. Figure 4.10(b) shows the overheads for triggered link-state updates in the four topologies. Although the 10-ary 2-cube has fewer routers than the 5-ary 3-cube topology, the 10-ary 2-cube generates substantially more link-state update messages than the other two networks. The larger update rate stems from the long path lengths, relative to the number of routers. Interestingly the 5-ary 3-cube and the 5-ary 2-cube have nearly identical link-state update rates. In fact, the update rate is approximately $\lambda k/4$ across all of the k-ary n-cube topologies. Drawing on the analytic expression from Section 4.2.2 and the average path length in a k-ary n-cube network, the link-state update rate should be

$$2\lambda N\overline{h}/L = 2\lambda k^n \frac{k^2-1}{4k} n \frac{N-1}{N} / 2nk^n = \frac{\lambda(k^2-1)}{4k}\frac{N-1}{N} \approx \lambda(k^2-1)/4k \approx \lambda k/4,$$

for a fine-grain trigger, and assuming odd values of k (a similar expression holds for even values of k). This update expression is proportional to k and independent of n. Increasing the network dimension n corresponds to growing the underlying network in a manner that increase the average path length in proportion to the increase in the number of links.

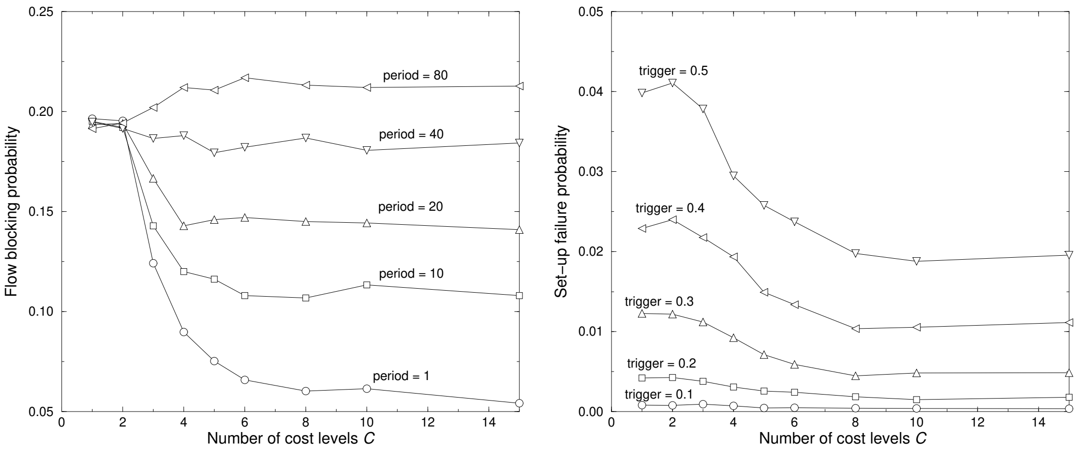
More generally, a densely-connected topology with a relatively low diameter should trigger fewer link-state updates since flows are routed on shorter paths. The reduction in overhead, however, may be offset by the cost of distributing the update messages. For example, if link-state messages are flooded throughout the network (as in OSPF and PNNI), then each router receives the message on each incoming link. As a result, each router receives $2n$ copies of every link-state update. Hence, the advantages of a richer topology are partially overshadowed by the cost of flooding the link-state messages. Also, the experiment in Figure 4.10(a) shows that stale information limits the benefits of richer connectivity, though the use of update triggers, instead of periods, can mitigate these effects. With an efficient update-distribution mechanism (such as spanning trees, instead of a flooding protocol), a richly-connected topology, coupled with a reasonable trigger level, can retain the advantage of having many routing choices and a low link-state update overhead.

4.3 Link-Cost Parameters

The link-state update rate also impacts the choice of the link-cost parameters (C and α) in the routing algorithm. Fine-grain cost metrics are much less useful, and can even degrade performance, in the presence of stale link-state information. With a careful selection of the exponent α, the path-selection algorithm can reduce the number of cost levels C without increasing the blocking probability. Smaller values of C reduce the space and time complexity of the route computation, allowing the QoS-routing algorithm to scale to larger network configurations. In addition, coarse-grain link costs increase the likelihood of having multiple minimum-cost routes, which allow the network to balance load across alternate routes.

4.3.1 Number of Cost Levels (C)

The experiments in Section 4.2 evaluate a link-cost function with a large number of cost levels, limited only by machine precision. With such fine-grain cost information, the path-selection algorithm can effectively differentiate between links to locate the "cheapest" shortest-path route. Figure 4.11(a) evaluates the routing algorithm over a range of cost granularity and link-state update periods. To isolate the effects of the cost function, the routing algorithm does not attempt to prune (seemingly) infeasible links before invoking the shortest-path computation. The C cost levels are distributed throughout the range of link utilizations by setting $u_{\min} = 0$. Compared to the high blocking probability for static routing ($C = 1$), larger values of C tend to decrease the blocking



(a) Blocking for periodic updates (b) Set-up failures for triggered updates

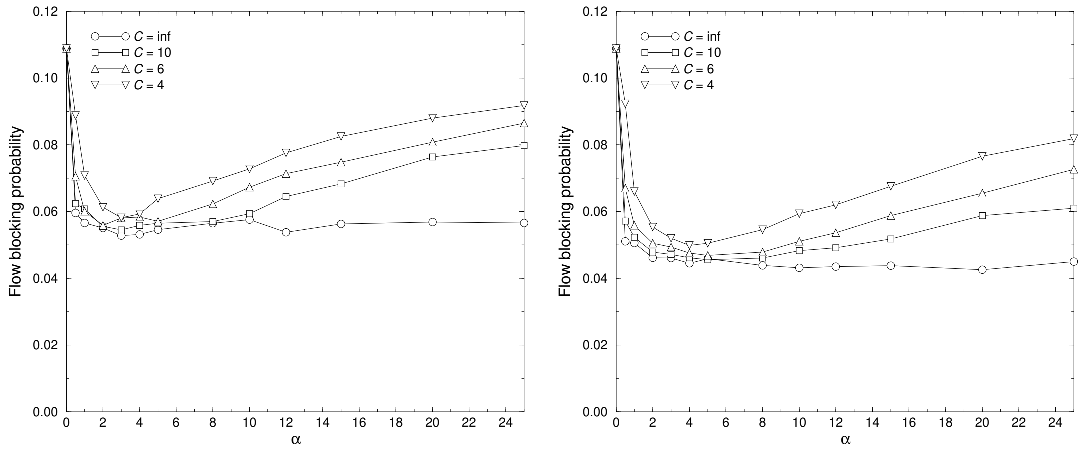
5-ary 3-cube, $\rho = 0.85$, $b \sim (0.0, 0.1]$, $\lambda = 1$, $\ell = 28$ (exponential), $\alpha = 1$

Figure 4.11: Discretized costs with stale link-state information

rate, particularly when the network has accurate link-state information, as shown in the "period=1" curve in Figure 4.11(a).

Fine-grain cost metrics are less useful, however, when link-state information is stale. For example, having more than four cost levels does not improve performance once the link-state update period reaches 20 times the average interarrival time. Although fine-grain cost metrics help the routing algorithm distinguish between links, larger values of C also limit the number of links that the routing algorithm considers, which can cause route flapping. In contrast, coarse-grain cost information generates more "ties" between the multiple shortest-path routes to each destination, which effectively dampens link-state fluctuations by balancing the load across several alternate routes. In fact, under stale information, small values of C can sometimes outperform large values of C, but this crossover only occurs once the update period has grown so large that QoS routing has a higher blocking probability than static routing. The degradation in performance under high update periods is less significant in the MCI and random topologies, due to the lower likelihood of having multiple minimum-hop paths between pairs of nodes.

The appropriate number of cost levels depends on the update period and the flow bandwidth requirements, as well as the overheads for route computation. Larger values of C increase the complexity of the Dijkstra shortest-path computation without offering significant reductions in the flow blocking probability. Fine-grain cost information is more useful in conjunction with triggered link-state updates, as shown in Figure 4.11(b). We still find, however, that experiments with a



(a) Blocking for varying C (period = 10) (b) Blocking for varying C (trigger = 0.2)

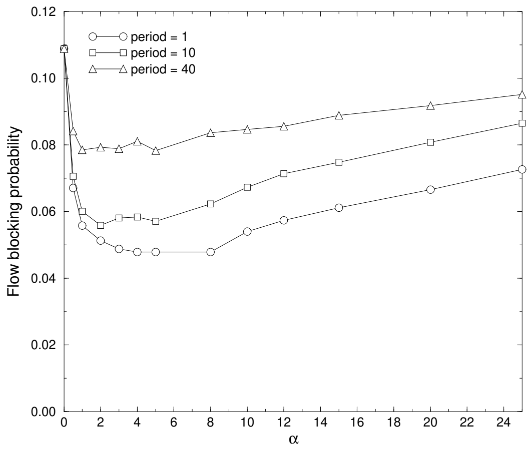
random topology, $\rho = 0.75$, $\lambda = 1$, $b \sim (0.0, 0.06]$, $\ell = 40.6$, pruning disabled

Figure 4.12: Exponent α with stale link-state information

finite number of C values are consistent with the results in Section 4.2.2; that is, the flow blocking probability remains constant over a wide range of triggers. Since the trigger value does not affect the overall blocking probability, Figure 4.11(b) plots only the set-up failures. In contrast to the experiment with periodic updates, increasing the number of cost levels beyond $C = 4$ continues to reduce the blocking rate. Since triggered updates do not aggravate fluctuations in link state, the fine-grain differentiation between links outweighs the benefits of "ties" between shortest-path routes. Although larger values of C reduce the likelihood of set-up failures by a factor of two, increasing the number of cost levels eventually offers diminishing returns.

4.3.2 Link-Cost Exponent (α)

To maximize the utility of coarse-grain load information, the cost function should assign each cost level to a critical range of link utilizations. Under fine-grain link costs (large C), the exponent α does not have a significant impact on performance; values of $\alpha \geq 1$ have nearly identical performance, as shown by the "$C = \infty$" curve in Figure 4.12(a). These results hold across a range of link-state update periods, suggesting that large values of α do not introduce much extra route flapping. This has important implications for path selection algorithms, since it suggests that widest shortest-path and cheapest shortest-path should have similar performance under stale link-state information. However, the choice of exponent α plays a more important role in cost-based routing with coarse-grain link costs, as shown by the other curves in Figure 4.12. Each plot shows a sharp



parameters as in Figure 4.12(a) with $C = 6$

Figure 4.13: Exponent α with fixed C under increased staleness

drop in the blocking probability due to the transition from static routing ($\alpha = 0$) to QoS routing ($\alpha > 0$), followed by an increase in blocking probability for larger values of α. When α is too large, the link-cost function concentrates most of the cost information in a very small, high-load region.

For large α and small C, some of the cost intervals are so narrow that the arrival or departure of a single flow could change the link cost by one or more levels. For example, when $\alpha = 8$ and $C = 10$, the link-cost function has four cost levels in the 90–100% range. This sensitivity exacerbates route flapping and also limits the routing algorithm's ability to differentiate between links with lower utilization. Further experiments (not shown) demonstrate that pruning lowers the differences between the curves for different C values. This occurs because pruning provides additional differentiation between links, even for small values of C. We also explored the effects of the link-state update period on the flow blocking probability as α is increased, for a fixed value of C. Interestingly, Figure 4.13 shows that larger update periods dampen the detrimental effects of large values of α, resulting in flatter curves than the plots in Figure 4.12(a). Although large values of α limit the granularity of the cost information, the drawback of a large value of α is largely offset by the benefit of additional "ties" in the routing algorithm when information is stale. Hence, the selection of α is actually more sensitive when the QoS-routing algorithm has accurate knowledge of link state.

4.4 Summary

The performance and complexity of QoS routing depends on the complex interaction between a large set of parameters. This chapter has investigated the scaling properties of source-directed link-state routing in large core networks. Our simulation results show that the routing algorithm, network topology, link-cost function, and link-state update policy each have a significant impact on the probability of successfully routing new flows, as well as the overheads of distributing network load metrics. Below we categorize our key observations into three areas.

Our key observations regarding QoS routing and update policies are:

- **Periodic link-state updates:** The staleness introduced by periodic link-state update messages causes flapping that substantially increases the rate of set-up failures. This increases flow blocking and also consumes significant resources inside the network, since most of the failures occur during connection set-up instead of during path selection. In extreme cases with large update periods, QoS routing actually performs worse that load-independent routing, due to excessive route flapping. Our results show that a purely periodic link-state update policy cannot meet the dual goals of low blocking probability and low update overheads in realistic networks.

- **Triggered link-state updates:** Triggered link-state updates do not significantly affect the overall blocking probability, though coarse-grain triggers do increase the amount of blocking that stems from more expensive set-up failures. Triggers reduce the amount of unnecessary link-state traffic but require a hold-down timer to prevent excessive update messages in short time intervals. However, larger hold-down timers increase the blocking probability and the number of set-up failures. Hence, our findings suggest using a combination of a relatively coarse trigger with a modest hold-down timer.

- **Pruning infeasible links:** In general, pruning infeasible links improves performance under low-to-moderate load by allowing flows to consider nonminimal routes, and avoiding unnecessary set-up failures by blocking more flows in the route computation phase. However, under heavy load, these nonminimal routes consume extra link resources, at the expense of other flows. Pruning becomes less effective under stale link-state information, loosely-connected topologies, and high-bandwidth flows, since these factors increase the amount of traffic that follows a nonminimal route, even when a minimal route is feasible. These results suggest that large networks should disable pruning or explicitly control path nonminimality, unless most source-destination pairs have multiple routes of equal (or near equal) length. Alternatively, the network could impose limits on the resources it allocates to nonminimal routes.

The intrinsic staleness of link-state information suggests new policies for alternate routing after a set-up failure. For example, introducing a small delay before trying an alternate route for a connection [27, 63] is particularly appropriate when the arrival of fresh link-state information has the potential to improve the routing decision.

Different types of network traffic and QoS requests exhibit significant effects on QoS routing performance and overheads:

- **Bandwidth and hopcount:** Flows with large bandwidth requirements experience higher blocking probabilities, but the effects of increasing link-state staleness are only slightly worse relative to lower-bandwidth flows. However, stale link-state information has a strong influence on flows between distant source-destination pairs, since long paths are much more likely to have at least one infeasible link that looks feasible, or one feasible link that looks infeasible. These effects degrade the performance of QoS routing in large network domains, unless the topology is designed carefully to limit the worst-case path length.

- **Flow durations:** Longer flow durations change the timescale of the network and allow the use of larger link-state update periods. Stale information has a more dramatic effect under heavy-tailed distributions, due to the relatively large number of short-lived flows for the same average duration and the additional variability introduced in network load, compared to exponentially distributed durations. Our findings suggest that the networks should limit QoS routing to long-lived flows, while carrying short-lived traffic on preprovisioned static routes. The network can segregate short- and long-lived traffic by partitioning link bandwidth for the two classes, and detecting long-lived flows at the edge of the network. This technique is designed and evaluated in detail in Chapter 6.

- **Flow arrivals:** Bursts of flow requests behave like single arrivals of very high-bandwidth flows, causing higher fluctuation in link-state and greater susceptibility to poor resource allocation. When uncontrolled pruning is enabled, routers choose significantly more nonminimal paths relative to non-bursty traffic, increasing blocking probability and link-state update rate. Effects of bursty arrivals are especially harmful in topologies with a limited number of equal-hop routes.

The network configuration, including routing algorithm parameters, also play an important role:

- **Rich network topologies:** The trade-off between routing and set-up failures also has important implications for the selection of the network topology. Although dense topologies offer more routing choices, the advantages of multiple short paths dissipate as link-state information becomes more stale. Capitalizing on dense network topologies requires more frequent link-state updates, and techniques to avoiding excessive link-state traffic. For example, the network could broadcast link-state updates in a spanning tree, and piggyback link-state information in signaling messages.

- **Coarse-grain link costs:** Computational complexity can be reduced by representing link cost by a small number of discrete levels without significantly degrading performance. This is especially true when link-state information is stale, suggesting a strong relationship between temporal and spatial inaccuracy in the link metrics. In addition, coarse-grain link costs have the benefit of increasing the number of equal-cost routes, which improves the effectiveness of alternate routing. We exploit these characteristics in our recent work on precomputation of QoS routes [77].

- **Exponent alpha:** Under fine-grain link costs (large C), routing performance is not very sensitive to the exponent α; an exponent of 1 or 2 performs well, and larger values do not appear to increase route flapping, even under very stale information. These results suggest that selecting routes based on widest shortest-paths or cheapest shortest-paths would have similar performance under stale link state. Coarse-grain link costs require more careful selection of α, to ensure that each cost level provides useful information, and that the detailed cost information is focused in the expected load region.

These observations represent an crucial step in understanding and controlling the complex dynamics of quality-of-service routing under stale link-state information. We find that our distinction between routing and set-up failures, and simulation experiments under a wide range of parameters, provide valuable insights into the underlying behavior of the network. In the next two chapters, we focus on exploiting these trends to reduce the computational and protocol overheads of QoS routing in large backbone networks.

CHAPTER 5

EFFICIENT PRECOMPUTATION OF QUALITY-OF-SERVICE ROUTES

5.1 Introduction

Chapter 4, as well as most previous research on QoS routing, has investigated *on-demand* policies that compute a QoS path at flow arrival. Recent work considers *precomputation* or *path caching* schemes that attempt to reduce set-up delay and amortize the overheads of route computation by reusing the paths for multiple flow requests [39, 47, 53, 56, 68]. Path precomputation introduces a trade-off between processing overheads and the quality of the routing decisions. Previous work on precomputed routes has focused on quantifying this trade-off and developing guidelines for when to recompute routes. We extend this work by presenting efficient *mechanisms* for precomputing paths under source-directed link-state routing. In particular, this chapter addresses several important practical questions:

- How should the precomputed routes be stored?
- How should multiple routes be computed?
- How much work should be performed at flow arrival?
- How should routing and signaling overheads be limited?

We answer these questions with four key elements that are described in detail in Section 5.2:

- **Coarse-grain link costs:** Path selection is based on link-cost metrics, which are a function of link-state information. Limiting link costs to a small number of values reduces the computational complexity of the path-selection algorithm. As shown in Chapter 4, coarse-grain link costs do not significantly degrade performance, and increase the likelihood of having more than one minimum-cost route to a destination.

- **Precomputation of minimum-cost graph:** Each router or switch precomputes a compact data structure that stores all minimum-cost routes to each destination. A small modification to Dijkstra's shortest-path algorithm can locate all minimum-cost routes to each destination. Instead of storing the precomputed paths in a cache or table, route extraction is postponed until flow arrival.

- **Route extraction with feasibility check:** As part of extracting a route, the source checks the feasibility of each link, based on the most recent link-state information and the bandwidth requirement of the new flow. The algorithm performs a depth-first search through the Dijkstra data structure, extracting the first route in the common case. Then, the source initiates signaling in the network to reserve resources along the selected path.

- **Reranking of multiple routes:** The depth-first extraction algorithm imposes an implicit ordering of the links when a node appears in multiple minimum-cost paths. As part of route extraction, the source can improve the path-selection process for the next flow by reranking these links to avoid duplicating work in the search for a feasible path. This also provides a simple framework for a number of alternate-routing policies.

These mechanisms enable a wide range of policies for when to compute new routes, how many candidate routes to try for a new flow, and how often to update link-state information. Coupled with our efficient routing mechanisms, these policy decisions allow significant reduction of processing overheads and set-up delay, in comparison to traditional on-demand algorithms. In addition, the simulation experiments in Section 5.3 show that the feasibility check, and the potential for multiple candidate routes, results in lower likelihood of rejecting requests and lower signaling overheads, in relation to other precomputation schemes. The performance evaluation also investigates the influence of the network topology and link-state update mechanisms on the effectiveness of the path-selection algorithm and routing policies. Section 5.4 compares our approach to related work on QoS route precomputation, and Section 5.5 concludes the chapter with a discussion of future research directions.

