Reliable Routing and its Applications in MPLS and Admission Control

Jian Pu

B.Sc. (Physics), SiChuan University, ChengDu, China, 1987
M.Sc. (Physics), University of Science and Technology of China, HeFei, 1990
M.Sc.(Computer Science), Carleton University, Ottawa, 1999

A Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in the Department of Computer Science

We accept this dissertation as conforming
to the required standard

Dr. G. C. Shoja, Supervisor (Department of Computer Science)

Dr. E. G. Manning, Supervisor (Department of Electrical & Computer Engineering)

Dr. W. Myrvold, Departmental Member (Department of Computer Science)

Dr. F. Gebali, Outside Member (Department of Electrical & Computer Engineering)

Dr. S. T. Vuong, External Examiner (Department of Computer Science, University of British Columbia)

Supervisors: Dr. G. C. Shoja and Dr. E. G. Manning

ABSTRACT

Reliable routing using alternate paths is investigated in this dissertation. We propose precalculated alternate paths as a method for fast recovery from link and node failures in IP networks. We demonstrate that path switching time, and thus failure recovery time are, as expected, considerably faster than the standard method of recalculating a new path on the fly. However, to be effective, the alternate paths should share a minimal set of links and nodes - preferably none - with the failed path. As shared links are considered in this work, we give a reliability model for this situation (non-disjoint alternate paths) and develop estimates of reliability as a function of the number of shared links. Alternate path finding algorithms to calculate suitable alternate paths subject to predefined constraints are also developed.

Implementation of these techniques for improving routing reliability is shown to be straightforward for explicit routing protocols such as Multi-Protocol Label Switching (MPLS) with Explicit Routing mode. This mode is expected to be the protocol of choice for applications requiring guaranteed Quality of Service (QoS) carried on the coming generation of wavelength-switched networks (Internet II, CA Net III, etc.) We propose a Reliable MPLS (R-MPLS) protocol by applying alternate path routing to MPLS, using our new algorithms to precalculate appropriate alternate paths. Simulation results show that R-MPLS can achieve fast recovery from failures.

We also address reliability issues for the problem of optimal Service Level Agreement (SLA) admission control. To achieve reliable admission control, we apply alternate path routing to an existing SLA-based admission controller called SLAOpt. In the existing Utility Model, SLA admission control is mapped to the Multiple-Choice Multi-Dimension Knapsack Problem (MMKP), where the aim is to maximize system utility (i.e., revenue). However, SLAOpt is static in terms of network topology and does not consider reliability. Motivated by this, we propose a Reliable SLAOpt (R-SLAOpt),

in which utility optimization is subject to the additional constraint of reliability. A new algorithm was also developed to calculate multiple groups of alternate paths that meet the desired QoS demands and reliability requirement. After QoS adaptation, R-SLAOpt selects an appropriate path group containing two or three paths for each admitted session and performs resource reservation on all paths in the group. In the event of node or link failure, a session can be quickly switched to one of the alternate paths, maintaining the guaranteed QoS without having to run the full admission algorithm again. In this way, we have obtained a unified treatment of routing reliability and optimal SLA admission control.

Finally, simulations are presented which investigate R-SLAOpt's impact on system performance and the gains made in reliability.

Examiners:

Dr. G. C. Shoja, Supervisor (Department of Computer Science)

Dr. E. G. Manning, Supervisor (Department of Electrical & Computer Engineering)

Dr. W. Myrvold, Departmental Member (Department of Computer Science)

Dr. F. Gebali, Outside Member (Department of Electrical & Computer Engineering)

Dr. S. T. Vuong, External Examiner (Department of Computer Science, University of British Columbia)

Table of Contents

List of Figures

List of Tables

Glossary of Terms

ATM	Asynchronous Transfer Mode
BFS	Breadth First Search
BLD	Backup Load Distribution
CR-LDP	Constraint-based Routing LDP
CR-LSP	Constraint-based Routed LSP
EBAK	Extension to the Bak's algorithm (a proposed alternate path finding algorithm)
EMPS	Extension to the MPS shortest paths algorithm (a proposed alternate path finding algorithm)
EN	Enterprise Network
ER	Explicit Route
EWI	The proposed Edge Weight Increasing alternate path finding algorithm
FEC	Forwarding Equivalence Class
FIR	Full Information Restoration
GAPA	The proposed alternate path finding algorithm to calculate acceptable path groups for R-SLAOpt
GMPLS	Generalized MPLS
H	The candidate paths whose lengths satisfy the end-to-end delay constraint. These *H* paths can be calculated by the *K*-shortest paths algorithms.
HEU	Heuristic for solving the MMKP
I-HEU	Incremental HEU
IP	Internet Protocol
k	The number of acceptable paths calculated for a given node pair*(k = 1, 2, 3)*, which is also called the size of the path group.
K	The all *K* shortest paths for a given node pair, calculated by a *K*-shortest paths algorithm
L&G	Lee and Gerla's alternate path finding algorithm
LDP	Label Distribution Protocol
LER	Label Edge Router
LSA	Link-State Advertisement
LSP	Label Switched Path

LSR	Label Switching Router
M	Number of links in a network
MIR	Minimum Interference Routing
MIRA	Minimum Interference Routing Algorithm
MMKP	Multiple-Choice Multi-Dimension Knapsack Problem
MPLS	Multi-Protocol Label Switching
MPS	A deviation approach to the *K*-shortest paths algorithm, developed by Martins, Pascoal, and Santos
MTTF	Mean Time To Failure
MTTR	Mean Time To Repair
N	Number of nodes in a network
OptAlt	The proposed Optimal Alternate path finding algorithm
OSPF	Open Shortest Path First routing protocol
QoS	Quality of Service
P-cycle	virtual Protection Cycle
QOSPF	QoS extension to OSPF
R	Connect Reliability (represent routing reliability in this dissertation)
$Rel_2(s, t, G)$	two terminal measure of network reliability
Rel(G)	all-terminal measure of network reliability
Rel(s, K, G)	*k*-terminal measure of network reliability
R-MPLS	The proposed Reliable MPLS
R-SLAOpt	The proposed Reliable SLAOpt
RSVP	Resource ReSerVation Protocol
RSVP-TE	RSVP Traffic Engineering Extensions
SLA	Service Level Agreement
SLAOpt	A Java simulation of an SLA-based utility-optimal admission controller. It was developed by Watson.
SPT	Shortest Path Tree
SRLG	Shared Risk Link Group
WDM	Wavelength-Division Multiplexing

Acknowledgement

I would like to express my sincere gratitude to my supervisors Dr. Gholamali C. Shoja and Dr. Eric G. Manning for their guidance and encouragement throughout this work. With their guidance, I could quickly correct my research directions, and overcome the problems I faced during my research work.

I would like to thank Dr. Kin F. Li, who encouraged me to complete the work on R-SLAOpt, and took time from his busy schedules to read this whole dissertation and offer very valuable suggestion. I also would like to thank the members of my dissertation committee Dr. Wendy Myrvold and Dr. Fayez Gebali for their useful advice and comments.

The support from all the members of PANDA lab is also gratefully appreciated. I thank Mostofa Akbar for useful discussion during my research. Eric Gowland carefully reviewed my dissertation chapters with creative comments, and I am especially grateful to him. Steven Shelford also read one chapter of my dissertation and gave me useful comments. I enjoyed co-operating with them.

Special thanks are for my wife, Yunxia Wang. Without her selfless support, this work would not have been possible.

Finally, I would like to thank the Nortel Networks and the Department of Computer Science in the University of Victoria for their financial support.

To my wife (Yunxia Wang),

my father,

and

my sons (Cullen and Gavin)

1. Introduction

Emerging multimedia-based Internet applications require both *Quality of Service* (QoS) guarantees, for traditional QoS demands such as bandwidth and end-to-end delay, and for routing reliability during transmission. This means that new routing protocols with these features must be designed and implemented.

1.1 The Growing Internet

The size of the Internet has been increasing rapidly for two decades. Growing from a predecessor – the ARPANET, built in 1970 to connect a small community of researchers via a few tens of machines – the Internet had about *605* million users in September 2002 [Nua02]. In July 1998, the number of Internet hosts was estimated at *36* million. After several years, in January 2003, the Internet connected more than *171* million hosts in total [Lot03]. The growth of the global Internet has been phenomenal.

The amount of traffic carried by the Internet is growing very rapidly as well. The types of traffic have become diverse too, with dramatic increases in voice, image and video traffic – anticipated for the foreseeable future. These types of traffic require that the underlying network provide higher levels of reliability, security, QoS guarantees, and multicasting support than are available from the present best-effort datagram architecture and protocols. Hence, some of the old Internet protocols will have to be extended, modified or replaced.

The explosive growth in the size and traffic volume of the Internet has not come without a price. Serious performance degradation and scalability problems have been observed during the past decade. These include severe packet loss, insufficient bandwidth and excessively high delay and variance of delay (jitter).

Routing protocols must meet several demands in the current Internet. They must continually update their routing tables to reflect changes of network topology. A failure

can change the best paths to certain destinations. It takes some time for any routing protocol to compute the new best paths, and the paths used in the meantime may be suboptimal or even non-functional. The process of finding the new paths after the network topology changes and switching the traffic from the failed paths onto the new paths is called *convergence* [Moy98a] [*]. The time spent in this process is the so-called *convergence time.* The preferred routing protocols can update the routing table quickly and have short convergence time. To significantly reduce the convergence time is a major consideration in designing a reliable routing protocol. Fast selection and establishment of the new paths are essential.