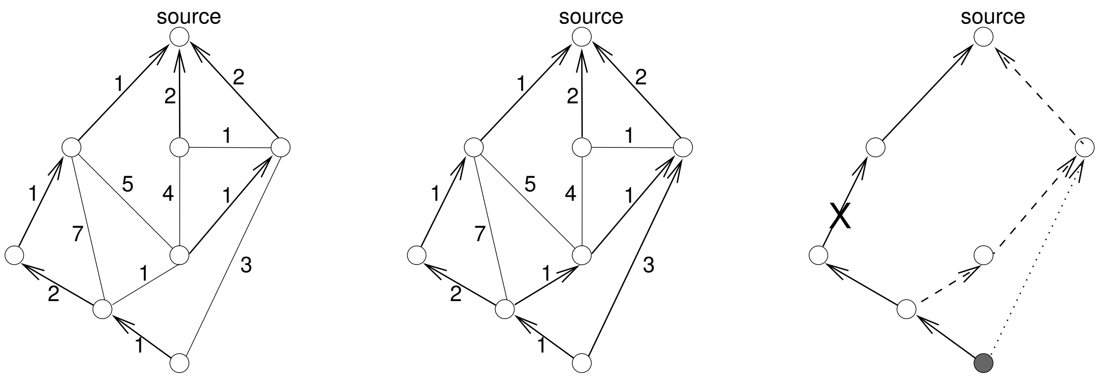
5.2 Precomputation of QoS Routes

Reducing the overheads of route computation requires careful consideration about how much information and processing are involved on various time scales. The source receives the most accurate information about network load and flow resource requirements upon the arrival of new link-state updates and flow requests. To lower complexity, though, our precomputation scheme does not perform any work upon receiving a link-state update message, beyond recording the new load information, and only modest work on flow arrival. Instead, most of the processing is relegated to background computation of a shortest-path graph of routes to each destination. By delaying route extraction until flow arrival, we exploit the most recent link-state to find a suitable path while incurring only modest overhead. Allowing occasional triggered recomputation for individual flow requests further improves the ability to find a path and to select better routes for future flows.

5.2.1 Compact Storage of Precomputed Routes

As in Chapter 4, our study of path precomputation focuses on intra-domain routing in large, flat networks with N nodes (routers or switches) and L links. We work within a similar context, focusing on topologies with relatively high connectivity, and on multimedia traffic that requires throughput guarantees. The algorithms we consider belong to the class of source-directed link-state routing. Recall that in this model, each router knows the underlying topology and has (possibly out-of-date) information about the unreserved bandwidth on each link. Link state is flooded periodically, or in response to a significant change in available bandwidth, ensuring that the routers have fairly accurate knowledge of network load. The source router selects a route for an arriving flow, based on the link state and the flow's bandwidth requirement. Route computation is based on the Dijkstra shortest-path algorithm, where link cost (or "distance") is a function of the link load. To minimize resource requirements and end-to-end delay, we focus on link-cost functions that favor routes with a small number of links.

The Dijkstra shortest-path algorithm computes a route to a destination node in $O(L \log N)$ time, when implemented with a binary heap [22]. Although advanced data structures can reduce the average and worst-case complexity [19], the shortest-path computation still incurs significant overhead in large networks. In computing a route to each destination, the Dijkstra algorithm generates a shortest-path graph, where each node has a parent pointer to the upstream node in its route from the source, as shown in Figure 5.1(a). Extracting a route to a particular destination involves traversing



(a) Shortest-path graph (b) All-shortest-path graph (c) Route extraction

Figure 5.1: Delayed pruning of shortest-path routes

these parent pointers, and introduces complexity in proportion to the path length. For on-demand routing of a single flow, the construction of the shortest-path graph terminates upon reaching the destination. Continuing the computation to generate a route for every destination, however, does not significantly increase the processing requirements, and the complexity remains $O(L \log N)$. This allows path precomputation schemes to amortize the overhead of the shortest-path calculation over multiple destination nodes. Thus, even if some source-destination pairs never communicate, computing their associated routes does not impose much extra cost.

Path precomputation schemes typically store one or more routes in a cache or table. In hop-by-hop routing, the router stores only the next hop of the route to each destination, allowing for simple storage in a route table. In contrast, source-directed routing typically requires the source to maintain a variable-length list of the links on the path to the destination. This list (or stack) becomes part of the signaling message that establishes a connection (e.g., a "designated transit list" in PNNI, or an "explicit route advertisement" in the QoS extensions to OSPF), instructing each intermediate node to forward the message to the next link on the route. To precompute routes, the source could generate paths to one or more destinations and store each route list in a cache, occasionally recomputing one or more routes based on the most recent link-state information. However, computing and extracting multiple routes introduces computational and storage complexity, particularly if some routes are never used. In addition, since link-state information changes over time, these cached routes must be invalidated and/or recomputed periodically. Instead of storing paths in a separate data structure, we consider a compact representation that maintains the routes directly in the shortest-path graph, as shown in Figure 5.1(a).

```
Heap = set of all N nodes;
while (Heap is not empty) {                          # visit each node
    u = PopMin(Heap);
    foreach node v ∈ Neighbors(u) {                  # relax each link
        if (dist[u] + cost[u,v] == dist[v])          # equal route
            parent[v] = parent[v] ∪ {u};
        else if (dist[u] + cost[u,v] < dist[v]) {    # cheaper route
            parent[v] = {u};
            dist[v] = dist[u] + cost[u,v];
        }
    }
}
```

Figure 5.2: Dijkstra algorithm with multiple parent pointers

5.2.2 Precomputation of Multiple Minimum-Cost Routes

Path precomputation schemes benefit from having multiple candidate routes to each destination, to balance network load and have additional routing choices in case of a set-up failure. However, the multiple routes should have similar cost, to avoid selecting paths that make inefficient use of network resources. A router could conceivably precompute the $k > 1$ shortest paths (in terms of hopcount or other cost) to each destination. Alternatively, a routing algorithm could compute all paths within some additive or multiplicative factor ϵ of the best path. However, these approaches introduce considerable computational complexity. For example, computing the k shortest paths for a single destination in a directed graph has complexity as high as $O(kN^3)$ [52, 93]. In addition, the k shortest paths (or paths within ϵ of optimal) to one node may not be part of the best routes to other destinations; hence, it is usually not possible to store these multiple routes in a compact, shortest-path graph representation.

Instead, we focus on an efficient special case of computing multiple *equal-cost* paths to each destination. This formulation permits a compact representation of the multiple routes, as shown in Figure 5.1(b). Computing all minimum-cost routes requires a small modification to the traditional Dijkstra computation to store multiple parent pointers for each node; each pointer identifies an upstream node along a shortest path to the destination. The nodes and parent pointers form a directed, acyclic subgraph of the original graph, rooted at the source router. Each node in the graph has a list of its parent pointers, or a bit-mask to indicate which upstream nodes reside on shortest-path routes. We maintain the parent pointers in a circular list to facilitate simple traversal of the graph when extracting routes. Figure 5.2 summarizes the algorithm, which is similar to a traditional Dijkstra computation, except for the addition of lines 4–5. The algorithm uses a heap to visit the nodes in order of their distance from the source, and relaxes each outgoing link. Initially, each node u has distance dist[u]=∞, except for the source which has a distance of 0. Starting with the source, the algorithm visits the node u with the smallest distance and considers the cost cost[u,v] of the link to each neighboring node v, extending or reassigning the set of parent pointers parent[v] if the new route has equal or lower cost, respectively. At the end, each connected node has one or more parent pointers to upstream nodes along minimum-cost routes from the source. To minimize the storage requirements and complexity of route extraction, the algorithm can impose a limit on the number of parents for each node (e.g., 2 or 3).

The likelihood of having multiple minimum-cost routes to a destination depends on the link cost function and the underlying network topology. To increase the chance of "ties" in the route computation, we discretize the link costs and map them into a small number of values, C (say, 5 or 10). This approach offers a much cheaper way to compute near equal-cost paths without resorting to a k-shortest path computation. Although fine-grain link costs (larger values of C) usually result in lower blocking probabilities, a moderately coarse link-cost function does not significantly degrade performance, particularly if link-state information is stale (see Section 4.3 in Chapter 4). When link-state information is somewhat out-of-date, the benefit of a fine-grain link cost function is greatly diminished. Perhaps more importantly, coarse-grain link costs reduce the processing requirements of the shortest-path computation by reducing heap complexity. The complexity of Dijkstra's algorithm, and the variation in Figure 5.2, decreases from $O(L \log N)$ to $O(L + CN)$ [22], while more advanced data structures offer even further reduction [19]. Hence, these coarse-grain link costs become particularly appropriate in large, well-connected networks (large L and N) where the routers have stale link-state information.

5.2.3 Delayed Extraction of Precomputed Routes

Rather than extracting and caching routes in a separate data structure, we store the precomputed routes in the shortest-path graph and delay the extraction operation until a flow request arrives. During the path extraction the source applies the most recent link-state information in selecting a route. The extraction process operates on the subgraph of nodes and parent pointers along minimum-cost routes to the destination, as shown in Figure 5.1(c). Though the router could conceivably run a new Dijkstra computation on this subgraph to select the "best" precomputed route, we instead optimize for the common case of extracting the first route by following a single set of parent pointers

```
 1:  push(dest);                                  # start at dest
 2:  ptr[dest] = head[dest];                      # start at 1st parent
 3:  while ((top != NULL) and (top != src)) {     # until reaching src
 4:      new = ptr[top].nodeid;                   # explore parents of top
 5:      if (feasible(new, top)) {
 6:         push(new);                            # explore next node
 7:         ptr[new] = head[new];
 8:      }
 9:      else {
10:         while (ptr[top].next == head[top])
11:            pop();                             # backtrack
12:         if (top != NULL)
13:            ptr[top] = ptr[top].next;          # go to next pointer
14:      }
15:  }
```

Figure 5.3: Depth-first route extraction

(e.g., the leftmost path in Figure 5.1(c)). More generally, we perform a depth-first search through the reduced graph to extract the first *feasible* route, based on the current link-state information and the bandwidth requirement of the new flow. If the extraction process encounters a link that does not (appear to) have enough available bandwidth (denoted by the “X” in Figure 5.1(c)), the algorithm backtracks to the previous node and tries a different parent pointer to find a alternate minimum-cost route (shown by dashed lines).

The depth-first search and the feasibility check effectively “simulate” hop-by-hop signaling, using the most recent link-state information. Note that this operation is purely local at the source, and much less costly than discovering an infeasible link by sending and processing signaling messages in the network. Starting at the destination node, the algorithm builds a stack of the nodes in the route from the source, as shown in Figure 5.3. Each node has a circular list of parent pointers, starting with the `head` parent; a second pointer `ptr` is used to sequence through the list of one or more parents, until `ptr` cycles back to `head`. Each iteration of the `while` loop considers the link between the node at the top of the stack and its current parent (pointed to by the `ptr` pointer and denoted by `new` in line 4). If the link is feasible, `new` is added to the stack and its parents are considered in the next iteration (lines 5–8). Otherwise, if the feasibility check fails, we proceed to the next parent, or backtrack until we find a node whose parent list has not been exhausted (lines 10–13). The algorithm terminates when we reach the source via a feasible path (`top == src`), or when the path stack becomes empty (`top == NULL`) because no feasible path was found. In the former case, the route is read by popping the stack from source to destination.

A potential drawback of this approach is that a subsequent flow request may have to duplicate some of the same backtracking as the preceding request, particularly since the traversal always starts with the parent marked by the `head` pointer. We can avoid this problem by performing some simple pointer manipulations as part of the route extraction process. As we pop each node in the route from the path stack, we may alter the position of its `head` pointer so that a subsequent extraction attempt for the same destination (or any destination along the path) visits a different set of links. Three possibilities are:

- Leave parents in existing order (do nothing).
- Make the selected parent the new head (`head[node] = ptr[node]`). This amounts essentially to "sticky" routing where we stay with the first route that was found to be feasible in the last extraction.
- Round-robin to the next parent (`head[node] = ptr[node].next`). This policy attempts to balance load by alternating the links that carry traffic for a set of destinations.

These alternation policies differ from traditional alternate routing, where the router rotates among a set of cached *paths*. Here, the rotation for one destination node influences the order in which links are visited for other destination nodes along the path. Hence, changing the position of the `head` pointer may actually provide added benefit because the alternate routing is performed on a more global scale.

5.2.4 Route Computation Policy Options

The path precomputation and extraction algorithms provide a useful framework for computing, storing, and selecting from multiple quality-of-service routes. These techniques provide significant latitude in handling individual flow requests, depending on the inaccuracy of the link-state information, connectivity of the underlying network, and tolerance to set-up delay. Table 5.1 summarizes the key policy decisions, starting with four options discussed in the previous subsections. The remaining design decisions concern how and when to recompute routes and initiate signaling for new flows. The simplest approach relies entirely on a periodic, background computation of the shortest-path graph. Periodic recomputation is likely to simplify CPU provisioning, at the expense of blocking flows on routing and set-up failures. When a request arrives, the source extracts a route

Policy	Description
on-demand or precomputed	recompute routes for every request or try first to extract from the existing shortest-path graph
allow multiple routes	limit extraction to just one route instead of allowing backtracking to find alternate routes
feasibility checking	whether to use feasibility checking during route extraction
re-rank multiple routes	policy to arrange `head` pointers in the shortest-path graph during route extraction (none, sticky, round-robin)
periodic recomputations	specification of a background route computation frequency
recompute on routing failure	trigger recomputation when initial route extraction fails
recompute on set-up failure	trigger recomputation on set-up failure
reextract on set-up failure	try to reextract another feasible route (rather than recompute) when signaling fails
prune on recomputation	whether to omit the links that failed the admission test when re-computing after a set-up failure
maximum signaling attempts	limit the number of times each flow request may attempt signal-ing

Table 5.1: Routing and signaling policy options

and initiates signaling. The source blocks the request if the extraction process does not produce a route, and set-up delay is determined almost entirely by signaling delay.

Instead of blocking the flow, the source could trigger recomputation of the shortest-path graph after a failure in route extraction or connection establishment. Recomputing routes with the most recent link-state information would reduce blocking, at the expense of additional processing load and set-up delay. Inaccurate link state or rapidly arriving flow requests may require frequent path recomputation, introducing computation overhead comparable to on-demand routing. To bound set-up delay and limit processing overheads, the source can impose a maximum number of route computations and signaling attempts for each flow. Since the source recomputes the entire shortest-path graph, the overhead is amortized over all destinations and can benefit subsequent requests. To bound the worst-case computational load, the source can impose a minimum time between route recomputations, even in the presence of routing and set-up failures.

When routes are recomputed after a set-up failure, the source can choose to omit the link that failed the admission test. In effect, the set-up failure provides more recent information about the load on the offending link. Temporarily removing this link from consideration is particularly important if the source attempts to compute and signal a new route for the same flow. This new route should avoid using the link that caused the previous failure. In addition, pruning the heavily-loaded link is useful for future flow arrivals, particularly if the shortest-path graph is dedicated to flows with

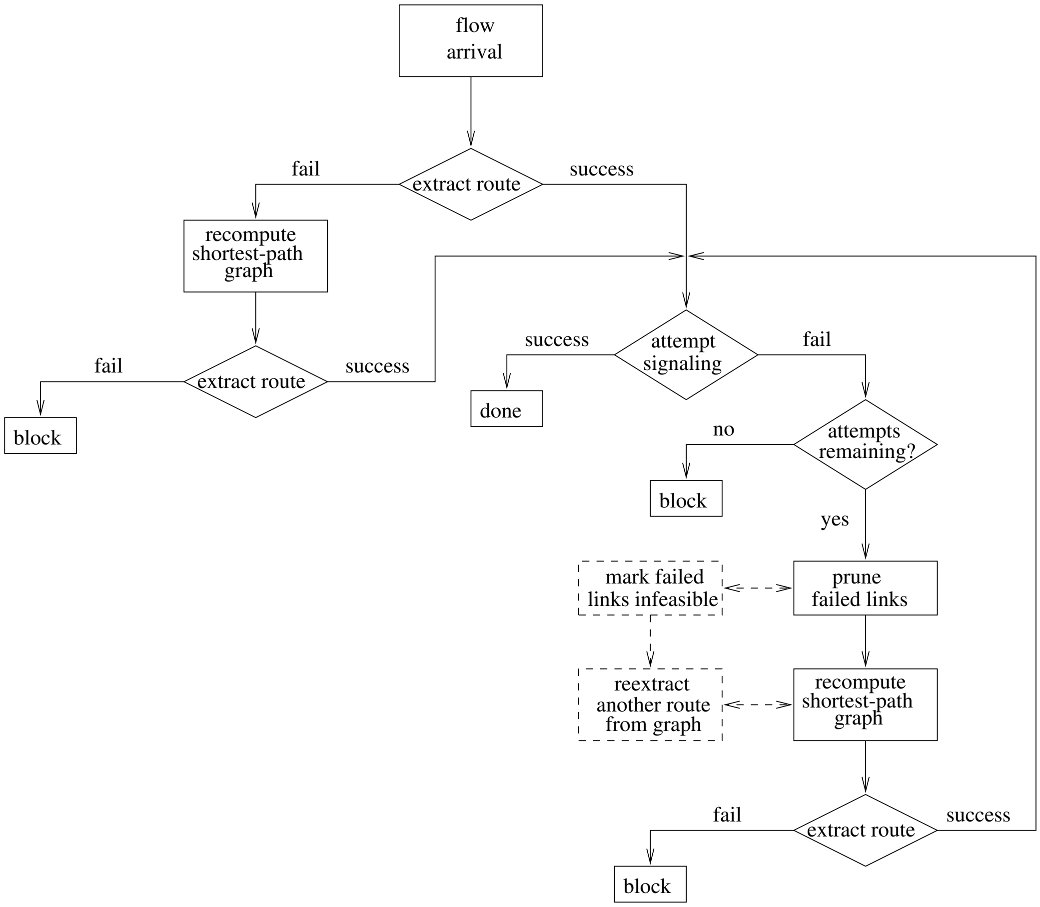


Figure 5.4: Flow request handling procedure

the same (or similar) bandwidth requirements. When flows have more diverse quality-of-service parameters, the source can support a small number of different bandwidth classes (e.g., audio and video), with separate precomputed routes that are tailored to the performance requirements [39, 56]. Employing a different link-cost function and path computation policy for each class enhances the network's ability to route high-bandwidth traffic.

Figure 5.4 illustrates the process of handling a flow request subject to some of the policies shown in Table 5.1. The dashed boxes show the alternate procedure of marking failed links as infeasible and reextracting a new path, rather than recomputing the shortest-path graph.

5.3 Performance Evaluation

In this section, we evaluate the proposed routing algorithm under a range of recomputation periods and link-state update policies. The experiments show that precomputation of the minimum-cost graph, coupled with a feasibility check, approximates the good performance of on-demand routing and the low computational overhead of path caching. The feasibility check is especially effective in reducing the likelihood of expensive set-up failures, even when link-state information is somewhat out-of-date.

5.3.1 Model Extensions for Route Precomputation

To evaluate the cost-performance trade-offs of precomputed routes, we extended `routesim` to support per-source precomputed routes as well as the routing and signaling policy options described in Table 5.1. As before, a route is chosen for each incoming flow based on a throughput requirement (bandwidth b) and the available bandwidth in the network, based on the source's view of link-state information. Then, hop-by-hop signaling reserves the requested bandwidth at each link in the route. A set-up failure occurs if a link in the path cannot support the throughput requirement of the new flow. For the simulations in this section, we assume that a flow blocks after a set-up failure, though we briefly summarize other experiments that allow multiple signaling attempts. Blocking can also occur during path selection if the feasibility check suggests that none of the candidate routes can support the new flow; these routing failures impose less of a burden on the network than set-up failures, which consume resources at downstream nodes.

The source selects a minimum-hop route with the least cost. As discussed in Chapter 3, we use link weights w_i so that a single invocation of the Dijkstra algorithm produces a minimum-cost, shortest path. In a network with diameter D, and link costs in the range $0 < c_i \leq 1$, the link weights $w_i = D + c_i$ ensure that paths with fewer hops always appear cheaper. Such an assignment for w_i also results in a relatively small number of possible path cost estimates, thus reducing the complexity of the computation. To distinguish among paths of the same length, each link has a cost in the set $\{1/C, 2/C, \ldots, C/C\}$. For the experiments in this chapter, a link with reserved capacity u has cost $c = (\lceil u^2 \cdot (C-1) \rceil + 1)/C$. Our experiments with link-cost functions in Chapter 4 showed that an exponent of 2 biases away from routes with heavily-loaded links, without being too sensitive to small changes in link-state information. For simplicity we assume that links are bidirectional, with unit capacity in each direction. We evaluate the routing algorithms on a "well-known" core topology (an early version of the MCI Internet backbone) and a uniformly connected 125-node 5-ary 3-cube topology (with 5 nodes along each of 3 dimensions). The relationship between size and connectivity in the 5-ary 3-cube is similar to existing commercial networks, and allows us to study the potential benefits of having multiple minimum-hop routes between pairs of nodes. The MCI

Parameter	MCI Internet	5-ary 3-cube
offered load	$\rho = 0.65$	$\rho = 0.85$
bandwidth	$b = (0, 4\%]$	$b = (0, 6\%]$
arrival rate	$\lambda = 1$	$\lambda = 1$
flow duration	$\ell = 46.8$	$\ell = 46.8$

Table 5.2: Simulation invariants

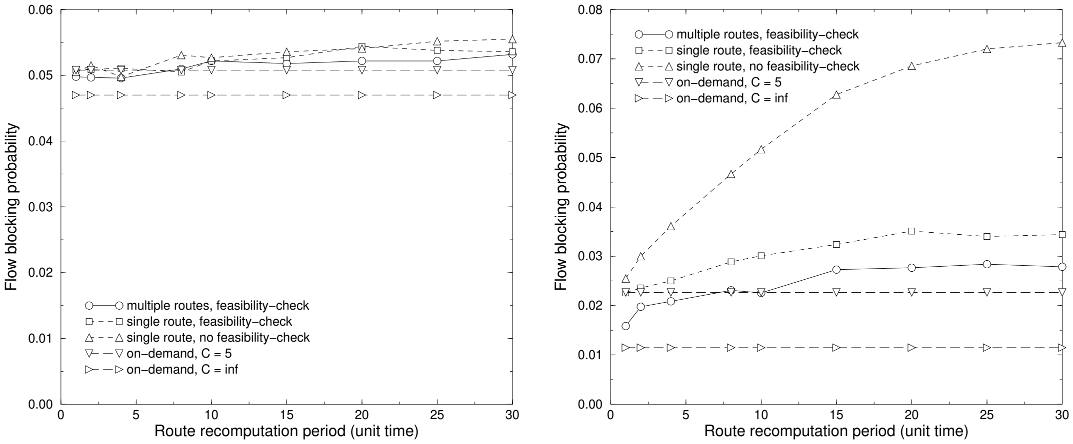
network, on the other hand, is very small and loosely connected. Pertinent characteristics of both topologies are listed in Chapter 3 in Table 3.1.

Flow requests arrive at the same rate at all nodes and destinations are selected uniformly. Durations have mean ℓ and follow a Pareto distribution with shape parameter 2.5 to capture the long-tailed nature of flow durations. Flow interarrival times are exponentially distributed with mean $1/\lambda$, and requested bandwidths are uniformly distributed with equal spread about a mean size $\bar{b}$. The experiments evaluate flows with mean bandwidth requirements in the range of 2–3% of link capacity. Chapter 4 considers a wider range of bandwidth requests in the context of QoS routing with inaccurate information. The expression for offered load remains unchanged, i.e. $\rho = \lambda N \ell \bar{b} \bar{h} / L$. Table 5.2 summarizes the simulation parameters for the two network topologies.

5.3.2 Accurate Link-State Information

The initial simulation experiments compare the performance and overhead of the routing algorithms under accurate link-state information, as shown in Figures 5.5 and 5.6. We vary the background period for path precomputation, and also allow a router to recompute its shortest-path graph when the route extraction does not produce a candidate route (due to failed feasibility checks), and after a set-up failure. Since we allow only one signaling attempt for each flow, recomputing after a set-up failure only benefits future arrivals. The single-route algorithms use a precomputed shortest-path graph with one route, as in Figure 5.1(a). The multiple-route approach offers greater flexibility by allowing the source to perform feasibility checks on several paths in the graph, as in Figure 5.1(b). We also consider two on-demand algorithms that compute routes on a per-request basis. One has a link-cost discretization of $C = 5$, identical to the precomputed routing algorithms, and the other has discretization limited only by machine precision (referred to as $C = \infty$).

Figure 5.5 plots the flow blocking probability as we increase the recomputation period to more



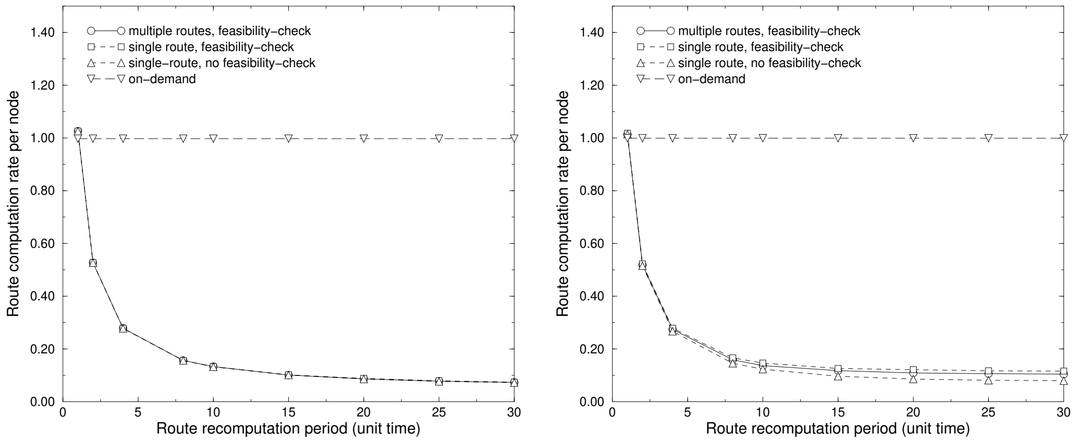
(a) MCI Internet topology

(b) 5-ary 3-cube

Figure 5.5: Performance with accurate link state

than 30 times the flow interarrival time. The precomputation algorithms perform well, relative to the more expensive on-demand schemes. Feasibility checking substantially improves performance over traditional path-caching techniques, allowing the use of much larger computation periods. In addition, under accurate link-state information, feasibility checking completely avoids set-up failures by "simulating" the effects of signaling during the route extraction process. This is particularly helpful in the 5-ary 3-cube network, since the richly-connected topology frequently has an alternate shortest-path route available when the first choice is infeasible. On-demand routing with $C = 5$ suffers somewhat by its inability to distinguish the "best" route, compared to the $C = \infty$ case, but this effect diminishes under link-state inaccuracy. Other experiments illustrate that, without a feasibility check, precomputed routing can only achieve these performance gains by allowing multiple signaling attempts for a flow, at the expense of longer set-up delays and higher processing requirements.

The ability to precompute multiple candidate routes does not substantially reduce the blocking probability in Figure 5.5(a), since the sparsely-connected MCI topology typically does not have multiple shortest-path routes, let alone multiple routes of equal cost. In contrast, the 5-ary 3-cube experiment in Figure 5.5(b) shows a more substantial performance benefit. These benefits will apply to real networks as they grow larger and more densely connected. We also expect a more significant gain under nonuniform traffic loads, since alternate routes would enable flows to circumvent regions of congestion. This is especially true during transient fluctuations in network load, caused by bursty flow arrivals or rerouting after a link failure. In these cases, the precomputation of multiple routes



(a) MCI Internet topology (b) 5-ary 3-cube

Figure 5.6: Overhead with accurate link state

allows the source to survive longer intervals of time without computing a new shortest-path graph. The algorithm with multiple precomputed routes, coupled with the feasibility test, performs almost as well as on-demand routing with the same number of cost levels. However, using $C = \infty$ offers a noticeable performance advantage in Figure 5.5, since the accurate link-state information enables the fine-grain link-cost function to locate the "best" routes.

The lower blocking probabilities for on-demand routing come at the expense of a significant cost in processing load, as shown in Figure 5.6. In both the MCI and 5-ary 3-cube topologies, the route computation frequency is lowered by a factor of 10 below that of on-demand routing. In addition, the precomputation schemes have much lower complexity for each route computation, relative to the $C = \infty$ on-demand algorithm. Comparing the different precomputation algorithms, the processing load is dominated by the background recomputation of the shortest-path graph, though the feasibility test introduces slightly more triggered recomputations. As the period increases, the graphs flatten since triggered recomputations become increasingly common for all of the algorithms. Although disabling these triggered recomputations results in more predictable processing overheads, additional simulation experiments indicate that this substantially increases the blocking probability. For example, Figure 5.7 compares the performance of using strictly periodic computation to the case when triggered recomputation is allowed for accurate link-state information. Even with no feasibility checking enabled, allowing occasional triggered recomputation (due to a set-up failure) is superior to relying only on periodic recomputation.

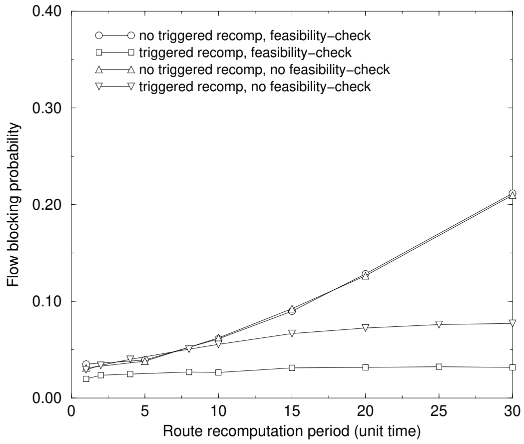
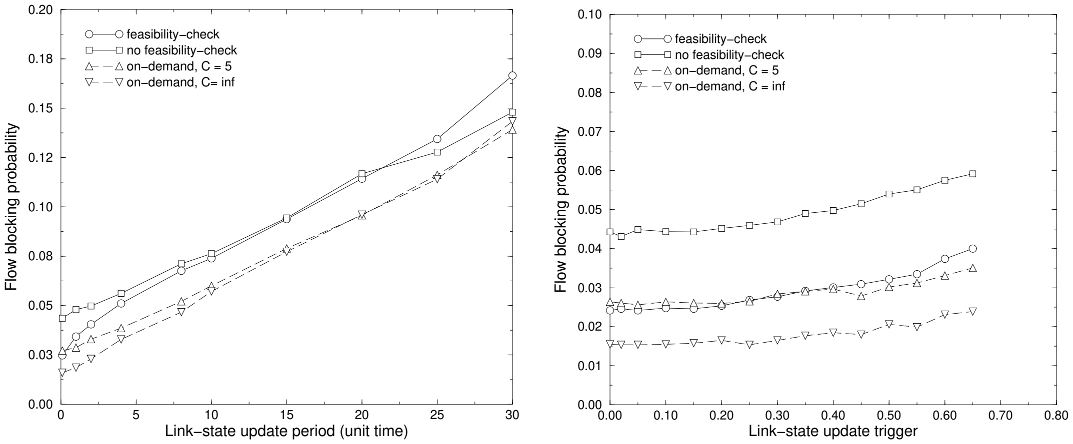


Figure 5.7: Performance with and without triggered recomputation

5.3.3 Inaccurate Link-State Information

While the previous experiments assume that the source has accurate knowledge of network load, Figure 5.8 considers the effects of stale link-state information, for both periodic and triggered link-state updates, with a background recomputation period of 5 time units. As staleness increases, the $C = \infty$ and $C = 5$ curves gradually converge, since fine-grain link costs offer progressively less meaningful information about network load. However, large link-state update periods also degrade the effectiveness of the feasibility check in our precomputed routing scheme, as shown in Figure 5.8(a). Periodic updates can lead the source router to mistakenly identify infeasible links as feasible, and feasible links as infeasible. When link-state information is extremely stale (e.g., an update period that is 20 times larger than the mean flow interarrival time), signaling blindly without a feasibility check offers better performance. In fact, under such large update periods, none of the QoS-routing algorithms perform well; under the same network and traffic configuration, even *static* shortest-path routing (not shown) can achieve a blocking probability of 16%. Experiments with the MCI topology configuration do not show as much benefit from feasibility checking, due to the smaller number of shortest-path routes, though the feasibility test does reduce the likelihood of set-up failures.

Despite the effects of large update periods, we find that the feasibility check still offers significant advantages under more reasonable levels of link-state staleness. This is particularly true when link-state updates are triggered by a change in available link capacity, as shown in Figure 5.8(b). Recall that, a trigger of 0.2 implies that a link-state update message is generated whenever available capacity has changed by 20% since the last update, due to the establishment and termination of



(a) Periodic link-state updates (b) Triggered updates

Figure 5.8: Performance with stale link state

flows. Triggered link-state updates, like triggered path recomputations, generate new routing information during critical fluctuations in network load. Under triggered link-state updates, the routers typically have accurate information about heavily-loaded links, even for large trigger values. Consequently, the flow blocking probability is fairly insensitive to the trigger, across all of the routing algorithms. In addition, feasibility checking remains very effective across the range of update triggers and competitive with on-demand routing (with C = 5), in contrast to the results for update periods in Figure 5.8(a). We find also that using triggered updates allows feasibility checking to reduce blocking even in the MCI topology, though the difference between feasibility and no feasibility checking is less significant than in the 5-ary 3-cube.

Although the blocking probability remains nearly constant as a function of the link-state trigger, small triggers result in fewer set-up failures, as long as the source performs a feasibility test. In contrast, set-up failures account for *all* flow blocking when routing does not involve feasibility checking. As the trigger grows, the feasibility test sometimes mistakenly concludes that a heavily-loaded link is feasible, and the flow blocks in signaling instead of experiencing a routing failure. Still, even for a 50% trigger, set-up failures only contribute 30% of the blocking under feasibility checking. Also, despite the fact that periodic updates cause significant staleness and worsen overall flow blocking, feasibility checking still manages to avoid signaling for flows that ultimately block about $30-40\%$ of the time. The ability of feasibility checking to reduce blocking inside the network is an important benefit since set-up failures consume processing resources and delay the establishment of other flows. Routing failures, on the other hand, are purely local and do not consume additional resources beyond the processing capacity at the source router.

Examining the blocking relative to hopcount shows that feasibility checking at the source also helps in routing flows between distant source-destination pairs. Since bandwidth must be reserved on more links, signaling blindly without checking recent link state has a lower probability of finding a successful route. Other experiments show that when a hold-down timer is used to impose a minimum time between consecutive update messages, the blocking probability rises slightly for all algorithms, though the benefit of feasibility checking remains. The hold-down timer is useful, however, in reducing link-state update overhead, especially when using small trigger levels. For example, with a hold-down timer equal to the mean flow interarrival time, the update generation rate can be reduced by over 35% for triggers in the range $0 - 0.1$. Moreover, even with a hold-down timer, coarser triggers do not degrade performance. The combination of triggered updates and a small hold-down timer, coupled with feasibility checks and multiple precomputed routes, offer an efficient and effective approach to quality-of-service routing in large networks.

5.4 Related Work in QoS Route Precomputation

Previous work on route precomputation has focused on path caching policies, performance evaluation, and algorithmic issues. Our work complements these studies by emphasizing lower-level mechanisms and introducing an efficient framework for precomputing and storing QoS routes, which applies to a variety of routing and signaling policies.

Research on path caching has focused on storing routes in a separate data structure and considering different policies for updating and replacing precomputed routes. The work in [68] introduces a policy that invalidates cache entries based on the number of link-state updates that have arrived for links in the precomputed paths. The proposed algorithms also check the current link-state when selecting a path from the cache and allow recomputation when the cached paths are not suitable, similar to the feasibility check in this paper. This earlier work does not, however, address route computation or path extraction mechanisms. Another study proposes a set of route precomputation policies that optimize various criteria, such as flow blocking and set-up latency [47]. The algorithms try to locate routes that satisfy several QoS requirements through an iterative search of precomputed paths (optimized for hopcount) followed, if necessary, by several on-demand calculations that optimize different additive QoS parameters, one at a time.

Other research has focused on detailed performance evaluation to compare precomputed and on-demand routing under different network, traffic, and staleness configurations. The work in [6] evaluates the performance and processing overhead of a specific path precomputation algorithm. The study adopts the Bellman-Ford-based algorithm from [39] and evaluates a purely periodic precomputation scheme under a variety of traffic and network configurations. The study presents a detailed cost model of route computation to compare the overhead of on-demand and precomputed strategies. As part of a broader study of QoS routing, the work in [56] evaluates a class-based scheme that precomputes a set of routes for different bandwidth classes. The evaluation compares the performance of several algorithms for class-based path computation to on-demand computation. These two studies do not propose any particular strategy for path storage or extraction but instead focus on performance trends.

The remaining studies consider different ways to precompute paths for multiple destination nodes and flow QoS requirements. Some of this work, namely that in [39], was discussed in Chapter 2. Another algorithm, introduced in [53], precomputes a set of extremal routes to all destinations such that no other route has both higher bottleneck bandwidth *and* smaller hopcount. The Bellman-Ford-based algorithm in [39] uses a similar optimization criterion to construct a next-hop routing table with multiple routing entries for each destination. The emphasis of these proposals is on algorithmic issues, such as reducing complexity. In addition to a focus on computational overheads, we also consider efficient ways to store precomputed paths and apply the most recent link-state information.

5.5 Summary

In this chapter, we introduced efficient mechanisms for precomputing quality-of-service routes, while still applying the most recent link-state information at flow arrival. Route computation employs a small extension to Dijkstra's algorithm, coupled with discretized link costs, to generate a shortest-path graph with one or more routes to each destination. Upon flow arrival, route extraction involves a depth-first search with a feasibility test, which returns the first route in the common case. Simulation experiments show that the algorithm offers substantial reductions in computational load with only a small degradation in performance, compared to more expensive on-demand algorithms. Our precomputation scheme continues to perform well under stale link-state information, particularly under triggered link-state updates. In addition to having lower blocking probabilities than traditional path-caching schemes, the feasibility test reduces network overhead by decreasing the frequency of signaling failures.

By introducing new algorithms for path precomputation, this chapter addresses the problem of reducing computational overheads of dynamic routing. These mechanisms must be coupled with other techniques for reducing the overheads of frequent link-state update distribution to make dynamic routing practical. The next chapter, therefore, considers a new approach for load-sensitive routing that dramatically reduces computational *and* bandwidth overheads, and can make use of the precomputation algorithms described in this chapter.

As part of future work, we are interested in investigating enhancements to our path precomputation and extraction algorithms. To increase the likelihood of having multiple candidate routes to each destination, we are considering heuristics for generating *near*-minimum-cost alternate routes, while still storing the precomputed routes in a compact graph representation. In addition, we are pursuing other ways to test the suitability of routes during the extraction process. For example, instead of checking *link* feasibility, the source could use the most recent link-state information to compute new *path* costs as part of the depth-first search for a route; whenever the accumulated path cost exceeds a certain threshold, the algorithm can backtrack to consider other precomputed routes. This approach is well-suited to precomputing paths that balance network load without requiring information about the traffic or QoS parameters of individual flows.

CHAPTER 6

DYNAMIC ROUTING OF LONG-LIVED IP FLOWS

6.1 Introduction

The previous chapters considered dynamic routing in the context of providing explicit QoS guarantees for individual application flows. Recently, however, dynamic routing is being recast as a tool for network providers to manage the increasing variability and fluctuation in backbone traffic. Traffic engineering of large IP backbone networks has become a critical issue in recent years, due to the unparalleled growth of the Internet and the increasing demand for predictable communication performance. Currently, network providers must resort to coarse timescale measurements to detect network performance problems, or may even depend on complaints from their customers to realize that the network requires reconfiguration. Detection may be followed by a lengthy diagnosis process to discover what caused the shift in traffic. Finally, providers must manually adjust the network configuration, typically redirecting traffic by altering the underlying routes.

These traffic engineering challenges have spurred renewed interest in dynamic routing as a network management tool, rather than as a method for providing QoS guarantees. Dynamic routing balances network load and improves application performance by diverting traffic from already congested links. As illustrated in Chapter 4, however, load-sensitive routing can lead to route flapping and excessive control traffic overhead. This chapter introduces a new routing scheme that maintains the benefits of dynamic routing by controlling route flapping, and thus improves its stability and efficiency.

6.2 Flow-based Dynamic Routing

As described in Chapter 2, early attempts in the ARPANET to route based on dynamic link metrics resulted in dramatic fluctuations in link load over time. Improvements in the definition of the link metrics reduced the likelihood of oscillations [49], but designing stable schemes for load-sensitive routing is fundamentally difficult in packet-based networks like the Internet [88]. With the evolution toward providing QoS in IP networks, recent research focused on load-sensitive routing of flows or connections, instead of individual packets, as in the model presented in Chapter 3. For example, a flow could correspond to a single TCP or UDP session, all IP traffic between a particular source-destination pair, or even coarser levels of aggregation.

The hope was that dynamic routing of flows should be more stable than selecting paths at the packet level, since the load on each link should fluctuate more slowly, relative to the time between updates of link-state information. Also, defining network load in terms of reserved bandwidth and buffer space, rather than measured utilization, should enhance stability. Most TCP/UDP transfers consist of just a handful of packets, however, and Chapters 4 and 5 illustrated that load-sensitive routing of short flows requires frequent propagation of link-state metrics and recomputation of routes to avoid the same instability problems that arise in dynamic routing at the packet level. We address this problem by proposing and evaluating a hybrid routing scheme that exploits the variability of IP flow durations to avoid the undesirable effects of traditional approaches to dynamic routing.

While most Internet flows are short-lived, the majority of the packets and bytes belong to long-lived flows, and this property persists across several levels of aggregation [20, 24, 28, 82]. Although this inherent variability of Internet traffic sometimes complicates the provisioning of network bandwidth and buffer resources, heavy-tailed flow-size distributions can be exploited to reduce the overheads of certain control mechanisms. Observations of heavy-tailed lifetime distributions of UNIX processes have been similarly exploited in the context of processor load balancing, where migrating long-lived jobs can significantly reduce migration overhead [42]. Most notably in the networking context, variability in flow duration has been the basis of several techniques that reduce router forwarding overheads by establishing hardware switching paths for long-lived flows [8, 48, 66]. These schemes classify arriving packets into flows and apply a trigger (e.g., arrival of some number of packets within a certain time interval) to detect long-lived flows. Then, the router dynamically establishes a shortcut connection that carries the remaining packets of the flow. The shortcut terminates if no packets arrive during a predetermined timeout period (e.g., 60 seconds). Several measurement-based studies have demonstrated that it is possible to limit the set-up rate and the number of simultaneous shortcut connections, while forwarding a large fraction of packets on shortcuts [14, 28, 55, 66, 91].