Alternate path routing is one approach that can reduce the convergence time and speed up recovery from network component failures. In such a routing mechanism, a suitable path group containing a primary path and multiple alternate paths can be precalculated and preestablished for a given source-destination pair. Therefore, the failed paths can be quickly replaced by previously calculated alternates in case of failures. This process is called *path switching.*

1.2 Motivation

Traditionally, calculating new paths based on the new network topology resulting from a failure of a link or switch is the standard way to bypass the failure in a routing protocol. However, this means that the entire routing table has to be rebuilt when a failure occurs. This solution works well when the network is small and stable. However, when the network becomes big and/or unstable, such a solution will be inefficient because of the significant overhead of time costs and system resources involved. In this type of large network, nodes and edges may relatively frequently fail and recover. To adapt to these changes quickly, routing protocols are required to frequently recalculate new paths that do not use the failed nodes or edges, and then deploy the new paths to bypass those failures.

---

[*] Congestion in the network may also trigger a similar process – find a new path and switch some traffic to the new path, so that the congestion is removed. In this dissertation, we concentrate on routing reliability. Congestion control is beyond the scope of our work.

One typical approach to speed up the recalculation of new paths based on the latest network topology is to reuse old path information, i.e., partially rebuild the routing table [Mcq80, Nar99a, Nar00b]. When a failure occurs, the cost of path recalculation will be reduced significantly. However, this approach still cannot eliminate the recalculation overhead for each failure: a certain amount of computation is still unavoidable for partial rebuilding of the routing table.

Reliable routing protocols should tolerate maximal failures of nodes and edges with minimal effect on performance, so that the system reliability is improved. Responses to the failures by using suitable redundancy and adaptation are ways to build reliable and survivable services [Hil03]. Typically, a routing protocol using an alternate path mechanism is a good candidate, because using precalculated alternate paths when failures occur allows us to eliminate the path recalculation at the time of failure [Wan90, Bah92, Seg98]. This idea is the basis of alternate path routing, and was first applied to the public switched voice network more than half a century ago. In alternate path routing, the convergence time can be improved substantially by simply replacing the failed path with a precalculated alternative.

Disjoint paths contain no shared links and nodes among paths for a given source-destination pair and can be used for alternate path routing. Much research has been done on efficient algorithms for finding disjoint paths [Has85, Gol88, Che97]. However, some drawbacks exist when using strictly disjoint paths in routing protocols. In some network topologies, we may not find enough disjoint paths, or such paths may be too long.

Non-disjoint paths (i.e., partially disjoint paths) may also be used for alternate path routing, and this may solve the problems that are encountered in disjoint path routing. Because this approach can tolerate some common links or nodes among paths, more paths may be found and their lengths may be shorter as compared to the approach of disjoint path routing. The *K*-shortest paths algorithms [Yen71, Epp99, Mar99] are candidates to calculate the non-disjoint paths. However, there may be too little variation (i.e., too much overlapping) among the calculated paths. Therefore, directly applying a *K*-shortest paths algorithm to calculate non-disjoint paths is not practical.

The routing convergence time is desired to be reduced. If such time is small in routing protocols, we can speed fault recovery and improve routing reliability. It is now an important research field. In time-critical applications such as remote medical systems or delivery of interactive multimedia streams, reliable data transmission is essential even in an unstable network. To satisfy these demands, alternate path routing is necessary. However, the subject of alternate path routing by using non-disjoint paths has not been adequately studied to date. Also, studies on how the common links affect reliability in alternate path routing have not matured. These motivate our research on improving reliability by using non-disjoint alternate paths.

1.3 Objectives

Improving routing reliability by using precalculated alternate paths is the main consideration of our research. For this purpose, we describe related problems and our objectives in the following subsections.

1.3.1 Seeking Efficient Alternate Path Finding Algorithms

Calculating appropriate alternate paths is essential if alternate path routing is applied to achieve high reliability. We attempt to select multiple disjoint or non-disjoint paths, which meet the predefined constraints on reliability and other QoS demands such as bandwidth and end-to-end delay. Any overlapping of the alternate paths has a major impact on reliability. Hence, common links and nodes among the paths have to be limited. However, we only consider common links in our work.

An alternate path finding algorithm that optimizes reliability and meets the predefined constraints will provide a solution. However, such an algorithm we develop has significant computational overhead. Hence, to efficiently calculate suitable alternate paths for our reliable routing scheme, faster and simpler heuristic algorithms are also necessary and desirable.

The major tasks for seeking new alternate path finding algorithms are:

a). Quantifying routing reliability when certain common links exist between the primary and alternate paths for a given source-destination pair.

b). Developing efficient algorithms to calculate suitable alternate paths (disjoint or non-disjoint) as backups. The constraints of path selection are reliability and expected QoS guarantees on bandwidth and end-to-end delay.

1.3.2 Improving Reliability for MPLS

Multi-Protocol Label Switching (MPLS) [Ros01] is an emerging routing scheme for the coming generation of applications requiring guaranteed QoS. A label-driven forwarding mechanism is used in MPLS, which can easily achieve explicit routing. More analysis of MPLS is represented in Sections 6.1 and 6.2.

Currently, problems associated with the offering of reliable services on MPLS are getting more attention from researchers. An important practical issue for MPLS is the capacity to recover quickly from failures. Our work concentrates on achieving reliable routing for IP networks in the MPLS framework. Based on the proposed alternate path finding algorithms, we will propose a *Reliable MPLS* (R-MPLS). This will improve recovery time by quickly switching a primary path, affected by network component failures, to a precalculated and preestablished alternate path.

1.3.3 Unifying Reliable Routing with Utility-optimal Admission Control

Existing research on optimal *Service Level Agreement* (SLA) admission control [Kha98, Wat01, Akb02a] has aimed to maximize system utility (i.e., revenue) subject to system resource constraints. An SLA, representing a potential session and describing all requirements and features of that session, is a contractual agreement between end-users (or subscribers) and a network provider,

Watson developed an SLA-based utility-optimal admission controller called SLAOpt [Wat01] by using the *Utility Model* proposed by Khan [Kha98]. Based on the Utility Mode, the admission control can be mapped onto an *Multiple-Choice Multi-Dimension*

Knapsack Problem (MMKP) [Kha98, Akb01b]. However, SLAOpt did not consider reliability: it assumed that there were no failures of network components.

A reliable extension to SLAOpt called R-SLAOpt will be proposed in this dissertation, based on alternate path routing and R-MPLS. R-SLAOpt adds a way to deal with reliability requirements of SLAOpt. This work is a unification of reliable routing and performance optimization in admission control. In R-SLAOpt, after path switching, the desired QoS level (typically, bandwidth demand and end-to-end delay) for an admitted SLA will still be guaranteed.

Several tasks are involved to obtain such unification:

a). Using the concept of a *Path Group*, which contains a primary path and acceptable alternate paths for a given source-destination pair. In SLA adaptation, a path group will be associated with a certain QoS level.

b). Developing a new algorithm to select acceptable candidate path groups subject to reliability requirements and QoS guarantees.

c). Extending the existing process of QoS adaptation (i.e., the process of solving the MMKP for admission control in SLAOpt by upgrading and downgrading QoS levels) to accommodate incoming SLA requests among the candidate path groups. When an SLA is admitted, a path group will be associated with it. This group, called the working path group, satisfies the QoS guarantees and reliability requirements specified by the SLA. The primary path in the working group will be used as the working path; the others are the alternate paths.

d). Switching to an available alternate path in the working path group to reduce the convergence time when the working path is affected by a failure. All QoS service levels provided by the alternate path are kept unchanged after path switching.

1.4 Main Contributions

We introduce a concept called *Connection Reliability* for quantifying the routing reliability when multiple paths with common edges exist for a given source-destination pair. For a homogeneous network, mathematical formulae are derived to calculate the connection reliability. Then, we develop four alternate path finding algorithms to calculate paths with common edges subject to QoS guarantees and reliability requirements. These algorithms have different features, which may satisfy diverse demands of network applications in a core network with limited size.

We propose a reliable version of MPLS with the Explicit Routing mode by applying our new algorithms and using path switching at sources. Furthermore, we study an existing static admission control model and consider network component failures in admission control. Based on our work in routing reliability and the alternate path routing mechanism, we develop a new admission control model with reliability guarantees.

1.5 Dissertation Organization

The rest of this dissertation is organized as follows. Chapter 2 provides background information on the routing protocols, existing work on MPLS reliability and related work on SLA-based admission control.

In Chapter 3, the alternate path routing mechanism is discussed. Several aspects are addressed: how and when to calculate alternate paths, how to dynamically use the alternate paths, and what the main features of alternate path routing are.

In Chapter 4, definitions of network reliability are presented. The preferred number of alternate paths is discussed by considering the tradeoff between reliability and resource assumption. Furthermore, by analyzing the types of path overlaps, mathematical formulae are derived to calculate the connection reliability.

In Chapter 5, new alternate path finding algorithms are developed by considering reliability (i.e., connection reliability in this context) and QoS demands. A reliability-optimal algorithm is developed at first, then three heuristic algorithms are proposed to

efficiently calculate suitable alternate paths. Simulations are performed to validate these new algorithms.

In Chapter 6, we carefully analyze the label-driven explicit routing mechanism in MPLS, and apply our new alternate path finding algorithms to MPLS. By extending MPLS, a new Reliable MPLS (R-MPLS) is introduced. Related simulation tests show that reliability in R-MPLS can be significantly improved.