This chapter builds on the above observations to study the implications of the variability in flow durations on the stability of load-sensitive routing. In contrast to previous work on flow switching and the work described in previous chapters, the choice to separate long-lived and short-lived flows is not necessarily motivated by the desire to forward packets in hardware, or the QoS requirements (if any) of the individual flows. We focus on dynamic routing as a traffic engineering technique that reacts to fluctuations in network load, rather than as a way to provide explicit performance guarantees. The key elements of our approach are:

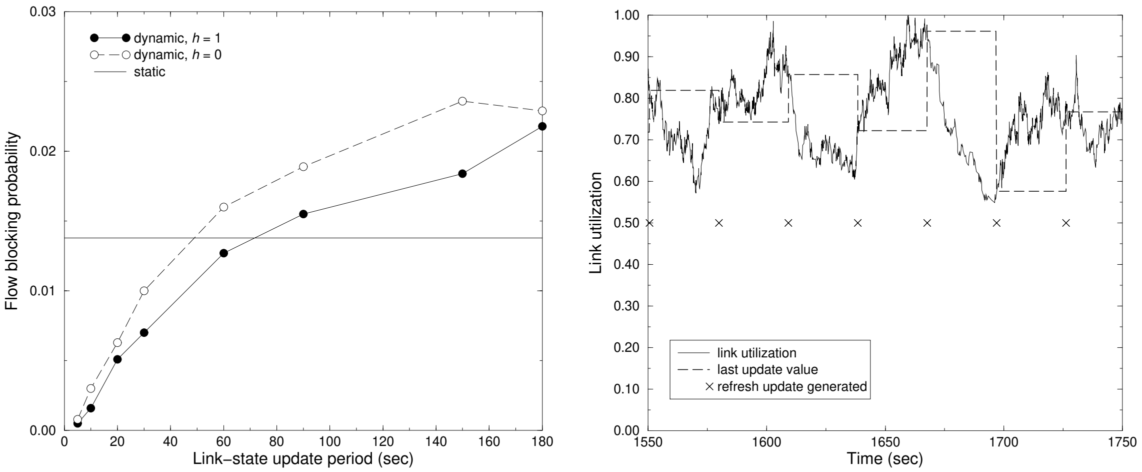
- **Load-sensitive routing of long-lived flows:** We propose that backbone networks should perform dynamic routing of long-lived flows, while forwarding short-lived flows on preprovisioned static paths. Our approach exploits flow-classification hardware at the network edge and techniques for flow pinning, as well as basic insights from earlier work on QoS routing.

- **Relating traffic timescale to routing stability:** Separating short-lived and long-lived IP flows dramatically improves the stability of dynamic routing. Our hybrid scheme reacts to fluctuations in network load without introducing route flapping, by relating the detection of long-lived flows to the timescale of link-state update messages.

- **Network provisioning rules:** We propose simple and robust rules for allocating network resources for short-lived and long-lived flows, and techniques for sharing excess link capacity between the two traffic classes. The provisioning rules are tailored to measurements of the distribution of IP flow sizes, and the triggering policy for detecting long-lived flows.

Our approach complements and extends recent work on Multi-Protocol Over ATM (MPOA) [8], which triggers the creation of IP shortcut connections across an underlying ATM network. In contrast to recent studies that evaluate the overheads of a single-link MPOA system [14, 91], we concentrate on the dynamics of load-sensitive routing over an entire network. Though our work is applicable to MPOA, we focus more broadly on IP networking rather than techniques for carrying IP traffic over ATM networks.

In the following sections, we outline our approach for separating short-lived and long-lived



(a) Flow blocking vs. update period (b) Link state vs. time for 30-sec period

Figure 6.1: QoS routing under stale link-state information

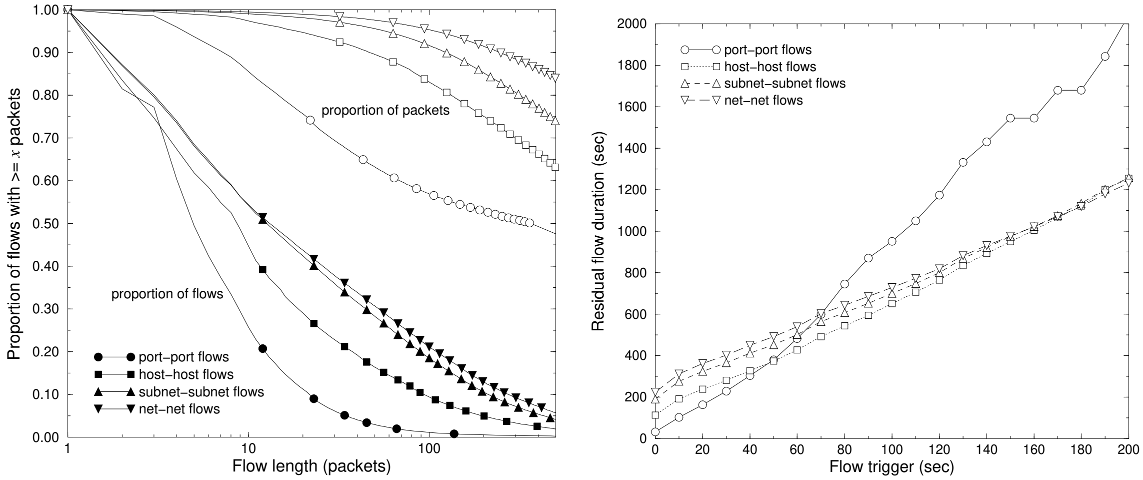
flows, and provide details on the policies for flow detection, path selection, and network provisioning. The benefits of the hybrid approach are illustrated in through detailed simulations based on packet traces from a large ISP network.

6.3 Stable Load-Sensitive Routing

In this section, we describe how to dynamically route long-lived flows, while forwarding short-lived flows on static, preprovisioned paths. We argue that stability and efficiency can be achieved by tying the frequency of link-state update messages to the timescale of the long-lived flows.

6.3.1 Stability Challenges in Load-Sensitive Routing

Through experiments in Chapter 4, we observed that QoS routing based on out-of-date information severely degrades performance. Figure 6.1(a) plots the blocking probability for QoS routing across a range of link-state update periods, similar to experiments in Section 4.2, but using flow durations sampled from our ISP packet trace (with traffic aggregated to the port level). That is, rather than using synthetic duration distributions and unitless time values as described in Chapter 3, we use real flow durations and time units. The graph shows minimal ($h = 0$) and non-minimal ($h = 1$) QoS routing, as well as static minimal routing. The flow-size distributions are shown in Figure 6.2(a), and the details of the experimental set-up are described later in Section 6.5. The graph reinforces the observation that load-sensitive routing is effective under accurate link-state information, and



(a) Proportion of packets in long flows (b) Average residual lifetime of long flows

Figure 6.2: Heavy-tailed nature of IP flows

degrades in performance for reasonable update periods in the tens of seconds.

Increasing the update period or trigger threshold reduces the link-state update frequency but results in out-of-date information which can cause substantial route flapping and induce a router to select a suboptimal or infeasible path. Figure 6.1(b) illustrates the effects of route flapping, again with flow durations drawn from the packet trace.

It is possible to reduce flapping by selecting amongst a set of several paths, or by using coarse-grained link metrics to increase the likelihood of "ties" among similar paths. Such techniques avoid the problem of targeting the one "best" path, but merely extend the range of link-state update periods under which conventional load-sensitive routing performs well. Ultimately, flapping still arises when the timescale of the arriving and departing traffic is small relative to the link-state updates.

6.3.2 Routing Short and Long Flows

To address these efficiency and stability challenges, we propose a hybrid routing scheme that performs load-sensitive routing of long-lived traffic flows, while forwarding short-lived flows on static preprovisioned paths. Focusing on the long-lived flows reduces the overheads of load-sensitive routing in three critical ways:

- **Fewer signaling operations**: Limiting load-sensitive routing to long-lived traffic substantially reduces the number of signaling operations for pinning routes, while still carrying the majority of packets and bytes on dynamically-selected paths, as shown in Figure 6.2(a). The solid lines in the figure show the proportion of flows that have x or more packets for several levels of flow aggregation, plotted on a logarithmic scale. The dashed lines show the proportion of packets on the corresponding flows. For example, for port-to-port flows, only about 20% of the flows have more than 10 packets but these flows carry 85% of the total traffic.

- **Fewer link-state update messages**: Dynamic routing of long-lived flows reduces the frequency of link-state update messages, both by reducing the number of flows that are dynamically routed, and by dramatically increasing the average flow duration, as shown in Figure 6.2(b). The right graph shows the average residual lifetime of flows after applying a x-second flow trigger for several levels of aggregation.

- **Fewer route computations**: The slower changes in link-state information permit the routers to execute the path-selection algorithm less often without significantly degrading the quality of the routes. The routers can exploit efficient techniques for path precomputation rather than computing paths at flow arrival.

In addition, recent measurement studies suggest that long-lived flows have a less bursty arrival process than short-lived flows [28]. Hence, focusing on long-lived flows should reduce the variability in the protocol and computational demands of dynamic routing, and lower the likelihood that a large number of flows route to the same links before new link-state metrics are available.

Exploiting these potential gains requires careful consideration of how long-lived traffic interacts with the many short-lived flows in the network. The interaction between short-lived and long-lived flows depends on how link-state metrics are defined in our hybrid routing scheme. This motivates three key design decisions, discussed in more depth in the next section:

- **Defining link state in terms of allocated resources:** Earlier approaches to dynamic routing in packet networks operated on *measured* quantities, such as average utilization, queue length, or delay. These measured quantities can fluctuate on a fairly small timescale, due to the variability of Internet traffic, and are very sensitive to the selection of the estimation interval. In contrast, we define link state in terms of *allocated* resources on each link, which results in a more stable quantity that changes on the timescale of flow arrival or departure. Separating routing for long-lived traffic may in fact improve the stability of measured quantities like link utilization and queue length, allowing the use of such metrics in path selection. This would be an interesting avenue for future research.

- **Distributing link state for dynamically-routed traffic:** The link state could conceivably reflect the resources consumed by both the short- and long-lived flows. This model is appropriate in an integrated services network that initiates signaling for all flows. However, explicitly allocating resources for each of the many short-lived flows would introduce significant overheads. In addition, the burstiness in the arrivals of short-lived flows would increase the variability of the link-state metric. Instead, we define link state in terms of the resources allocated to the long-lived flows.

- **Allocating resources for statically-routed traffic:** In basing link state only on the dynamically-routed traffic, our hybrid routing protocol runs the risk of directing long-lived flows to links that already carry a significant amount of statically-routed, short-lived traffic. To prevent this situation, we do not permit dynamically-routed flows to be allocated the entire capacity of a link. This is similar in spirit to earlier work on *trunk reservation* [36]. The proportion of link resources that can be allocated for short-lived flows can be tailored to the proportion of traffic carried by these flows.

For example, suppose a link has capacity C. Then, our hybrid routing protocol allows long-lived flows to reserve some portion $c_\ell \leq C$ of these resources. At any time t, these dynamically-routed flows have been allocated some portion $u_\ell(t) \leq c_\ell$ of these resources.

Despite this logical partitioning of network resources, the short-lived flows should be able to consume excess capacity when the long-lived partition is underutilized. This approach is well-suited to best-effort and adaptive Internet applications, which can exploit additional bandwidth when it is available, and reduce the sending rate when the resources are constrained. The allocation of bandwidth c_ℓ exists only to control routing, and need not dictate the link-scheduling policies. In fact, the short-lived and long-lived flows could each select from a variety of link-scheduling and buffer-management techniques, such as class-based queuing and weighted Random Early Detection, to differentiate between flows with different performance requirements. For example, a router could direct all incoming traffic receiving assured service to a single queue, irrespective of whether a packet belongs to a short-lived or long-lived flow. The router need not provide per-flow scheduling or buffer management for long-lived flows, though a particular implementation could employ per-flow mechanisms to further improve performance. But, these router mechanisms are not necessary to exploit the traffic engineering benefits of separating long-lived and short-lived flows.

Parameter	Definition
Flow trigger	Minimum bytes, packets or seconds before dynamic routing (e.g., 10 packets)
Flow timeout	Maximum idle period before terminating a flow (e.g., 60 seconds)
Flow aggregation	Level of address aggregation (e.g., port, host, subnet, or net)
Update period	Maximum time between link-state updates (e.g., 90 seconds)
Hold-down timer	Minimum time between link-state updates (e.g., 5 seconds)
Link-state trigger	Minimum proportional change in link state (e.g., 30% change)
Path threshold	Minimum number of hops beyond a shortest path (e.g., 2 hops)
Capacity partition	Proportion of link capacity allocated to long-lived flows (e.g., 55%)

Table 6.1: Key parameters of hybrid routing model

6.4 Routing and Provisioning

This section describes the details of how to separate short-lived and long-lived traffic, including flow detection, path selection, and network provisioning. Like most routing protocols, our approach has a number of tunable parameters that affect the network dynamics. These parameters are summarized in Table 6.1, which serves as the basis of the simulation study in Section 6.6.

6.4.1 Detecting Long Flows and Pinning Routes

The hybrid routing scheme requires an effective way for the network to classify flows, and to initiate selection of a dynamic route for the long-lived traffic. Routers at the edge of the network can employ flow classification hardware [51, 80] to associate each packet with a flow, based on bits in the IP and TCP/UDP headers. Depending on the flow definition, the classifier could group packets from the same TCP connection or, more broadly, from the same host end-points or subnets. The hardware can also keep track of the number of bytes or packets that have arrived on each flow, or the length of time that the flow has been active. By default, the router forwards arriving packets on the path(s) selected by the static intradomain routing policy. For example, in OSPF, the router would forward the incoming packet to an outgoing link along a shortest-path route, based on static link weights. Then, once the accumulated size or duration of the flow has exceeded some threshold (in terms of bytes, packets, or seconds), the router selects a dynamic route for the remaining packets in the flow. The flow classifier continues to track the arriving packets, and signals the termination of the dynamic route after the timeout period expires. Figure 6.3 summarizes the flow detection and routing procedure.

The dynamic route could be established by creating a label-switched path in MPLS, which pop-

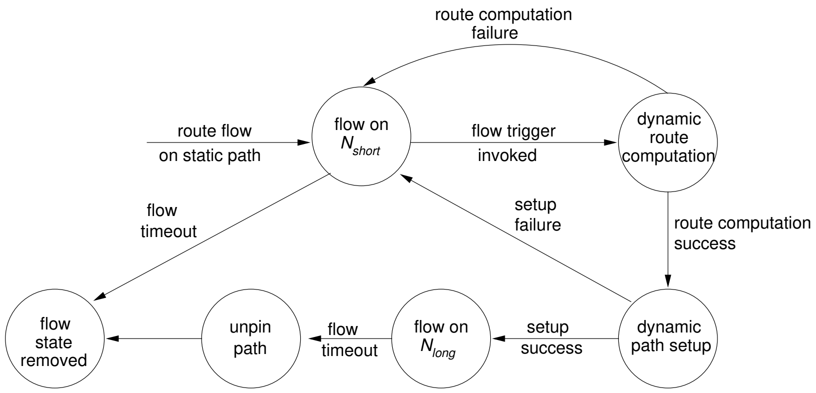


Figure 6.3: Flow state machine

ulates the forwarding tables in the routers along the flow's new path. In this scenario, the edge router selects an explicit route and signals the path through the network, as in a traditional application of MPLS. If the network consists of ATM switches, the dynamic route would involve path selection and connection signaling similar in spirit to MPOA. The selected route could be cached or placed in a routing table, to ensure that future packets are forwarded along the selected route, until the flow timeout is reached. The dynamic path is selected based on the router's current view u_l' of the load from the dynamically-routed, long-lived traffic on each link, as well as the resources b requested for the flow. Since the router may have out-of-date link-state information, the value of u_l' may differ from the actual utilization u_l. The resource requirement b for each flow could be included in the flow classifier at the edge router (e.g., the parameters of the flow conditioner under differentiated services). Alternatively, the value of b could be implicitly associated with other parameters in the classifier, such as the port numbers or IP addresses (e.g., dedicating a fixed bandwidth to Web transfers, or to the set of users with IP addresses in the range assigned to a modem bank).

Dynamic routing of long-lived flows draws on metrics u_l that represent the reserved resources on each link. Link-state advertisements can be flooded throughout the network, as in QOSPF and PNNI, or may be piggybacked on the messages used for the default intradomain routing protocol. As with the work in earlier chapters, our study focuses on link bandwidth as the primary network resource. Although network load may be characterized by several other dynamic parameters, including delay and loss, initial deployments of load-sensitive routing are likely to focus on a single simple metric to reduce algorithmic complexity. In addition, bandwidth is an additive metric, which simplifies the computation of the link-state metric and ensures that the core routers need only store aggregate information about the resource requirements of the set of flows on each link (e.g., upon arrival of a long-lived flow, the router could update $u_l = u_l + b$). The value of b could represent the peak, average, or effective bandwidth of the flow. Since any of the short-lived or long-lived flows can consume excess link capacity, inaccuracies in estimating the resource requirements of any particular flow need not waste network bandwidth.

6.4.2 Selecting Paths for Long-Lived Flows

When dynamically routing a long-lived flow, the edge router can prune links that do not appear to satisfy the bandwidth requirement of the flow (i.e., $u'_l + b > c_l$). After selecting the path, the flow undergoes hop-by-hop signaling to reserve the bandwidth on each link. In the meantime, the flow's packets continue to travel on the default static path. Upon receiving a signaling message, a core router tests that the link can actually support the additional traffic (i.e., $u_l + b \leq c_l$) and updates the link state upon accepting the flow; these resources are released upon flow termination. If any of the links in the path is unable to support the additional traffic (i.e., $u_l + b > c_l$), the flow may be blocked and forced to remain on the static path. Note that, in contrast to conventional QoS-routing schemes, we do not reject a blocked flow, but instead continue to forward the flow's packets on the static, preprovisioned path. Another dynamic routing operation may be performed after the flow trigger is reached again. As an alternate policy, the flow could be accepted on dynamically-routed path, even though the resources are temporarily over-allocated.

With effective provisioning of the resource c_ℓ for long-lived flows, encountering an over-constrained link should be an unlikely event. Once the flow has been accepted, the remaining packets are forwarded along the new path. Still, only a subset of the long-lived flows are active at any moment, and others may not consume their entire allocated bandwidth across time. In particular, no bandwidth is consumed during the flow timeout period, which could be as large as 60 seconds. The presence of inactive flows can be handled by allocating a smaller amount of bandwidth for each individual flow. For example, suppose that measurement of the flow-size distribution $f(x)$ shows that long-lived flows have an average residual lifetime of ℓ_l seconds. Then, each flow could be allocated a bandwidth b that is a proportion $\ell_l/(\ell_l + 60)$ of its estimated resource requirement. In addition, the short-lived flows can capitalize on transient periods of excess capacity ($u_\ell < c_\ell$), making our scheme robust to inaccuracies in estimating the aggregate bandwidth requirements of the long-lived flows.

A variety of load-sensitive routing algorithms could be used for path selection for long-lived flows. Drawing on the insights of previous studies of load-sensitive routing [18, 36], the dynamic algorithm should favor short paths to avoid consuming extra resources in the network. Long routes make it difficult to select feasible paths for subsequent long-lived flows, and also consume excess bandwidth that would otherwise be available to the short-lived traffic. Out-of-date link-state information exacerbates this problem, since stale link metrics may cause a flow to follow a non-minimal path even when a feasible shortest path exists. To further benefit the short-lived flows, the long-lived flows are routed on paths that have the most unreserved capacity ($c_\ell - u_\ell$). In other words, amongst a set of routes of equal length, the algorithm selects the *widest* path. If the widest, shortest path cannot support the bandwidth of the new flow, the algorithm considers routes up to some $h \geq 0$ hops longer than the shortest path. By reducing the effects of stale link-state information, our proposed routing scheme should permit the use of nonminimal paths (e.g., $h = 1$), unlike an approach that performs dynamic routing of all flows. Path selection can be implemented efficiently using the Bellman-Ford algorithm [39], or a variant of the Dijkstra shortest-path algorithm.

6.4.3 Trunk Reservation for Short-Lived Flows

The effectiveness of the hybrid routing policy depends on how the link resources are allocated between the short-lived and long-lived flows. In the simplest case, the network could allow dynamically-routed traffic to be allocated the entire capacity C on a link. Since path-selection favors paths with more available bandwidth, the long-lived flows would not typically consume an excessive amount of resources on any one link. However, if a link carries a large amount of short-lived traffic, dynamically routing additional traffic to this link would increase the congestion, particularly under non-uniform traffic. To avoid unexpected congestion, the routing protocol should be aware of the expected load that the statically-routed traffic introduces on each link. The network can preallocate these resources by limiting long-lived flows to some portion c_ℓ of the link capacity C. A larger value of c_ℓ provides greater flexibility to the long-lived flows, while a smaller value of c_ℓ devotes more resources to the statically-routed traffic. Despite the advantages of allocating resources for short-lived flows, selecting c_ℓ too small would increase the likelihood of "blocking" of dynamically-routed, long-lived flows. Such blocking would, in turn, increase the resources consumed by statically-routed traffic to carry the "blocked" long-lived flows. As illustrated in Section 6.6, the effectiveness of our hybrid routing scheme is not very sensitive to selecting a relatively large value of c_ℓ, since excess short-lived traffic can exploit underutilized resources that were allocated for long-lived flows.

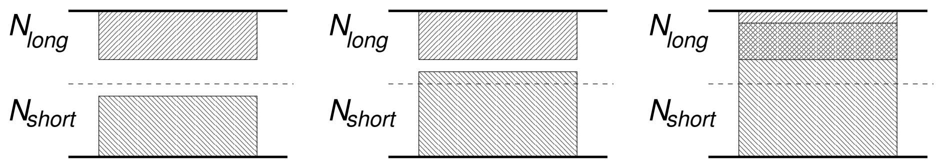
Fortunately, although the traffic load may vary across time, the distribution of the number of packets and bytes in a flow is relatively stable, particularly on the timescale of dynamic path selection; hence, the flow-size distribution can be determined in advance, based on network traffic measurements. We note, though, that the flow-size distribution may not be the same on every link. For example, the distribution of flow sizes for an access link to a video server would differ from the distribution near a DNS server. These links would arguably devote different proportions of resources to short and long flows. In the core of the network, the mixing of traffic from a variety of applications and source-destination pairs should result in relatively similar flow-size distributions across different links. The simplest provisioning model divides link bandwidth in proportion to the amount of statically-routed and dynamically-routed traffic. As an example, we consider the distribution $f(x)$ of the number of packets in a flow. Suppose a flow is dynamically routed after T packets have arrived. That is, a proportion $f(T)$ of the flows are short-lived. Let ℓ_s and ℓ_l be the average number of packets in the short-lived flows (i.e., flows less than T packets long) and the long-lived flows, respectively.

All of the traffic in the short-lived flows, and the first T packets of each long-lived flow, travel on static preprovisioned shortest paths. In the absence of blocking, the remainder of the traffic in the long-lived flows travels on dynamic routes. This suggests that c_ℓ should be a fraction

$$\frac{(1-f(T))(\ell_l - T)}{f(T)\ell_s + (1-f(T))\ell_l}$$

of the link capacity C. The value of T should be large enough to control the fluctuations in the link-state metrics for the dynamically-routed traffic, yet small enough to ensure that a large amount of traffic can be dynamically routed. In addition, if T is too large, the capacity c_l dedicated to long-lived flows may not be large relative to the flow bandwidth b; this would introduce additional blocking of long-lived flows due to bandwidth fragmentation. Fortunately, the heavy-tailed flow-size distribution $f(x)$ ensures that c_l is at least half of the link capacity, even for large flow triggers T. Bandwidth fragmentation should not be a problem for reasonable flow trigger values. The simulation experiments in Section 6.6 address how to select the appropriate value of T that balances the various cost-performance trade-offs.

The allocation rule based purely on $f(x)$ is unnecessarily conservative for the short-lived traffic in two key ways. First, the provisioning rule does not account for any blocking of long-lived flows. Although the blocking probability could be simulated, and incorporated into the resource partitioning equation, the target network configuration should have a negligible blocking probability. In the unusual case when blocking occurs, subsequent routing attempts for the long-lived flow are likely to be successful. Second, since long-lived flows have greater routing flexibility, they can more easily adapt to limited link resources, allowing a slight over-provisioning of bandwidth for short-lived



(a) Underutilization (b) Overutilization (c) Congestion

Figure 6.4: Dynamic sharing of link bandwidth

flows. The logical separation of network resources also helps the short-lived traffic, since they do not need to guard against the occasional arrival of a long-lived flow that would consume a large amount of bandwidth. This enables resource allocation for short flows to focus on the average case, rather than accounting for the high variability in aggregate load introduced by a small number of long-lived flows.

6.5 Model Extensions for Hybrid Routing

The simulation model developed in Chapter 3 allows us to study the dynamics of load-sensitive routing, and the effects of out-of-date link-state information, under reasonable simulation overheads, while still capturing much of the Internet's variable nature. In this chapter, we extend the simulation environment to model flow durations with an empirical distribution drawn from a packet trace from an actual ISP network. We also add support for flow triggering and separating traffic into short- and long-lived flows.

In the simulation experiments, we denote the short-lived and long-lived traffic classes as N_{short} and N_{long}, respectively. That is, each link has capacity c_s allocated for N_{short}, and c_l allocated for N_{long}, as described in Section 6.4.3. For simplicity, we model flows as consuming a fixed bandwidth for their duration. The scenarios in Figure 6.4 illustrate the way link capacity is shared between short and long flows. When a link is not congested, both sets of flows have sufficient resources. Even if the short-lived traffic exceeds the capacity of N_{short}, the traffic is able to consume bandwidth allocated to N_{long}, as shown in Figure 6.4(b). The allocation of bandwidth between N_{long} and N_{short} exists only to control routing, and need not dictate the link-scheduling policies. Congestion only arises in Figure 6.4(c), when the aggregate traffic exceeds the link capacity. The goal of our routing scheme is to limit the proportion of time that a link is in this state, through appropriate network provisioning and effective routing policies for N_{long}.

Aggregation	Avg. flow duration (sec) (full distribution)	Avg. flow duration (sec) (truncated distribution)
port	30.0	15.1
host	95.0	57.4
subnet	166.4	95.4
net	195.4	116.2

Table 6.2: Flow durations with different levels of aggregation

6.5.1 Traffic Model

In choosing a traffic model, we must balance the need for accuracy in representing Internet traffic flows with practical models that are amenable to simulation of large networks. The inherent variability of flow durations, arrival processes, and flow bandwidths complicates the simulation task by making convergence unlikely within reasonable simulation time. Our approach is to make some simplifying assumptions, while remaining representative of the long-lived flows we wish to study.

Flow durations: To accurately model the heavy-tailed nature of flow durations, we use an empirical distribution from a one-week packet trace collected in June 1997 from a single access point of the AT&T WorldNet ISP network. The trace consisted of $795,446$ port-to-port, $199,638$ host-to-host, $87,336$ subnet-to-subnet, and $51,046$ net-to-net flows, each using a 60-second timeout. Net- and subnet-level flows are classified in the trace by matching the first two or three octets, respectively, of the source and destination IP addresses. As shown in Figure 6.2(a), the data exhibits a heavy tail both in terms of the flow duration and the traffic volume relative to the number of flows. Such variability in the traffic introduces a fundamental challenge in simulation, requiring extremely long runs to reach convergence while assuring that the distribution is fully sampled and simulated. For this reason, we truncate the distributions at $1000, 1500, 2000$, and 2000 seconds, which still accounts for 99.8%, 99.3%, 99.3%, and 99.0% of the port-to-port, host-to-host, subnet-to-subnet, and net-to-net flows, respectively. Note that this understates the advantages of our hybrid routing scheme, which actually benefits from the very long-lived flows in the tail of the distribution. Table 6.2 shows the average flow durations before and after truncation of the distributions.

Flow arrivals: Each router in the network generates flows according to a Poisson process with rate λ, with a uniform random selection of the destination router. The value of λ, is chosen to fix the offered network load, ρ, at a particular value ($\rho = 0.8$ in most of our experiments). This assumption slightly overstates the performance of the traditional dynamic routing scheme, which would normally have to deal with more bursty arrivals of short-lived flows. Burstiness in the flow-arrival process tends to degrade the performance of load-sensitive routing, particularly under out-of-date link-state information, as illustrated in Chapter 4. Long-lived flows typically have a less bursty arrival process [28]. Hence, this assumption slightly biases our results in favor of traditional dynamic routing.

Flow bandwidth: Flow bandwidth is uniformly distributed with a 200% spread about the mean $\bar{b}$ to reflect heterogeneity in the traffic. The value of $\bar{b}$ is chosen to be about $1 - 5\%$ of the average link capacity. Smaller bandwidths, while perhaps more realistic, inflate simulation time significantly since many more flows must arrive for the links to reach the high utilization regime we are interested in. Higher bandwidth values may also be more representative of aggregated flows, which would consume a larger portion of link capacity.

Traffic pattern: Except where noted, the experiments evaluate a network where static routing is well-matched to the volume of traffic between the end-points. This allows us to focus on the effects of out-of-date link-state information in a well-provisioned network, rather than the ability of load-sensitive routing to circumvent hot-spots. Some experiments in Section 6.6 also consider slightly non-uniform traffic. In this scenario each router sends 20% of its traffic to a randomly-selected set of 15 destinations; these "hot" destinations receive 30% more traffic than in the earlier experiments. The rest of the traffic is evenly distributed amongst the remaining destinations.

6.5.2 Provisioning and Capacity Allocation

In evaluating our hybrid routing scheme, we focus on a network provisioned according to the expected traffic load. In a production network this is typically done on a very coarse timescale by a network administrator who configures link weights (e.g., in OSPF) or tagged routes (e.g., in MPLS) to control the distribution of traffic over the links in the network. We follow a slightly different approach of first fixing the target topology, and then sizing the link capacities so that link utilization is uniform throughout the network, similar to the approach in [5].

Network topology: Our evaluation model focuses on backbone networks with relatively high connectivity keeping with the trend towards more highly connected networks. Rather than considering regular graphs, which may hide important effects of heterogeneity and non-uniformity, we consider a 100-node random topology, generated using Waxman's model [90, 94]. The topology has degree 5.6, diameter 5, and an average hopcount between each source-destination pair of 2.8. Note that this topology is slightly different from the random graph used in Chapter 4.

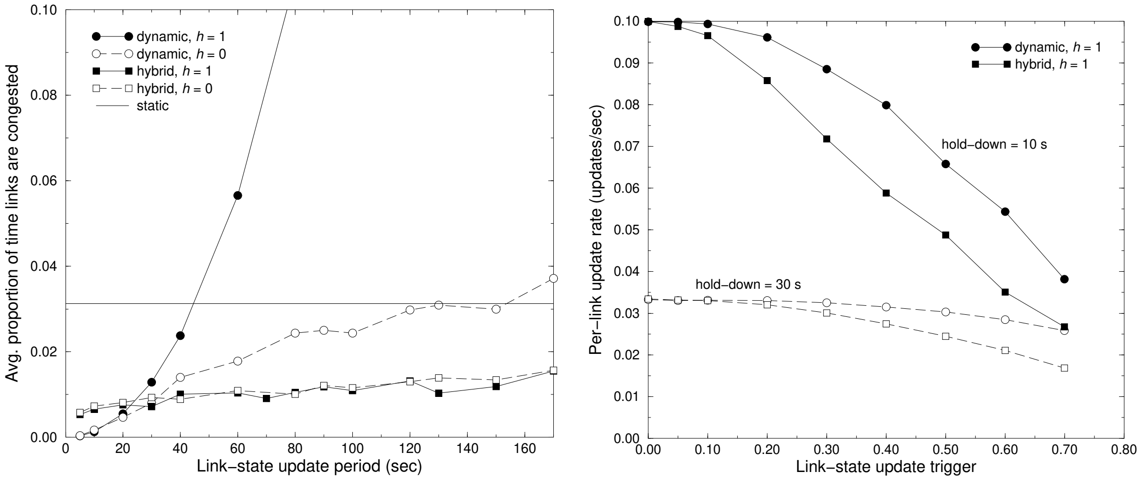
In assuming that propagation and processing delays are negligible, our model focuses on the primary effects of stale link-state information on path selection. This assumption slightly biases our results in favor of the traditional dynamic routing algorithm, which would suffer additional route flapping under small update periods due to the feedback delays. These effects are much less significant for our hybrid scheme, since propagation delays are negligible relative to the duration of long-lived flows. Each link introduces a small random component to the generation of successive updates to prevent update synchronization [30].

Network provisioning: After computing all shortest paths between each pair of routers, we determine the traffic volume on each link, assuming that a source communicates with equal frequency with each destination, and set its capacity proportional to $\lambda \ell \bar{b}$. When multiple shortest paths are present between a node-pair, links on those paths are dimensioned assuming a uniform random selection among the paths (i.e., links are assigned capacity to handle an equal fraction of the total traffic volume between the node-pair). Provisioning in this way essentially produces a load-balanced network with no "hot spots". Note also that considering a well-provisioned network creates a best-case scenario for static routing. This understates the ability of dynamic routing to adapt to shifts in the underlying traffic matrix.

Resource allocation: After sizing the network links, we use the duration distribution to allocate link capacity to N_{short} and N_{long}. Choosing a particular flow trigger (in seconds) determines the proportion of flows that are routed on N_{short} or N_{long}, as well as the mean duration of flows on either partition. Average residual lifetime of flows after applying a particular trigger (i.e., mean duration of flows on N_{long}) is shown in Figure 6.2(b) (without truncation of the distribution). Capacity is then allocated to each partition as described in Section 6.4.3, except that the flow trigger is time-based rather than packet-based. Hence, we implicitly assume that the number of bytes in a flow is proportional to its duration. Although this ignores effects that might inflate the lifetime of a flow (e.g., a slow bottleneck link, or TCP retransmissions due to loss) the collected data generally supports this assumption.

6.6 Performance Evaluation

This section evaluates the hybrid routing approach based on detailed simulation experiments, since tractable analytical models for studying the effects of out-of-date link-state information in



(a) Link-state update periods (b) Link-state triggers (for $h = 1$)

$b \sim (0.0, 0.02)$ avg. capacity, $\ell = 15.07$ sec, $\lambda = 11.7$ flows/sec, $\rho = 0.80$

Figure 6.5: Effects of link-state staleness

large networks are elusive. We compare our hybrid scheme to traditional static and dynamic routing under a range of link-state update periods and triggers. We show that, in contrast to traditional dynamic routing, the hybrid scheme performs well under a wide range of link-state update frequencies. Then, we investigate how to further improve its performance by tuning the flow trigger to the timescale of link-state update messages. Finally, we show that our approach is robust to inaccuracies in network provisioning under both uniform and non-uniform traffic.

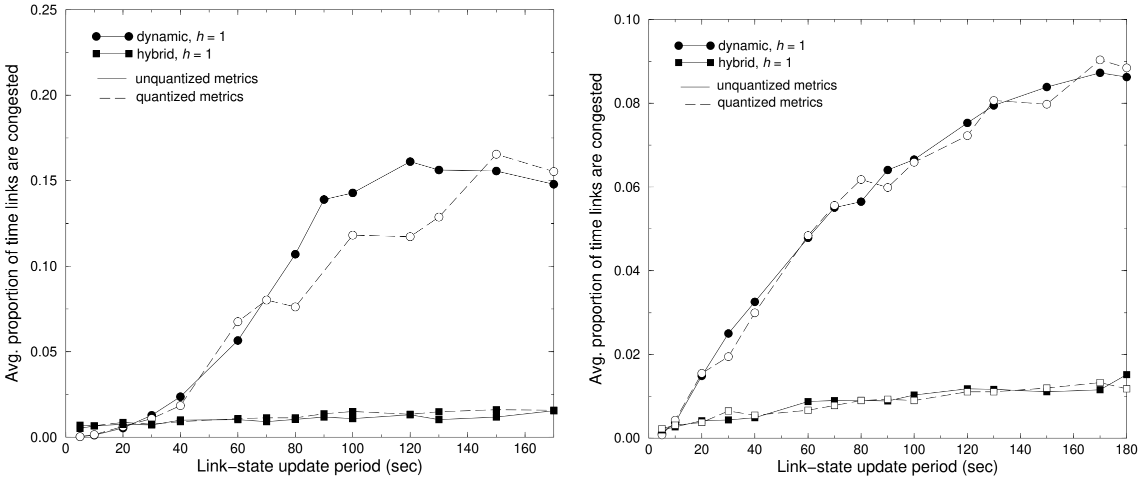
6.6.1 Stale Link-State Information

To investigate the effects of stale link-state information, Figure 6.5(a) plots routing performance as a function of the link-state update period for static routing, dynamic routing, and our hybrid scheme. This graph mirrors the results in Figure 6.1(a), except that the earlier experiment in Section 6.3 evaluated a network that blocks a flow after an unsuccessful attempt to reserve resources on the selected route. In contrast to the traditional blocking model in QoS routing, we focus here on a non-blocking model, where a flow defaults to a static shortest-path route when the dynamically-selected path cannot support the additional traffic. Hence, rather than plotting a blocking probability, Figure 6.5(a) plots the proportion of time that the aggregate bandwidth requirements of the flows routed to a link exceeds the capacity (averaged over all links), as illustrated in Figure 6.4(c). The performance trends as a function of the link-state update period are similar to the earlier results for the blocking model in Figure 6.1(a) for traditional static and dynamic routing.

Under small link-state update periods traditional load-sensitive routing typically outperforms static routing and our hybrid scheme, as shown in Figure 6.5(a). However, performance degrades dramatically under larger link-state update periods. The poor performance of traditional dynamic routing under stale link-state information is also exacerbated by the consideration of nonminimal routes, as shown by the differences between the $h = 0$ and $h = 1$ curves. The $h = 1$ curve in Figure 6.5(a) eventually flattens at a level of about 0.15. Route flapping increases the likelihood that shortest-path routes are busy, and out-of-date information can also cause the dynamic routing algorithm to select a nonminimal route even when a feasible shortest-path route is available. In fact, for reasonable link-state update periods in the tens of seconds, static routing is preferable to traditional dynamic routing. Even when dynamic routing is restricted to shortest paths ($h = 0$), performance degrades significantly as the link-state update period increases, and eventually performance becomes worse than static routing.

In contrast, using a flow trigger of 20 seconds allows our hybrid scheme to perform well across a wide range of update periods. For the port-to-port flows shown in Figure 6.5, a 20 second flow trigger places 15% of the flows and 60% of the traffic on N_{long}, and results in an average residual flow duration of 83 seconds. Under accurate link-state information, the hybrid scheme performs slightly worse than traditional dynamic routing, since the short-lived flows (and the first 20 seconds of the long-lived flows) are forced to travel on static routes. In this regime, link-state updates are distributed so frequently that flapping due to staleness is unlikely under any of the routing schemes. Under more reasonable update periods, however, the effectiveness of the hybrid scheme becomes clear, as the performance does not degrade much even for periods in the range of up to several minutes. By applying dynamic routing only to the long-lived flows, the hybrid routing scheme does not suffer from rapid fluctuations in link state, and can better exploit the presence of nonminimal routes. For the uniform traffic load considered in this experiment, the ability to choose nonminimal routes does not significantly improve performance, but the additional routing flexibility is critical to adapting to fluctuations in the underlying traffic pattern.

To further investigate the influence of the link-state update policy, Figure 6.5(b) considers a range of link-state update triggers, subject to a hold-down timer. A trigger of 0.0 corresponds to periodic updates, based on the hold-down timer. In this case, the performance matches the results in Figure 6.5(a) for periods of 10 and 30 seconds, respectively. Consistent with results in Chapter 4, we find that larger update triggers do not change the likelihood that the routing algorithm finds a



(a) Random topology

parameters as in Figure 6.5

(b) Uniform topology

$b \sim (0.0, 0.03), \ell = 15.07$ sec, $\lambda = 7.5$ flows/sec, $\rho = 0.75$

Figure 6.6: Effects of link metric quantization

feasible route, though staleness does increase the likelihood that failures are discovered during path set-up rather than path selection. Since performance does not vary with the link-state update trigger, we only plot the resulting link-state update rate. As expected, larger hold-down timers and larger triggers both result in fewer link-state updates. The decrease in the number of dynamically-routed flows, and the reduction of route flapping, ensures that the hybrid scheme has a lower link-state update frequency than traditional dynamic routing. Though it might be expected that the hybrid scheme would have a considerably lower triggered update rate (since updates are for only a fraction of the traffic), the partitioning of network bandwidth under the hybrid scheme limits the reduction. For example, a 30% change in the available bandwidth on N_{long} corresponds to less than a 30% change relative to the entire link capacity. So trigger thresholds are relative to a smaller link capacity. Still, despite this effect, the hybrid scheme achieves an appreciably lower link-state update rate than the traditional dynamic scheme.