In Chapter 7, we propose a new model for an SLA-based admission controller by adding reliability to the existing SLAOpt. The new controller, called Reliable SLAOpt (R-SLAOpt), is based on alternate path routing and R-MPLS. The new model combines routing reliability with utility-optimal admission control. Simulations illustrate that R-SLAOpt has much higher reliability than SLAOpt does, and the alternate paths can be quickly selected and activated when network component failures occur.

Chapter 8 concludes this dissertation. We discuss the work we have presented, our major contributions, and directions for future research.

2. Background

Useful background information related to our work on routing protocols, routing reliability and admission control is described in this chapter.

2.1 Network Model

A network is modelled as an undirected graph, *G(V, E),* where *V* is the set of nodes and *E* is the set of edges. Typically, we use *N* to represent the number of nodes and *M* to represent the number of edges in the graph. Routers or switches in the network are modelled as nodes of the graph, and links (transmission facilities) between nodes are modelled as edges with associated weights. A node or an edge in the network is called a *network component.* In this dissertation, the terms "link" and "edge" are interchangeable, as are the terms "router", "switch", "node" and "vertex". We also use the terms "length" and "weight" interchangeably in relation to an edge. Denoted by a positive number, the edge weight may be used to convey various meanings such as distance, cost, delay, load, or failure probability.

A path P_{st} from a source node *s* to a destination node *t* is a sequence of edges starting at *s* and ending at *t.* All nodes on path P_{st} except *s* and *t* are called *internal nodes* or *intermediate nodes.* The *length of path* P_{st} is the sum of the lengths of all the edges on P_{st}. We use the terms "path" and "route" interchangeably. In this dissertation, edge length denotes link delay and path length then means end-to-end delay, as we ignore node delays. The *number of hops* of path P_{st} is the total number of edges on P_{st}. A *shortest path* from *s* to *t* is A path with minimum path length from the set of all possible paths from *s* to *t.* In some cases, there are multiple equal-length shortest paths between these two nodes.

A pair of paths from *s* to *t* is *edge-disjoint* if they have no edge in common; they are *node-disjoint* if they have no node in common (except the source and the destination

nodes). The terms *"common edge"* and *"shared edge"* are used interchangeably here, as are the terms *"common node"* and *"shared node"*. The term *"disjoint paths"* refers to paths with no shared edges and no shared nodes. The term *"non-disjoint paths"* refers to paths with some shared edges or shared nodes. It has the same meaning as the term *"partially disjoint paths"*. In this dissertation, we permit shared edges in our proposed new routing algorithms. *"Double-shared edges"* (DSEs) are edges shared by two paths, while *"triple-shared edges"* (TSEs) are edges shared by three paths. Typically, both types of shared edges may exist among multiple paths between s and t; they tend to degrade reliability, as the failure of a shared edge can affect all of the paths which share that edge.

A graph G may have several disjoint paths from node s to node t. If there are n such paths, then the *connectivity* of pair (s, t) is n. Obviously, increasing the connectivity of each node pair tends to improve G's reliability, but the cost of building the network represented by G will also increase.

It is common that network component failures may occur due to hardware faults or software bugs. In this case, a link or node becomes non-operational. We represent such a non-operational component as a *"failed link"* or *"failed node"*, and say that it has suffered a failure. For simplicity, we often use the term *"failures"* to represent the non-operational components, including failed links and failed nodes. A non-operational component may also be called a *"faulty link"* or *"faulty node"*. Obviously, a non-operational component can cause a path that contains this component to fail. Such a path is called a *"failed path"* or an *"affected path"*.

2.2 Overview of Routing Protocols

Routing algorithms can be grouped into two major classes: adaptive and non-adaptive. Adaptive algorithms adjust the paths for packet delivery according to changes in network topology or congestion; non-adaptive algorithms do neither. Static routing algorithms, an example of non-adaptive algorithms, use routing tables that are configured

and manually maintained by network managers. Adaptive routing algorithms play a fundamental role in the Internet. Most current routing protocols fall into this class.

From the viewpoint of packet delivery along paths, there are two routing modes which can be deployed by routing protocols: hop-by-hop mode and explicit mode. In explicit routing, all intermediate nodes will follow a path specified by some other entity, usually the source (in source routing) or a network manager. The desired path is established and used as an end-to-end path. However, in hop-by-hop routing, every node independently calculates its own routing table, and independently makes all routing decisions. Such independence of routing decisions at each node favours scalability, but makes it hard to choose consistent paths (expected by the source node) at each intermediate node.

In *source routing algorithms* (one kind of explicit routing) [Sun77], a node keeps information about complete paths to all possible destinations. When the node sends a packet, the complete path specification is stored in the packet. Intermediate nodes use the path information supplied to forward the packet onward towards its destination. This feature makes explicit routing very useful for a number of purposes, such as implementing policy routing [Ste93a, Ste93b] or traffic engineering [Awd99, Awd02]. The MPLS [Ros01] framework and the ATM PNNI [ATM02] routing protocol provide support for source routing.