Chapter 5 (as well as work in [5]) showed that coarse-grained or quantized link metrics can be used to improve the ability of dynamic routing to use alternate minimum-hop routes by increasing the likelihood of “ties” between routes. Since the results shown in Figure 6.5 use unquantized link metrics for available bandwidth, we conducted additional experiments to investigate the benefits of coarse-grained metrics in improving the relative performance of traditional load-sensitive routing versus hybrid routing. We use link-state update values that are quantized into equal classes of

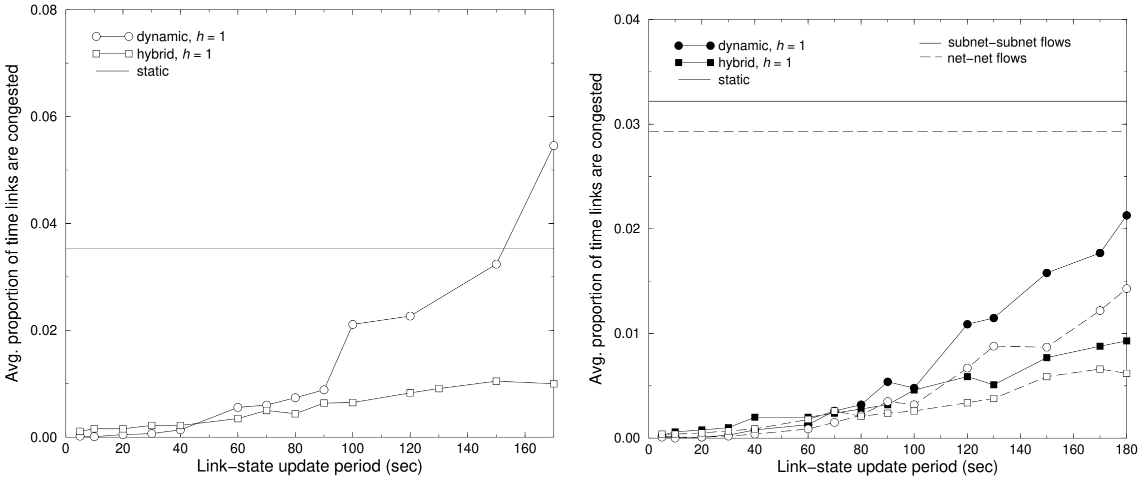
Aggregation	Proportion of flows	Proportion of traffic	Residual duration (sec)
port	$.15$	$.60$	83
host	$.39$	$.82$	141
subnet	$.51$	$.87$	184
net	$.53$	$.89$	213

Table 6.3: Characteristics of long flows with varied aggregation and a 20 second flow trigger

50% of the total bandwidth request range (as described in [5]) with $h = 1$. Thus, if flows require bandwidth in the range $(0, 2b)$, the available link bandwidth is advertised as the midpoint of the classes $(0, b), (b, 2b), (2b, 3b), \ldots$ As illustrated in Figure 6.6(a), quantization reduces route flapping for dynamic routing somewhat, but the performance gains are minor, particularly under update periods which are very long relative to the average flow duration. Figure 6.6(b) illustrates the results of additional experiments with the uniformly-connected 64-node 4-ary 3-cube (see Section 4.2.3 in Chapter 4). In this topology, 90% of the node-pairs have more than one shortest-path. Since there are already multiple shortest-paths to select from, there is little additional advantage to using quantized metrics for dynamic or hybrid routing.

We also consider the effects of link-state staleness when traffic is further aggregated to the host, subnet, or net levels, as shown in Figure 6.7. Table 6.3 lists the characteristics of the long-lived flows after applying a 20 second flow trigger. As the aggregation increases, the performance advantage of hybrid routing over dynamic routing diminishes somewhat but is still significant, especially in the case of host-to-host aggregation. Observe from Figure 6.2 that when flow triggers are small, the residual average duration increases with more aggregation (Figure 6.2 does not reflect the truncation of the distribution). Thus, even when all flows are dynamically routed (i.e., flow trigger of 0), the average flow duration is 57.4 seconds and 95.4 seconds for host- and subnet-level aggregation, respectively. Since the overall flow duration is longer, dynamic routing suffers less from flapping, and its performance is improved. For example, there is no crossover between traditional dynamic routing and static routing in the case of subnet- and net-level aggregation. Notice that hybrid routing still outperforms dynamic routing and static routing for a range of reasonable link-state update periods.

In Figure 6.8, we examine the performance when the traffic is non-uniform, according to the pattern described in Section 6.5.1. When the traffic is imbalanced, the gains from dynamic routing are more evident, as shown in Figure 6.8(a). Both traditional dynamic (with accurate information)



(a) Host-level flows
$\lambda = 3.07$ flows/sec

(b) Subnet- and net-level flows
subnet: $\lambda = 1.85$ flows/sec
net: $\lambda = 1.52$ flows/sec

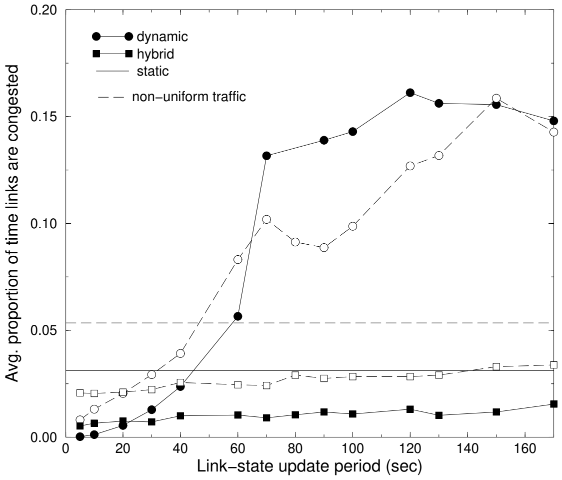
$b \sim (0.0, 0.02)$ avg. capacity, $\rho = 0.80$, durations as in Table 6.2

Figure 6.7: Effects of increased traffic aggregation

and hybrid routing perform better relative to static routing when the traffic is non-uniform. But performance is degraded overall since the random graph does not have a high proportion of source-destination pairs with multiple similar length paths, limiting the flexibility of dynamic routing. For instance, only about 50% of the node-pairs have more than one minimum hopcount path between them. Figure 6.8(b) considers non-uniform traffic in the 4-ary 3-cube. The greater connectivity results in only a minor performance degradation under non-uniform traffic, particularly for hybrid routing. Moreover, the relative performance gains of hybrid routing over static routing are greater in the non-uniform case when the topology offers greater flexibility.

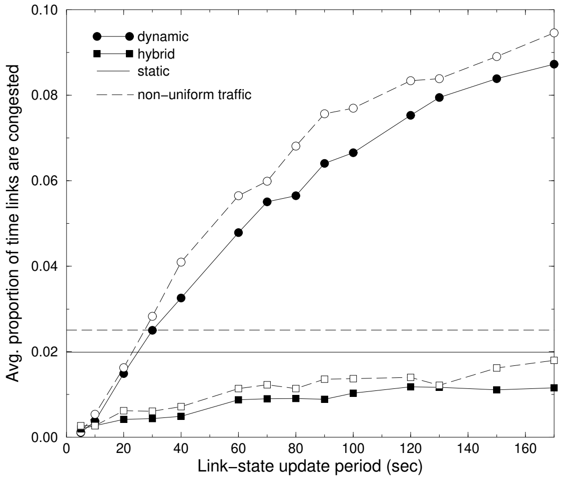
6.6.2 Detecting Long-Lived Flows

The cost-performance trade-offs of our hybrid scheme relate directly to the selection of the flow trigger, as shown in Figure 6.9. Figure 6.9(a) illustrates that the hybrid scheme substantially reduces the frequency of dynamic route computations, particularly for larger values of the flow trigger. This result stems directly from the reduction in the number of flows that are dynamically routed. The link-state update period has a very small, second-order effect. Under out-of-date link-state information, the hybrid scheme occasionally selects an already-congested route on N_{long}, forcing a subsequent routing attempt after activating the flow trigger a second time. This effect is very



(a) Random topology

parameters as in Figure 6.5



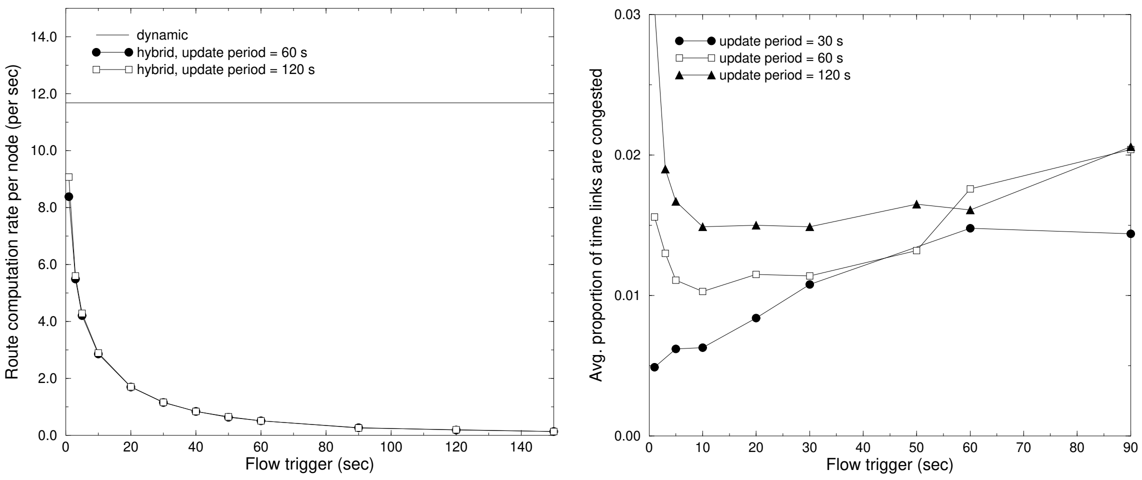
(b) Uniform topology

parameters as in Figure 6.6(b)

Figure 6.8: Non-uniform traffic with $h = 1$, port-to-port flows

small for two reasons. First, since relatively high link-state periods do not appreciably degrade the performance of the hybrid scheme, these recomputations occur very infrequently. Secondly, the flow trigger is large enough to ensure that the second routing attempt will be initiated only after new link-state information is available, particularly when the flow trigger is larger than the link-state update period. As a result, two routing attempts do not typically operate on the same link-state database, in contrast to traditional rerouting or "crankback" operations that are likely to draw on the same information about most of the links. Implicitly introducing delay between the successive routing attempts increases the likelihood of selecting a feasible route.

Despite the advantages of large flow triggers in reducing computational overheads, a smaller flow trigger ensures that more traffic can be routed dynamically. The graph in Figure 6.9(b) investigates how to select the flow trigger to strike the best balance between stability and adaptiveness in the hybrid scheme. Each setting of the flow trigger corresponds to a different division of the network resources between N_{long} and N_{short}, following the provisioning rule in Section 6.4.3. For a range of link-state update periods and network configurations, the graphs have roughly a cup shape, with worse performance for smaller and larger flow triggers. A small flow trigger allows dynamic routing of a larger proportion of the traffic, at the risk of greater sensitivity to stale link-state information. The flow trigger controls these staleness effects by determining the residual duration distribution for the flows on N_{long}. Hence, the flow trigger should be chosen such that most flows on N_{long}



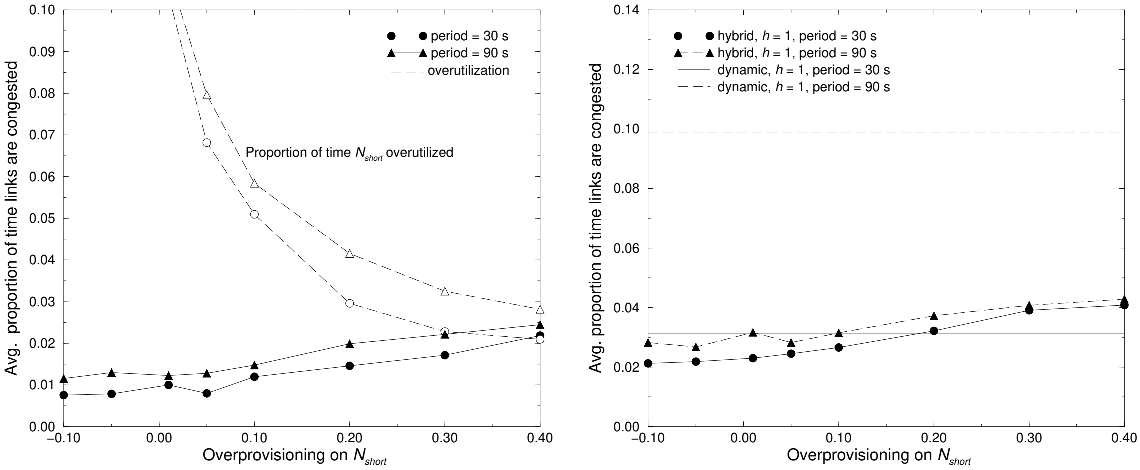
(a) Route computation rate (b) Avg. time links are congested

parameters as in Figure 6.5

Figure 6.9: Choice of flow trigger

have a residual lifetime that is large relative to the link-state update period. When the update period is small (e.g., 30 seconds), choosing small flow triggers that assign more traffic to dynamic routes improves performance since dynamic routing does not suffer from much flapping in this staleness regime.

The control over stale link-state information afforded by large flow triggers also comes at a cost. A large flow trigger limits the proportion of traffic that is dynamically routed, which degrades the ability of the hybrid algorithm to respond to fluctuations in the offered traffic load. In addition, large flow triggers effectively allocate fewer resources to N_{long}, which increases the likelihood of bandwidth fragmentation, resulting in more "blocking" of the dynamically-routed flows. These effects are demonstrated by the gradual rise of the curves in Figure 6.9(b) for larger flow triggers. The degradation in performance is not very significant, since an increase in the flow trigger does not have a very substantial impact on the proportion of traffic on N_{long}, due to the heavy tail of the flow-size distribution. For example, flow triggers of 20 seconds and 40 seconds place 60% and 47% of the traffic on N_{long}, respectively. Still, to maximize the hybrid algorithm's ability to react to shifts in traffic load, the flow trigger should be made as small as possible, subject to the link-state update period and the target route computation rate.



(a) Uniform traffic (b) Non-uniform traffic

parameters as in Figure 6.5

Figure 6.10: Over-provisioning for short-lived flows

6.6.3 Network Provisioning

The previous subsection illustrated that the hybrid routing scheme is robust across a range of flow triggers. In this section, we evaluate the sensitivity of our scheme to inaccuracies in allocating resources between N_{short} and N_{long}. This is an important consideration since parameters such as the flow trigger, which are configured by a network administrator, are always subject to error. Figure 6.10(a) plots the performance of our hybrid scheme with a 20-second flow trigger across a range of over-provisioning factors, for a uniform traffic pattern. An over-provisioning factor of 0 corresponds to the earlier simulation results, using the provisioning rule in Section 6.4.3 to divide link bandwidth between N_{short} and N_{long}. A larger over-provisioning factor implies additional resources devoted to N_{short}, at the expense of N_{long}, without changing the flow trigger.

The effectiveness of our hybrid scheme remains relatively undiminished as the resource allocation changes, though large over-provisioning factors typically result in slightly worse performance. This occurs because an under-provisioned N_{long} sometimes rejects flows, which are forced to stay on static routes on N_{short}. In contrast, if N_{long} is over-provisioned, the long-lived flows are rarely blocked. In either case, the selection of widest paths on N_{long} helps ensure that the dynamically-routed traffic does not consume the full allocation of each link, allowing the short-lived flows to consume the excess capacity. Links are congested about 1–2% of the time for the uniform traffic pattern in Figure 6.10(a). During these infrequent periods of congestion, the average amount of excess traffic (not shown) is approximately 8% across a range of over-provisioning factors. This implies that the network experiences only brief periods of minor congestion, even when the bandwidth provisioning rule does not exactly match the proportion of short-lived and long-lived traffic.

Despite the potential advantages of under-provisioning the resources on N_{short}, devoting too much of the link resources to long-lived flows makes the network more vulnerable under changes in the traffic pattern, as suggested by the dashed lines in Figure 6.10(a), which plots the proportion of time that the short-lived traffic exceeds its allocation on a link in N_{short}. Typically, when the allocation of resources for short-lived flows is overutilized, the corresponding resources for long-lived flows are underutilized, and the link is not actually congested. But, this does not necessarily remain true under shifts in the traffic pattern. To quantify the risk of under-provisioning N_{short}, we experimented with a non-uniform traffic pattern, as shown in Figure 6.10(b).

The non-uniform traffic increases the proportion of time that links are congested, as shown by the higher curve in Figure 6.10(b). The links are congested 2–4% of the time, across the range of over-provisioning factors. But, the degree of congestion increases when N_{short} is under-provisioned. For example, the congested links have an average of 16% excess traffic for a over-provisioning factor of -0.10, compared to just 11% for an over-provisioning factor of 0.30 (not shown). When N_{short} does not have sufficient resources, a significant amount of excess traffic can be dynamically routed to certain links. This occurs because the network continues to route long-lived flows to congested links on N_{long}, even when the link is already busy. A larger over-provisioning factor on N_{short} effectively acts as a form of trunk reservation [36] that controls the proportion of link resources that are devoted to load-sensitive routing. This helps ensure stability under fluctuations in the traffic pattern. These results suggest that the network resources should be divided in proportion to the amount of short-lived and long-lived flows, with perhaps a slight over-provisioning of N_{short}, as discussed in Section 6.4.3. More importantly, across a range of provisioning rules, the hybrid scheme significantly outperforms traditional dynamic routing on the non-uniform traffic pattern, particularly under larger link-state update periods.

6.7 Summary

In this chapter, we have introduced a dynamic routing scheme that exploits the extreme variability in IP flow durations by performing dynamic routing of long-lived flows, while forwarding short-lived flows on static preprovisioned paths. Route stability is achieved by relating the detection of long-lived flows to the timescale of the link-state update messages in the routing protocol. Link bandwidth resources are allocated between the two traffic classes based on measurements of the flow-size distribution, and the trigger used for detecting long-lived flows. Our simulation experiments demonstrate that the proposed hybrid scheme significantly outperforms traditional static and dynamic routing algorithms, and can operate effectively under reasonable link-state update periods in the range of 60–180 seconds. In addition, we show that our scheme is robust to inaccuracies in network provisioning and shifts in the offered traffic.

Future work on hybrid routing may include collection and analysis of additional Internet traces to study specific aspects of our hybrid routing scheme in greater detail. These problems include the provisioning rules for short-lived flows and the possibility of routing long-lived flows based on measured link utilization rather than reserved bandwidth, as well as the impact of TCP dynamics on load-sensitive routing. In addition, measuring and analyzing the traffic volume between pairs of routers in the network would provide insight into how much the offered load fluctuates, and on what timescale. We are also investigating generalizations of the hybrid routing scheme, including dynamic selection of the flow trigger based on the traffic characteristics, and support for precomputation of load-sensitive routes to avoid the processing overheads and set-up delays introduced by on-demand path selection. These studies can provide insight into how to best exploit the flexibility and efficiency of this approach for load-sensitive routing of IP traffic.

CHAPTER 7

CONCLUSIONS

Much of the phenomenal success of the Internet, and particularly the World-Wide Web, is attributable to the simplicity and ease with which new services can be deployed using the packet delivery service implemented by the IP protocol. This service falls short, however, in the face of the growing variety of applications that impose additional requirements on performance, reliability, security, and availability. Further, standard shortest-path IP routing limits the ability to redirect traffic automatically when load conditions change over short timescales. In contrast, dynamic or QoS routing offer the advantage of choosing paths that balance the load in the network and meet application QoS requirements. These benefits do not come without drawbacks, though, as dynamic routing protocols often consume computational and bandwidth resources at a substantially higher level than conventional routing protocols.

This dissertation investigates the performance-overhead tradeoffs of dynamic routing in realistic network settings. After characterizing these trade-offs through representative models and extensive simulation-based evaluation, the dissertation follows with novel algorithms for improving the efficiency of dynamic routing. The remainder of this chapter discusses the contributions of this dissertation in more detail, and also presents some related problems that are the subject of future research.

7.1 Research Contributions

This dissertation contributes a number of new insights into the behavior of load-sensitive routing protocols. Using several proposed QoS routing algorithms as a basis, Chapter 3 presents a general and flexible model to study the important parameters affecting the performance and overhead of dynamic routing. A major contribution of the model is its clear formulation of dynamic routing in terms of a set of route computation algorithms, routing, signaling, and update policies, and network and traffic configurations. Furthermore, the components of the model are practically motivated by existing static and dynamic routing protocols, other research efforts to characterize Internet traffic and topologies, and work on design and evaluation of path computation algorithms.

A significant component of this research, also described in Chapter 3, is the realization of the routing model in `routesim`, a simulation environment developed to enable experimentation with a wide variety of dynamic routing configurations. The simulator is modularized to separate functions for routing algorithms, topology and network traffic generation, and statistics collection. In this way, each of these pieces of software is individually useful for other network simulation work.

Recognizing that dynamic routing in real networks must operate with inaccurate information about network state, the performance evaluation in Chapter 4 focuses on this critical issue while examining the performance trade-offs of dynamic routing. The evaluation constitutes the first in-depth investigation of the effects of stale link-state information on dynamic routing protocols under an uncommonly wide variety of realistic network configurations. The most significant contributions are the numerous specific guidelines and observations arising from the evaluation. These guidelines assist service providers in configuring the network and setting policies for dynamic or QoS routing.

Our initial model and evaluation considered the case where QoS paths are computed for each flow request as it arrives. This on-demand approach leads to high computational overhead at each router, particularly when using more complex QoS routing algorithms. In the interest of reducing this computational overhead, other work has examined the performance trade-offs of precomputing and reusing QoS paths, rather than computing them in an on-demand fashion. These studies have not addressed more practical questions, however, such as how routes should be precomputed or how they should be stored and retrieved efficiently to enhance performance. Chapter 5 provides answers to exactly these crucial questions in the form of new algorithms and policies for realizing route precomputation. Chapter 5 demonstrates the ability of these algorithms to meet the dual goals of reducing computational resource requirements while achieving accurate route selection in the face of stale network information. Moreover, the mechanisms are applicable in a number of protocols that follow the common source-directed link-state QoS routing model.

Through the course of this research, and especially from the findings in Chapter 4, it is increasingly clear that the deployment of dynamic or QoS routing in IP networks faces obstacles. To attain good performance, for example, the frequency of link-state distribution must be increased substantially over conventional shortest-path protocols such as OSPF. While the mechanisms in Chapter 5 reduce the computational costs, the control traffic overhead remains a key problem. Other impediments arise from the extensive modification to the Internet infrastructure necessary to support dynamic routing. For example, without application-level signaling protocols such as RSVP, the usefulness of QoS routing is hampered. Also, the blocking model used in QoS routing is contrary to the fundamental "universal access" model of the Internet.

Chapter 6 addresses many of the drawbacks of load-sensitive routing in IP networks by limiting dynamic routing to a subset of the network traffic. The remainder travels on static shortest-path routes computed by the underlying conventional intradomain protocol. This hybrid routing scheme draws from the results in Chapter 4 which demonstrate that the reduced fluctuation exhibited by longer-lived traffic requires less frequent link-state distribution. In addition, other research on IP traffic characterization shows that though long-lived transfers comprise a small fraction of the flows, they account for the majority of the overall traffic volume. Thus, by dynamically routing only long-lived flows, hybrid routing is able to improve the stability and efficiency of load-sensitive routing while also providing automated load-balancing capability that reacts to fluctuations in network traffic. Hybrid routing is therefore a much more palatable scheme for deploying dynamic routing compared to traditional approaches, since an update frequency on the same order as currently used link-state protocols gives good performance. Further, the hybrid routing architecture can capitalize on flow detection and classification hardware at the network edge, obviating the need for explicit signaling by the application. By eliminating the possibility of blocking the admittance of a flow into the network, hybrid routing also fits more readily into the Internet model.

7.2 Future Research Directions

This dissertation contributes new insights and solutions for improving the efficiency of dynamic routing in wide-area networks. A number of interesting research problems remain, however, and warrant further investigation.

The Transmission Control Protocol (TCP) [81] is the most commonly used standard transport protocol in the Internet. As a result, many researchers attempt to design network control mechanisms that are "TCP-friendly". That is, care is taken not to defeat the congestion avoidance and detection mechanisms of TCP when introducing new protocols or algorithms. While the work in Chapter 6 on hybrid routing attempts to improve performance for short-lived flows which typically use TCP, it does not investigate the interactions with TCP congestion control. For example, TCP retransmissions may cause flows to be misclassified as short or long. Or, by moving high-bandwidth long-lived flows to dynamic routes, the performance of TCP connections on short-lived paths may be improved. One approach to evaluate these types of effects would be to conduct packet-level simulations of hybrid routing with accurate models of the TCP protocol, perhaps on smaller network configurations to maintain simulation tractability.

The work in this dissertation considers QoS or load-sensitive routing only for unicast flows. The growing interest in multicast applications with performance requirements, such as Internet video-conferencing, multiplayer games, and remote collaboration, requires that dynamic routing be extended to multicast. For example, several multicast protocols (e.g., PIM sparse-mode [25] and core-based trees [11]) construct a shared multicast tree by grafting branches onto the tree from the group members using unicast routing. The precomputation algorithms presented in Chapter 5 may be useful if modified appropriately to operate within such a framework.

Another under-explored area in dynamic routing is the effect of link or router failure on the QoS delivered to flows that were using the failed component. In general, a dynamic or QoS routing architecture should have some ability to transparently unpin paths (and possibly release reserved resources) and find new routes with minimal disruption. This may require the use of redundant disjoint paths for flows, or the ability to fall back temporarily to conventional shortest-path routes while a new dynamic path is computed. Some interesting issues relate to how to efficiently introduce path redundancy while still providing adequate protection, and minimizing the detrimental effects of rerouting such as packet reordering, violation of QoS, and introduction of routing loops.

With the growing interest in differentiated services as a means to provide QoS on the Internet, traditional QoS routing as motivated by the Integrated Services model is losing momentum. Instead, dynamic routing is clearly being targeted as a tool for traffic engineering, most notably as a component of MPLS (described in Section 2.5.4). The hybrid routing scheme developed in Chapter 6 fits very well into the MPLS constraint-based routing architecture. In constraint-based routing, traffic trunk definitions and their associated constraints are manually configured, and a dynamic routing protocol is expected to find a route for the traffic trunk subject to the constraints. Hybrid routing offers a concrete way to perform these functions automatically. Though it considers only long-lived flows for load-sensitive routing, with appropriate flow classifiers dynamic routing can be applied to any other defined class of traffic. Integrating hybrid routing with MPLS is an interesting problem in protocol design, that can also increase the appeal of deploying dynamic routing.

APPENDIX

APPENDIX A

`routesim` MANUAL PAGES

NAME

routesim - Dynamic routing simulator

SYNOPSIS

routesim [options]
routesim -conf *configuration file*

AVAILABILITY

Compiled for Solaris (sparc), IRIX64, and Linux (x86) platforms.

DESCRIPTION

routesim is an event-driven simulator that facilitates the study of source-directed link-state dynamic and QoS routing in large networks. To reduce simulation overhead, *routesim* operates at the flow (or connection) level, rather than simulating individual data packets. The simulator takes as input a variety of parameters that allow precise control over network traffic characteristics, network topology, routing algorithm and policies, and link-state update policy. Though all parameters can be specified on the command line, it is more convenient to place them in a configuration file. Parameters specified on the command line override any specification in the configuration file.

OPTIONS

Each option is shown with both its short and long forms. The short form is used on the command line and the long form, shown in parentheses, is used in the configuration file. Most options have arguments that are either values (i.e., numbers) or strings. When the argument is listed as *value* it should be a number. Arguments shown as [<opt1> | <opt2> ...] require one of the given <opt> values.

-accuracy [CURRENT | STALE]

(link-state-accuracy)
CURRENT causes link-state to be perfectly accurate, while STALE makes link-state subject to updates.

-alpha *value*

(link-cost-alpha)
Link cost function exponent (steepness)

-altrouting [ROUND_ROBIN | STICKY | NO_ALT]

(alternate-routing)
Alternate routing policy when using precomputed routing and depth-first route extraction. No effect with on-demand routing.

-arrdist [WEIBULL | POISSON]

(arrival-dist)
Flow interarrival distribution to use.

-arrscale *value*

(arrival-scale)
Scale parameter for interarrival distribution. In the case of POISSON, scale is the mean.

-arrshape *value*

(arrival-shape)
Shape parameter for WEIBULL flow interarrival distribution.

-arrspec [ARR_UNIFORM | ARR_FILE]

(arrival-spec)
Traffic matrix specification. ARR_UNIFORM gives a uniform traffic pattern, while ARR_FILE reads the src-dest arrival rates from a file called *rstraffic.dat*

-bandwidth *value*

(mean-bandwidth)
Mean requested flow bandwidth.

-blocking [NON_BLOCKING | BLOCKING]

(blocking-policy)
Blocking model to use in the case of SINGLE network-model option. See the netmodel option.

-bwclass *value*

(bw-class-size)
Base bandwidth class size, B (for equal class based updates). Avail bw will be advertised as $\ldots, (-B, 0), (0, B), (B, 2B), (2B, 3B), \ldots$

-costconst *value*

(link-cost-constant)
Additive bias for link costs when using SPT routing-algorithm option.

-costlevels *value*

(link-cost-levels)
Number of discrete cost levels when using SPT routing-algorithm option.

-dist [EXP_SINGLE_COST | EXP_MINHOP_BIAS | HOPCOUNT | WIDEST]

(distance-function)
Cost function for SPT routing-algorithm. HOPCOUNT produces shortest-path routes. EXP_MINHOP_BIAS uses the exponential cost function but always biases toward shorter routes. EXP_SINGLE_COST uses the exponential cost function but adds a constant defined by the link-cost-constant option. WIDEST produces a shortest-widest algorithm with ties between multiple min-hop paths broken by choosing the one with most available link bandwidth.

-dualalg [SPT | WIDE_SHORT]

(dual-dyn-algorithm)
Dynamic routing algorithm to use in DUAL network model for hybrid routing. SPT uses the Dijkstra-based algorithms with policies specified by the normal SPT options (e.g., distance-function, alternate-routing, multi-route-policy, prune-policy). WIDE_SHORT uses the Bellman-Ford-based widest-shortest path algorithm.

-dumptop

(dump-topology)
Dump the topology to a file *rstopology.dat.out*

-failrecomp

(recompute-after-fail)
In precomputed routing, recompute routes after final set-up failure

-feascheck

(feasibility-check)
In precomputed routing, perform feasibility check on extraction.

-flowtrigger *value*

(flow-trigger)
In DUAL nework-model, this is the long-lived flow detection threshold (time)

-holdspec [EXP | PARETO | USERDEF]

(holding-time-spec)
Flow duration distribution, exponential, Pareto, or user defined from a file *holdcdf.dat*

-hopthresh *value*

(hopcount-threshold)
Number of hops allowed beyond min-hop route. Only applies in WIDE_SHORT routing-algorithm option.

-hscale *value*

(holding-time-scale)
Scale parameter for Pareto duration distribution or mean for exponential distribution.

-hshape *value*

(holding-time-shape)
Shape parameter for Pareto duration distribution.

-kndim *value*

(kn-dimen)
Dimensions in k-ary n-cube. Only applies in kncube topology-type option.

-knedge *value*

(kn-edge)
Edges in k-ary n-cube. Only applies in kncube topology-type option.

-level *value*

(confidence-level)
Confidence level on blocking rate or overutiliziation, depending on blocking model.

-mail *address*

(mail-recipient)
Recipient for simulation notifications.

-maxdiam *value*

(max-rand-diameter)
Maximum diameter for random topology generated at simulation time.

-maxhops *value*

(max-hopcount)
Maximum hopcount allowed. Checked during route extraction phase.

-maxsigs *value*

(max-sig-attempts)
Maximum number of signaling attempts per flow request – applicable for precomputed routing.

-mincost *value*

(link-cost-minval)
Available bandwdith corresponding to minimum link cost value, when using SPT routing-algorithm.

-minint *value*

(min-update-interval)
Minimum interval between link updates, i.e. hold-down timer.

-mintime *value*

(min-sim-time)
Minimum simulation time.

-monlink

(monitor-link)
Monitor and log link state over time. List of links to monitor is specified in *rsmonitor.dat*

-multipolicy [MULTI | FIRST | UNIQ]

(multi-route-policy)
Multi-path routing policy when using SPT routing-algorithm option. MULTI extracts a path with the option of checking link-feasibility and backtracking. UNIQ extracts a path according to the uniq-route-policy option. FIRST does no multi-path route extraction.

-netmodel [DUAL | SINGLE]

(network-model)
DUAL is the hybrid routing scheme which applies dynamic routing to a subset of flows using the WIDE_SHORT routing algorithm and shortest-path to the remainder using SPT. SINGLE applies the algorithm specified by the routing-algorithm option to all flows.

-netsplit *value*

(network-split)
Network capacity split for short/long traffic. The *value* sets the proportion of capacity placed on the short partition.

-pdrecomp

(periodic-recomputations)
Use periodic route recomputation (in precomputed routing)

-printtop

(print-topology-stats)
Print topology stats and exit

-prune [DEFAULT_PRUNE | NO_PRUNE]

(prune-policy)
Prune policy when using SPT routing-algorithm option. Pruning removes seemingly infeasible links from the route computation.

-prunerecomp

(prune-on-recomputation)
Prune links that failed in signaling when recomputing routes (in precomputed routing)

-ralg [SPT | WIDE_SHORT]

(routing-algorithm)
Sets the routing algorithm for the SINGLE network-model. SPT is a Dijkstra-based algorithm and WIDE_SHORT is the Bellman-Ford algorithm by Guerin.

-rcskew *value*

(recomputation-skew)
Route recomputation period skew (in precomputed routing).

-recomp *value*

(recomputation-interval)
Route recomputation period (in precomputed routing).

-reextsig

(reextract-sig-failure)
Reextract a new route and try again on set-up failure.

-refresh *value*

(refresh-interval)
Link-state update refresh period.

-requests *value*

(min-requests)
Minimum number of flow requests in simulation.

-rgalpha *value*

(rand-graph-alpha)
Random graph model parameter in Waxman's RG2 model. Controls the number of edges.

-rgasym

(rand-graph-asymmetric)
Use asymmetric random graphs (usually not used).

-rgbeta *value*

(rand-graph-beta)
Random graph model parameter in Waxman's RG2 model. Controls the ratio of long to short edges.

-rgmethod [RG2 | PURE]

(rand-graph-method)
Random graph generation method: RG2 is Waxman's model. PURE is a simple coin flip to decide an edge using rand-graph-alpha value as the probability of having an edge.

-rgplane *value*

(rand-graph-planesize)
Random graph parameter for Euclidean plane dimension – used in Waxman's RG2 model.

-route [ON_DEMAND | PRECOMP]

(routing-policy)
Routing policy: on-demand (ON_DEMAND) or precomputed (PRECOMP)

-rskew *value*

(refresh-skew)
Link update refresh period skew.

-seedval *value*

(random-seed)
Random number generator seed (should be a long int).

-simtol *value*

(sim-tolerance)
Simulation confidence tolerance.

-spread *value*

(bandwidth-spread)
Width of bandwidth distribution (proportional to the mean value).

-timeofday [MORNING | AFTERNOON | EVENING]

(time-of-day)
Time of day: MORNING, AFTERNOON, EVENING. Used with the ARR_FILE option.

-tmultiplier *value*

(traffic-multiplier)
Scaling factor for traffic matrix arrival distribution scale parameters.

-topology [kncube | file | full | rand]

(topology-type)
Type of topology to use: k-ary n-cube (kncube), user file specified (file), fully-connected (full), or random graph (rand). Note lowercase.

-topsize *value*

(topology-size)
Number of nodes in fully-connected topology.

-trigger *value*

(update-trigger)
Link-state update trigger threshhold.

-trigroute

(trigger-recomp-routefail)
Trigger recomputation on routing failure.

-trigsig

(trigger-recomp-sigfail)
Trigger recomputation on set-up failure.

-uniqpolicy [SRCHASH | RANDOM]

(uniq-route-policy)
Policy to use with UNIQ multi-route-policy option. SRCHASH picks a route based on source node id. RANDOM picks randomly among multiple routes.

-version

(print-version)
Print the routesim version and exit

-warmup *value*

(min-warmup)
Minimum warmup-phase flow requests

-warmupprop *value*

(warmup-proportion)
Minimum proportion of total requests that must be warmup-phase.

-warmuptol *value*

(warmup-tolerance)
Confidence interval tolerance for ending warmup phase.

-wsmaxhops *value*

(wide-short-maxhops)
Maximum allowed hopcount (for WIDE_SHORT alg)

-conf *configuration file*

(configuration-file)
Name of configuration file

EXIT STATUS

routesim returns 0 upon successful completion, and 1 otherwise.

FILES

rstopology.dat
rsmonitor.dat
rstraffic.dat
holdcdf.dat

SEE ALSO

rsfiles(l)

DIAGNOSTICS

routesim has extensive support for tracing, with different trace levels that offer progressively more debugging and monitoring information. Changing the trace level, however, requires recompilation of the simulator. The default trace level, RS_TRACE_ERRORS, prints information on standard error only for error conditions.

BUGS

routesim has a ridiculous number of parameters.

routesim does not do enough sanity checking of options at startup.

There are probably some other bugs.

AUTHOR

Anees Shaikh (University of Michigan, AT&T Labs–Research, 1997-99)

NAME

rsfiles - Description of *routesim* simulator file formats

SYNOPSIS

rstopology.dat
rstopology.dat.out
rstraffic.dat
holdcdf.dat
rsmonitor.dat
rslinkmonitor.log
routesim.log
configuration file

DESCRIPTION

rstopology.dat

The rstopology file describes a arbitrary network topology for use in a **routesim** simulation. Each line of the topology file specifies a network link, and all links are unidirectional. The format of a topology file is as follows:

```
num-nodes
from to capacity propdelay adminweight maxcalls
⋮
```

Items are separated by a space. The first line in the file is an integer indicating the number of nodes in the network. Node ids are in the range 0 to `num-nodes - 1`. Each subsequent line defines a link and has 6 fields. `from` and `to` specify the nodes the link connects. The third, fourth, and fifth fields are floating point numbers. Link capacity is specified in the third field. Propagation delay is specified by the fourth field. An arbitrary administrative weight is in the fifth field. The final field is the maximum number flows that can be supported on the link (integer). Each field, except `from` and `to` may be set to -1.0 (or -1 for `maxcalls`) to get assigned the default value.

rstopology.dat.out

This file is dumped when the **-dumptop** or **dump-topology** options are specified to *routesim*. It follows the format described for **rstopology.dat** above.

rstraffic.dat

The rstraffic file specifies a *routesim* traffic matrix. The traffic matrix specifies the arrival rate between each source-destination pair. It also allows different values for different times-of-day. The arrival rate is expressed in terms of a scale value and shape value that serve as parameters to the flow arrival distribution. The possible distributions are Poisson and Weibull, and is specified in a *routesim* configuration file or on the command line. In the case of Poisson, only the scale value is significant, and it represents the mean value. For the Weibull distribution, both the scale and shape values are significant. Together they determine the mean of the distribution. The file format is:

from to mscale mshape ascale ashape escale eshape
⋮

Each line describes a traffic rate from the node *from* to the node *to.* The *mscale*, *ascale*, and *escale* are floating point values that specify the scale parameters for morning, afternoon, and evening, respectively. *mshape*, *ashape*, and *eshape* are the corresponding shape parameters.

Note: scales are the arrival rates – NOT interarrival times.

holdcdf.dat

The holdcdf file specifies an inverse cumulative distribution function for the flow durations. The file format is:

num-entries
u x_first
⋮
1 *x_last*

The first line specifies the number of entries that follows. Each subsequent line is a single (u, x) entry, in ascending u order. The first entry should NOT be for $u = 0$. But the last entry must be for $u = 1$. See examples below for further explanation.

rsmonitor.dat

The rsmonitor file specifies the network links that should be monitored during simulaton. It's format is simply:

from to
⋮

Each line specifies the node ids for the link endpoints. Typically only a few links (maybe just one) are specified since the monitoring information is copious.

rslinkmonitor.log

The rslinkmonitor file is the log of events on the links specified in **rsmonitor.dat**

routesim.log

This file logs the progress of the *routesim* simulation. It lists the phase (warmup or actual sim), current simulation time, number of flow requests, and confidence interval for the stopping metric.

configuration file

The configuration file contains *routesim* options, one per line, in long format. It may also contain comment lines with `/*...*/` delimiters. Note that there must be a space between the '`*`' character and the comment text.