Major benefits of explicit routing are:

1). *Improving path controllability* – This is obvious. The sender of the packets can precisely control the path through the network to the destination. However, in hop-by-hop routing, every node makes routing decision independently.

2). *Enhancing path stability* – Since only the source node will perform path calculations, certain methods may be easily applied at the source to reduce recalculation for new paths.

3). *Eliminating path cycles* – loop avoidance is not a big problem in explicit routing. After the source calculates a loop free path, all intermediate nodes will follow this path. Hence, further mechanisms for loop avoidance may be not necessary.

4). *Supporting QoS guarantees* – explicit routing is essential to support desired QoS guarantees in transmission. When the source performs path calculations, the QoS demands of a session request can be taken into account in the design of the path finding algorithms.

5). *Improving routing security* – the entire end-to-end path can be solely decided by the sender. Hence, the sender can direct data packets so that they reach the destination through trusted nodes and links.

Routing algorithms also can be categorized as *centralized* or *distributed.*

Centralized routing algorithms are controlled by a single, centralized node and that node requires global knowledge of the network topology. The central node runs the routing algorithm, and informs the other nodes of the network about the paths from each source to each destination. However, these types of algorithms suffer serious scalability and reliability problems. For example, when the central node fails, the whole network will also fail.

Distributed routing algorithms do not depend on a central node (hop-by-hop routing falls into this category). Each node independently selects a successor node from its set of adjacent nodes to use to reach a given destination. The complete path for a given node pair is formed by consecutive successor nodes. Distributed routing algorithms can be classified as *link-state* or *distance-vector* algorithms [Hui95].

In distance-vector algorithms, such as RIP [Hed88] and RIP2 [Mal98], each node in the network keeps a routing table. This routing table contains entries for all the reachable destinations of the network. Each entry includes the following information – the distance to each destination and which node is the next stop on this path (i.e., the vector). By broadcasting each node's current routing table to all its neighbors, nodes can compare the routing tables and then choose the minimum cost paths to desired destinations. Distance-vector algorithms often use a very simple metric, namely the *hop count* (i.e., the number of hops) of a path, to calculate the best path. A path with the minimum hop count to a destination will be selected for data transmission. In general, the advantage of the distance-vector algorithm is its simplicity.

One major drawback of the distance-vector algorithm is the so-called *Counting-to-Infinity* problem [Hed88], which makes the routing algorithm very slow to respond to topology changes in the network.

In link-state algorithms, such as OSPF [Moy98b] and IS-IS, [ISO02], each node builds a description of the entire network topology by receiving link-state updates. A link-state update message describes local link states (e.g. current link delay) and associated neighbors for a particular node. Once a network has converged to a steady state, each node in the network will have an identical copy of the link-state database, which represents the latest network topology. A shortest-path algorithm can be applied to calculate paths of minimum total delay to destinations based on such a database.

Link-state algorithms eliminate the Counting-to-Infinity problem in path calculation when network topology changes, and they have much faster response time than distance-vector algorithms do. A global view of the network topology can be extracted from the link-state database, upon which complicated routing mechanisms can be built, for example, to calculate paths that meet hard QoS guarantees or reliability requirements. Unfortunately, the link-state algorithms do not scale well owing to the need to provide the entire topological description at each node, so their use in practice is restricted to small networks or to small components of larger networks, e.g. the component *Autonomous Systems* [Moy98b] of the Internet.

2.3 OSPF and QOSPF

IP routing protocols can be divided into two classes: *Interior Gateway Routing Protocols* (IGPs) and *Exterior Gateway Routing Protocols* (EGPs) [Hui95, Tan96]. IGPs are used for routing in a network, all of which is under a common administration, i.e., a component Autonomous System (AS) of the Internet. EGPs are used to exchange routing information between networks that are located in different ASs. As a popular link-state IGP, *Open Shortest Path First* (OSPF) [Moy98a, Moy98b] is designed to be used inside an AS. *Border Gateway Protocol,* (BGP) [Rek95], a scalable distance-vector algorithm, is often used as an Internet Exterior Gateway Routing Protocol.

OSPF, a link-state algorithm, is a dynamic routing protocol designed to support routing in TCP/IP networks. It was developed by the OSPF working group of the *Internet Engineering Task Force* (IETF) [Moy98b]. Each node stores an identical copy of a link-state database that describes the latest topology of the whole network. Using this database, every node individually calculates the *Shortest Path Tree* (SPT) to all other nodes with itself as the root. Such a path calculation uses Dijkstra's shortest path algorithm [Dij59], which has the time complexity $T_{Dijkstra} = O(MlogN)$ using a heap data structure. Then, data packets are transmitted along those paths with shortest length.

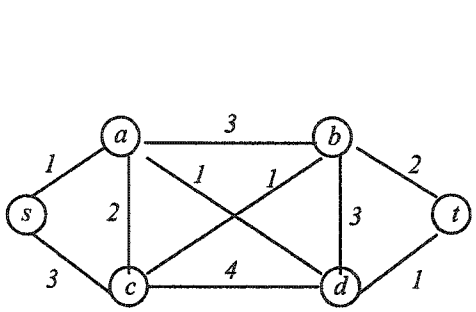
OSPF consists of three major sub-protocols: the *Hello Protocol,* the *Routing Data Exchange Protocol* and the *Flooding Protocol.* The Hello Protocol dynamically discovers and maintains neighbor adjacencies by exchanging periodical *Hello Packets.* The Routing Data Exchange Protocol is used to achieve synchronization of link-state databases when two nodes become adjacent. The Flooding Protocol performs reliable flooding of *Link-State Advertisements* (LSAs) to all nodes within a network whenever network topology changes occur. Every LSA is an individual entry in the OSPF link-state database. Figure 2.1A shows a sample OSPF network with *6* nodes. Figure 2.1B illustrates the SPT at node *s.* Figure 2.1D is the related link-state database built up from individual LSAs. Based on this database, the calculated routing table at node *s* is illustrated in Figure 2.1C.

Multiple SPTs may exist in certain networks. It is common that multiple equal-length shortest paths may exist between some pairs of nodes. Figure 2.1A is such an example. OSPF will choose one for packet forwarding according to certain policy.

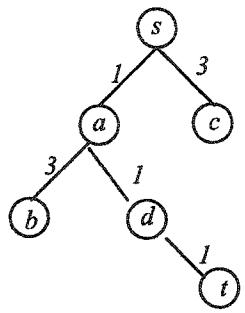
QOSPF [Gue97, Apo99a, Apo99b] is an extension to OSPF for achieving Quality of Service (QoS) routing, which tries to improve network utilization and user service levels based on certain types of explicit routing. The latest link load information is contained in the QOSPF link-state database. QOSPF includes metrics required to support QoS, an extension to the OSPF LSA mechanism to propagate updates of QoS metrics.

QOSPF selects QoS paths from candidates by *the widest-shortest path selection criterion* [Apo99b, Cor90a]. When several such paths are available, the preference is the path whose available bandwidth (i.e., the smallest value of available bandwidth on any of the links in the path) is maximal. This selection strategy benefits load balancing. In the

worst case, complexity of precomputation of the widest-shortest paths for K bandwidth levels is *O(KMNlogN)*.



(A). A sample OSPF network.



(B). The SPT at node *s*.

Destination	*Next Hop*	*Path Length*
a	*a*	*1*
b	*a, c*	*4*
c	*a, c*	*3*
d	*a*	*2*
t	*a*	*3*

(C). Routing table at node *s*.

LSA from s

NB	*Len*
a	*1*
c	*3*

LSA from a

NB	*Len*
s	*1*
c	*2*
d	*1*
b	*3*

LSA from b

NB	*Len*
a	*3*
c	*1*
d	*3*
t	*2*

LSA from c

NB	*Len*
s	*3*
a	*2*
b	*1*
d	*4*

LSA from d

NB	*Len*
c	*4*
a	*1*
b	*3*
t	*1*

LSA from t

NB	*Len*
b	*2*
d	*1*

(D). A conceptual view of the exchanged LSAs (here, *NB* – neighbor; *Len* – edge length).

Figure 2.1. The main data structures in OSPF.

In alternate path routing, information about link load and delays in a network is a fundamental base for alternate path calculations. Based on OSPF, QOSPF can provide such information by its link-state database. Hence, QOSPF can be further used as an underlying routing protocol to build reliable routing architectures.

2.4 Previous Work on Routing Reliability

The Multi-Protocol Label Switching (MPLS) [Ros01] is an IP-based routing architecture using ATM-like "label swapping" to speed up packet forwarding without introducing changes to existing IP routing protocols. MPLS effectively provides a path or

circuit constructed on top of IP, permitting the selection of a path from source s to destination t and then constraining all IP datagrams of a flow to follow that path by way of the explicit routing mechanism. General issues on MPLS-based recovery and fault tolerance are addressed in [Sha03, Far03].

Other literature [Che99, Dov01] addresses major aspects and challenges when routing reliability and QoS guarantees are considered in routing protocols. In this section, we survey some research work on improving routing reliability.

Minimizing bandwidth demands on alternate paths [Ban01a, Li02, Nor01] is one of the major research concerns when such paths are used to restore failed paths. As a backup restoration, an acceptable alternate path should achieve efficient bandwidth usage. Based on this idea, Li proposed a path-selection algorithm, called *Full Information Restoration* (FIR) [Li02], for restoration of a connection over shared bandwidth in MPLS and *Generalized MPLS* (GMPLS) [Man03] frameworks. Extensions to GMPLS signaling protocols were also proposed to collect information of bandwidth usage. By modifying the weight of each link based on its load, FIR calculates the disjoint paths (for the primary and alternatives) by using a shortest path algorithm and link-pruning process. Li's approach may reduce the amount of reserved restoration bandwidth dramatically in the calculated paths; however, this approach focused on a new signaling protocol and used disjoint alternate paths, which does not meet our need to improve reliability by non-disjoint alternate paths.