EXAMPLES

rstopology.dat

```
16
0 1 1.000000 0.000000 0.000000 2147483647
0 3 1.000000 0.000000 0.000000 2147483647
0 4 1.000000 0.000000 0.000000 2147483647
0 12 1.000000 0.000000 0.000000 2147483647
1 0 1.000000 0.000000 0.000000 2147483647
1 2 1.000000 0.000000 0.000000 2147483647
1 5 1.000000 0.000000 0.000000 2147483647
⋮
```

rstraffic.dat

```
0 1 0.150000 1.000000 0.150000 1.000000 0.150000 1.000000
0 2 0.113208 1.000000 0.113208 1.000000 0.113208 1.000000
0 3 0.150000 1.000000 0.150000 1.000000 0.150000 1.000000
0 4 0.113208 1.000000 0.113208 1.000000 0.113208 1.000000
0 5 0.113208 1.000000 0.113208 1.000000 0.113208 1.000000
⋮
2 0 0.113208 1.000000 0.113208 1.000000 0.113208 1.000000
2 1 0.150000 1.000000 0.150000 1.000000 0.150000 1.000000
2 3 0.150000 1.000000 0.150000 1.000000 0.150000 1.000000
2 4 0.113208 1.000000 0.113208 1.000000 0.113208 1.000000
2 5 0.113208 1.000000 0.113208 1.000000 0.113208 1.000000
2 6 0.113208 1.000000 0.113208 1.000000 0.113208 1.000000
2 7 0.113208 1.000000 0.113208 1.000000 0.113208 1.000000
⋮
```

holdcdf.dat

Consider the following CDF:

$$F(x) = \begin{cases} 0.0 & 0 \leq x < 64 \\ 0.7 & 64 \leq x < 128 \\ 0.8 & 128 \leq x < 256 \\ 0.9 & 256 \leq x < 512 \\ 1.0 & 512 \leq x \end{cases}$$

The corresponding inverse CDF is:

$$F^{-1}(x) = \begin{cases} 64 & 0 < u \leq 0.7 \\ 128 & 0.7 < u \leq 0.8 \\ 256 & 0.8 < u \leq 0.9 \\ 512 & 0.9 < u \leq 1.0 \end{cases}$$

and the corresponding holdcdf.dat file is:

```
4
0.7 64
0.8 128
0.9 256
1.0 512
```

routesim.log

A typical log file looks like:

```
Warming up ...
847500 (0.0022 [0.0021, 0.0024]) (overutil) Time: 725.72
Warmup complete: 848011 calls
2655000 (0.0566 [0.0563, 0.0568]) (overutil) Time: 2998.96
Simulation complete.  Dumping stats ...
```

configuration file

Sample configuration file:

```
topology-type file
network-model SINGLE
arrival-spec ARR_UNIFORM
arrival-dist POISSON
arrival-scale .94118
arrival-shape 1.00
traffic-multiplier 1.0
time-of-day AFTERNOON
holding-time-spec USERDEF
holding-time-shape 2.5
holding-time-scale 1.00
mean-bandwidth 0.030
bandwidth-spread 1.99000
update-trigger 100000.0000
refresh-interval 120.000000
refresh-skew 0.02
min-update-interval 0.000000
min-requests 800000
min-warmup 500000
min-sim-time 3000.0
warmup-proportion 0.25
blocking-policy NON_BLOCKING
routing-policy ON_DEMAND
routing-algorithm WIDE_SHORT
distance-function HOPCOUNT
hopcount-threshold 1
wide-short-maxhops 10
alternate-routing NO_ALT
multi-route-policy UNIQ
uniq-route-policy RANDOM
max-sig-attempts 1
prune-policy NO_PRUNE
link-state-accuracy STALE
random-seed 1013553468
confidence-level 99
sim-tolerance .05
warmup-tolerance .075
```

SEE ALSO

routesim(l)

AUTHOR

Anees Shaikh (University of Michigan, AT&T Labs–Research, 1997-99)

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] H. Ahmadi, J. S. Chen, and R. Guerin, "Dynamic routing and call control in high-speed integrated networks," in *Teletraffic and Datatraffic in a Period of Change: Proceedings of the International Teletraffic Congress*, volume 14 of *Studies in Telecommunication*, pp. 397–403, North-Holland, June 1991.

[2] A. Alles, *ATM Internetworking*, May 1996. Document available at http://cell-relay.indiana.edu/cell-relay/docs/cisco.html.

[3] G. Apostolopoulos, R. Guerin, S. Kamat, A. Orda, T. Przygienda, and D. Williams. *QoS Routing Mechanisms and OSPF Extensions*. Internet Draft (draft-guerin-qos-routing-ospf-05.txt), work in progress, April 1999.

[4] G. Apostolopoulos, R. Guerin, and S. Kamat, "Implementation and performance measurements of QoS routing extensions to OSPF," in *Proceedings of IEEE INFOCOM*, New York, NY, March 1999.

[5] G. Apostolopoulos, R. Guerin, S. Kamat, and S. Tripathi, "Quality of service based routing: A performance perspective," in *Proceedings of ACM SIGCOMM*, September 1998.

[6] G. Apostolopoulos and S. K. Tripathi, "On the effectiveness of path pre-computation in reducing the processing cost of on-demand QoS path computation," in *Proceedings of IEEE Symposium on Computers and Communication*, June 1998.

[7] C. M. Aras, J. F. Kurose, D. S. Reeves, and H. Schulzrinne, "Real-time communication in packet-switched networks," *Proceedings of the IEEE*, vol. 82, no. 1, pp. 122–139, January 1994.

[8] ATM Forum MPOA Sub-Working Group, *Multi-Protocol over ATM Version 1.0 (AF-MPOA-0087.000)*, July 1997.

[9] D. O. Awduche, J. Malcolm, J. Agogbua, M. O'Dell, and J. McManus. *Requirements for Traffic Engineering Over MPLS*. Internet Draft (draft-ietf-mpls-traffic-eng-00.txt), work in progress, October 1998.

[10] J. J. Bae and T. Suda, "Survey of traffic control schemes and protocols in ATM networks," *Proceedings of the IEEE*, vol. 79, no. 2, pp. 170–189, February 1991.

[11] T. Ballardie, P. Francis, and J. Crowcroft, "Core based trees: An architecture for scalable inter-domain multicast routing," in *Proceedings of ACM SIGCOMM*, pp. 85–95, Ithaca, New York, September 1993.

[12] S. Blake, D. L. Black, M. A. Carlson, E. Davies, Z. Wang, and W. Weiss. *An Architecture for Differentiated Services*. Internet Request for Comments (RFC 2475), December 1998.

[13] L. Breslau, D. Estrin, and L. Zhang, "A simulation study of adaptive source routing in integrated services networks," Technical Report 93-551, Computer Science Department, University of Southern California, 1993.

[14] H. Che and S.-Q. Li, "MPOA flow classification design and analysis," in *Proceedings of IEEE INFOCOM*, March 1999.

[15] S. Chen and K. Nahrstedt, "Distributed QoS routing with imprecise state information," in *Proceedings of IEEE International Conference on Computer Communications and Networks*, Lafayette, LA, October 1998.

[16] S. Chen and K. Nahrstedt, "Distributed quality-of-service routing in high-speed networks based on selective probing," in *Proceedings of IEEE Conference on Local Computer Networks*, pp. 80–89, Boston, MA, October 1998.

[17] S. Chen and K. Nahrstedt, "On finding multi-constrained paths," in *Proceedings of IEEE International Conference on Communications*, pp. 874–879, Atlanta, GA, June 1998.

[18] S. Chen and K. Nahrstedt, "An overview of quality of service routing for next-generation high-speed networks: Problems and solutions," *IEEE Network Magazine*, pp. 64–79, November/December 1998.

[19] B. V. Cherkassky, A. V. Goldberg, and T. Radzik, "Shortest-path algorithms: Theory and experimental evaluation," *Mathematical Programming*, vol. 73, no. 2, pp. 129–174, May 1996.

[20] K. C. Claffy, H.-W. Braun, and G. C. Polyzos, "A parameterizable methodology for Internet traffic flow profiling," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 8, pp. 1481–1494, October 1995.

[21] D. D. Clark, S. Shenker, and L. Zhang, "Supporting real-time applications in an integrated services packet network: Architecture and mechanism," in *Proceedings of ACM SIGCOMM*, pp. 77–82, Baltimore, MD, August 1992.

[22] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press (McGraw-Hill), Cambridge, MA (New York), 1990.

[23] E. Crawley, R. Nair, B. Rajagopalan, and H. Sandick. *A Framework for QoS-based Routing in the Internet*. Internet Request for Comments (RFC 2386), August 1998.

[24] M. E. Crovella and A. Bestavros, "Self-similarity in world wide web traffic: Evidence and possible causes," *IEEE/ACM Transactions on Networking*, vol. 5, no. 6, pp. 835–846, December 1997.

[25] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C.-G. Liu, and L. Wei, "An architecture for wide-area multicast routing," in *Proceedings of ACM SIGCOMM*, pp. 126–135, London, UK, August 1994.

[26] M. Degermark, A. Brodnik, S. Carlsson, and S. Pink, "Small forwarding tables for fast routing lookups," in *Proceedings of ACM SIGCOMM*, pp. 3–14, Cannes, France, September 1997.

[27] A. Feldmann, "Impact of non-poisson arrival sequences for call admision algorithms with and without delay," in *Proceedings of IEEE GLOBECOM*, pp. 617–622, November 1996.

[28] A. Feldmann, J. Rexford, and R. Cáceres, "Efficient policies for carrying Web traffic over flow-switched networks," *IEEE/ACM Transactions on Networking*, pp. 673–685, December 1998.

[29] P. Ferguson and G. Huston, *Quality-of-Service: Delivering QoS on the Internet and in Corporate Networks*, John Wiley & Sons, Inc., New York, NY, 1998.

[30] S. Floyd and V. Jacobson, "Synchronization of periodic routing messages," *IEEE/ACM Transactions on Networking*, vol. 2, no. 2, pp. 122–136, April 1994.

[31] V. Fuller, T. Li, J. Yu, and K. Varadhan. *Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy*. Internet Request for Comments (RFC 1519), September 1993.

[32] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Co., New York, 1979.

[33] R. Gawlick, C. Kalmanek, and K. Ramakrishnan, "Online routing for virtual private networks," *Computer Communications*, vol. 19, no. 3, pp. 235–244, March 1996.

[34] R. Gawlick, A. Kamath, S. Plotkin, and K. Ramakrishnan, "Routing and admission control in general topology networks," Technical Report CS-TR-95-1548, Stanford University, May 1995.

[35] E. Gelenbe, S. Kotia, and D. Krauss, "Call establishment overload in ATM networks," *Performance Evaluation*, 1997.

[36] R. J. Gibbens, P. J. Hunt, and F. P. Kelly, "Bistability in communication networks," *Disorder in Physical Systems*, 1990.

[37] A. G. Greenberg and R. Srikant, "Computational tecniques for accurate performance evaluation of multirate, multihop communication networks," *IEEE/ACM Transactions on Networking*, vol. 5, no. 2, pp. 266–277, April 1997.

[38] R. Guerin and A. Orda, "QoS-based routing in networks with inaccurate information," in *Proceedings of IEEE INFOCOM*, April 1997.

[39] R. Guerin, A. Orda, and D. Williams, "QoS routing mechanisms and OSPF extensions," in *Proc. Global Internet Miniconference*, November 1997.

[40] B. Halabi, *Internet Routing Architectures*, Cisco Press, Indianapolis, IN, 1997.

[41] F. Hao and E. W. Zegura, "Scalability techniques in QoS routing," Technical Report GIT-CC-99-16, College of Computing, Georgia Institute of Technology, Atlanta, GA, 1999.

[42] M. Harchol-Balter and A. Downey, "Exploiting process lifetime distributions for dynamic load balancing," *ACM Transactions on Computer Systems*, vol. 15, no. 3, pp. 253–285, August 1997.

[43] C. Hedrick. *Routing Information Protocol*. Internet Request of Comments (RFC 1058), June 1988.

[44] R.-H. Hwang, J. Kurose, and D. Towsley, "On call processing delay in high speed networks," *IEEE/ACM Transactions on Networking*, vol. 3, no. 6, pp. 628–639, December 1995.

[45] Internet Performance and Measurement Analysis project. *IPMA project homepage*, March 1999. `http://www.merit.net/ipma`.

[46] ISO/IEC 10589, *Information technology–Telecommunications and information exchange between systems–Intermediate system to intermediate system intra-domain routeing information exchange protocol for use in conjunction with the protocol for providing the connectionless mode Network Service (ISO 8473*, International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC), April 1992.

[47] A. Iwata, R. Izmailov, H. Suzuki, and B. Sengupta, "PNNI routing algorithms for multimedia ATM internet," *NEC Research & Development*, vol. 38, no. 1, pp. 60–73, January 1997.

[48] Y. Katsube, K. Nagami, S. Matsuzawa, and H. Esaki, "Internetworking based on cell switch router - architecture and protocol overview," *Proceedings of the IEEE*, vol. 85, no. 12, pp. 1998–2006, December 1997.

[49] A. Khanna and J. Zinky, "The revised ARPANET routing metric," in *Proceedings of ACM SIGCOMM*, pp. 45–56, September 1989.

[50] S.-K. Kweon and K. G. Shin, "Distributed QoS routing using bounded flooding," Technical Report CSE-TR-388-99, Computer Science and Engineering, Dept. of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, March 1999.

[51] T. Lakshman and D. Stiliadis, "High speed policy-based packet forwarding using efficient multi-dimensional range matching," in *Proceedings of ACM SIGCOMM*, September 1998.

[52] E. L. Lawler, "A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem," *Management Science*, vol. 18, no. 7, pp. 401–405, March 1972.

[53] J.-Y. Le Boudec and T. Przygienda, "A route pre-computation algorithm for integrated services networks," *Journal of Network and Systems Management*, vol. 3, no. 4, pp. 427–449, 1995.

[54] W. C. Lee, M. G. Hluchyj, and P. A. Humblet, "Routing subject to quality of service constraints in integrated communication networks," *IEEE Network Magazine*, pp. 46–55, July/August 1995.

[55] S. Lin and N. McKeown, "A simulation study of IP switching," in *Proceedings of ACM SIGCOMM*, pp. 15–24, September 1997.

[56] Q. Ma and P. Steenkiste, "On path selection for traffic with bandwidth guarantees," in *Proceedings of IEEE International Conference on Network Protocols*, Atlanta, GA, October 1997.

[57] Q. Ma and P. Steenkiste, "Quality-of-service routing for traffic with performance guarantees," in *Proc. IFIP International Workshop on Quality of Service*, pp. 115–126, Columbia University, New York, May 1997.

[58] G. Malkin. *RIP version 2*. Internet Request for Comments (RFC 2453), November 1998.

[59] I. Matta and A. U. Shankar, "Dynamic routing of real-time virtual circuits," in *Proceedings of IEEE International Conference on Network Protocols*, pp. 132–139, Columbus, OH, 1996.

[60] N. F. Maxemchuk and M. El Zarki, "Routing and flow control in high-speed wide-area networks," *Proceedings of the IEEE*, vol. 78, no. 1, pp. 204–221, January 1990.

[61] J. M. McQuillan, G. Falk, and I. Richer, "A review of the development and performance of the ARPANET routing algorithm," *IEEE Transactions on Communications*, vol. 26, no. 12, pp. 1802–1811, December 1978.

[62] J. M. McQuillan, I. Richer, and E. C. Rosen, "An overview of the new routing algorithm for the ARPANET," in *Proc. Data Communications Symposium*, pp. 63–68, November 1979.

[63] D. J. Mitzel, D. Estrin, S. Shenker, and L. Zhang, "A study of reservation dynamics in integrated services packet networks," in *Proceedings of IEEE INFOCOM*, pp. 871–879, April 1996.

[64] J. Moy. *OSPF Version 2*. Internet Request for Comments (RFC 2328), April 1998.

[65] J. T. Moy, *OSPF: Anatomy of an Internet Routing Protocol*, Addison-Wesley, Reading, MA, 1998.

[66] P. Newman, G. Minshall, and T. Lyon, "IP switching: ATM under IP," *IEEE/ACM Transactions on Networking*, vol. 6, no. 2, pp. 117–129, April 1998.

[67] V. Paxson and S. Floyd, "Why we don't know how to simulate the Internet," in *Proceedings of the Winter Simulation Conference*, Atlanta, GA, December 1997.

[68] M. Peyravian and A. D. Kshemkalyani, "Network path caching: Issues, algorithms and a simulation study," *Computer Communications*, vol. 20, pp. 605–614, 1997.

[69] M. Peyravian and R. Onvural, "Algorithm for efficient generation of link-state updates in ATM networks," *Computer Networks and ISDN Systems*, vol. 29, no. 2, pp. 237–247, January 1997.

[70] S. Plotkin, "Competitive routing of virtual circuits in ATM networks," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 6, pp. 1128–1136, August 1995.

[71] PNNI Specification Working Group, *Private Network-Network Interface Specification Version 1.0*, ATM Forum, March 1996.

[72] C. Pornavalai, G. Chakraborty, and N. Shiratori, "QoS based routing in integrated services packet networks," in *Proceedings of IEEE International Conference on Network Protocols*, Atlanta, GA, October 1997.

[73] S. Rampal and D. Reeves, "Routing and admission control algorithms for multimedia data," *Computer Communications*, October 1995.

[74] Y. Rekhter and T. Li. *A Border Gateway Protocol 4 (BGP-4).* Internet Request for Comments (RFC 1771), March 1995.

[75] H. Rudin, "On routing and 'delta routing': A taxonomy and performance comparison of techniques for packet-switched networks," *IEEE Transactions on Communications*, vol. 24, no. 1, pp. 43–59, January 1976.

[76] A. Shaikh, J. Rexford, and K. Shin, "Dynamics of quality-of-service routing with inaccurate link-state information," Technical Report CSE-TR-350-97, Computer Science and Engineering Division, University of Michigan, Ann Arbor, MI, November 1997.

[77] A. Shaikh, J. Rexford, and K. Shin, "Efficient precomputation of quality-of-service routes," in *Proceedings of Workshop on Network and Operating Systems Support for Digital Audio and Video*, July 1998.

[78] S. Shenker, C. Partridge, and R. Guerin. *Specification of Guaranteed Quality of Service*. Internet Request for Comments (RFC 2212), September 1997.

[79] K.-Y. Siu and R. Jain, "A brief overview of ATM: Protocol layers, LAN emulation and traffic management," *ACM Computer Communication Review*, vol. 25, no. 2, pp. 6–20, April 1995.

[80] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, "Fast scalable algorithms for level four switching," in *Proceedings of ACM SIGCOMM*, September 1998.

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

[82] K. Thompson, G. J. Miller, and R. Wilder, "Wide-area internet traffic patterns and characteristics," *IEEE Network Magazine*, vol. 11, no. 6, pp. 10–23, November/December 1997.

[83] D. Towsley, "Providing quality of service in packet switched networks," in *Performance Evaluation of Computer and Communication Systems*, pp. 560–586, Springer Verlag, 1993.

[84] C. Villamizar. *OSPF Optimized Multipath (OSPF-OMP)*. Internet Draft (draft-ietf-ospf-omp-01), work in progress, October 1998.

[85] Virtual InterNetwork Testbed project, *VINT project homepage*, March 1999. `http://netweb.usc.edu/vint`.

[86] A. Viswanathan, N. Feldman, Z. Wang, and R. Callon, "Evolution of multiprotocol label switching," *IEEE Communications Magazine*, vol. 36, no. 5, pp. 165–173, May 1998.

[87] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, "Scalable high speed IP routing lookups," in *Proceedings of ACM SIGCOMM*, pp. 25–36, Cannes, France, September 1997.

[88] Z. Wang and J. Crowcroft, "Analysis of shortest-path routing algorithms in a dynamic network environment," *ACM Computer Communication Review*, vol. 22, no. 2, pp. 63–71, April 1992.

[89] Z. Wang and J. Crowcroft, "Quality-of-service routing for supporting multimedia applications," *IEEE Journal on Selected Areas in Communications*, vol. 14, no. 7, pp. 1228–1234, September 1996.

[90] B. M. Waxman, "Routing of multipoint connections," *IEEE Journal on Selected Areas in Communications*, vol. 6, no. 9, pp. 1617–1622, December 1988.

[91] I. Widjaja, H. Wang, S. Wright, and A. Chatterjee, "Scalability evaluation of multi-protocol over ATM (MPOA)," in *Proceedings of IEEE INFOCOM*, March 1999.

[92] J. Wroclawski. *Specification of the Controlled-Load Network Element Service*. Internet Request for Comments (RFC 2211), September 1997.

[93] J. Y. Yen, "Finding the k shortest loopless paths in a network," *Management Science*, vol. 17, no. 11, pp. 712–716, July 1971.

[94] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee, "How to model an internetwork," in *Proceedings of IEEE INFOCOM*, pp. 594–602, March 1996.

[95] H. Zhang, "Service disciplines for guaranteed performance service in packet-switching networks," *Proceedings of the IEEE*, vol. 83, no. 10, pp. 1374–1396, October 1995.

[96] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: A new resource reservation protocol," *IEEE Network Magazine*, pp. 8–18, September 1993.

[97] Z. Zhang, C. Sanchez, B. Salkewicz, and E. S. Crawley. *Quality of Service Extensions to OSPF or Quality of Service Path First Routing (QOSPF).* Internet Draft (draft-zhang-qos-ospf-01.txt), work in progress, September 1997.

[98] W. Zhao and S. K. Tripathi, "Routing guaranteed quality-of-service connections in integrated services packet networks," in *Proceedings of IEEE International Conference on Network Protocols*, pp. 175–182, Atlanta, GA, 1997.