Norden [Nor01] investigated distributed algorithms for routing with backup restoration (i.e., traffic on a failed working path will be restored by a precalculated backup path), and proposed a new concept of a *Backup Load Distribution* (BLD) matrix that captures partial network states. Each node maintains an $N \times N$ BLD matrix, which encapsulates values of all link capacities and the load states used by the primary and backup paths. Such a BLD matrix can be exchanged between peer nodes. During path calculation, link weight can be updated based on the BLD matrix, and a pair of paths (the primary and alternative) will be calculated. However, such a path-selection algorithm is developed for achieving load balance and effective bandwidth usage on the calculated path pair, and does not concentrate on improving reliability. The selection of the unique

backup path is only based on link load information. Another drawback of this approach is the significant overhead of message exchanges for BLD updates.

Banerjee and Sidhu proposed a failure protection model [Ban01a] to tolerate random failures. This approach uses a concept called *resilient bandwidth,* which is a certain amount of bandwidth shared among paths for a given node pair. When sufficient resilient bandwidth is reserved, a backup path can provide protection to multiple primary paths. The backup path is disjoint with the primary paths. This approach focuses on optimizing bandwidth usage and reducing bandwidth waste upon the backup paths. The drawback of this failure protection mode is that link load is the only criterion used to calculate the single backup path, which is disjoint to the primary path. Hence, this approach does not meet our need to use non-disjoint paths in alternate path routing.

In guaranteed QoS services, it is essential to meet end-to-end delay of transmission in delivering multimedia streams with guaranteed QoS, as well as to have enough bandwidth capacity. Lee and Gerla developed an approach [Lee01] for improving fault tolerance and load balancing in QoS provisioning using multiple alternate paths. This approach considered the constraints of end-to-end delay and bandwidth demand in path calculations. QOSPF [Apo99b] is used as the underlying routing protocol to offer delay and load information on links. The proposed multiple QoS path computation algorithm running at source nodes searches for maximally disjoint (i.e., minimum overlapping) paths. Multiple paths are calculated subject to the following conditions:

1). minimizing hop count,

2). satisfying the given QoS constraints (delay and bandwidth), and

3). maximally disjoint from already computed paths (by tracking common edges).

This approach combines QoS routing [Apo98] and reliability considerations. QoS routing is a routing scheme that supports the desired QoS guarantees. The proposed path finding algorithm selects suitable alternate paths from possible paths while increasing the number of hops from a given source to destination. The algorithm is based on a *Breadth First Search* (BFS [Cor90b]) method, which may have to search for all possible paths

from the source to the destination in the worst case. Computational overhead of this algorithm was significant. Also, routing reliability for this approach is only indirectly represented as the number of common edges, without precise definitions or quantified calculations. Further discussion of this algorithm is provided in Section 5.3.1.

Lee and Gerla's work is closely related to ours. However, our work goes much further. We will avoid their drawbacks of significant computational overhead, analyze the impact to the reliability caused by common edges, and propose new efficient alternate path finding algorithms subject to the desired constraints of QoS guarantees and reliability requirements.

Veerasamy and Venkatesan proposed a method to split traffic [Vee94] on the primary and disjoint alternate paths to achieve reliable routing. When one path fails, associated traffic on that affected path will be rerouted on the alternate paths. The alternate path will require enough prereserved bandwidth for future restoration. Two splitting methods, even-splitting and best-splitting, are discussed in the paper. This approach focused on improving sharing of spare bandwidth and reducing network cost, and did not fit our objective of non-disjoint alternate path routing.

One approach to reliable routing in MPLS is to use the so-called *Minimum Interference Routing* (MIR) [Kar00, Kod00, Ban01b, Ban01c], which means that the calculated paths for every node pair will have minimum *"interference"* with each other. In other words, a newly selected path should avoid picking *critical links* that may be critical for satisfying future requests. The critical links indicate the links with a very heavy traffic load. To obtain MIR, Kodialam, Lakshman and Kar proposed a *Minimum Interference Routing Algorithm* (MIRA) [Kar00, Kod00], where the smallest weighted path is calculated by Dijkstra's shortest path algorithm, after the critical links are identified and pruned. Banerjee and Sidhu extended the MIRA algorithm [Ban01b, Ban01c] by considering delay constraints in the path calculation, as well as bandwidth demands. The new algorithm developed uses a *K*-shortest paths algorithm [Epp99] to calculate candidate paths satisfying an end-to-end delay constraint. Among the *K* paths, the least critical path (containing the fewest critical links) is chosen.

MIR can achieve satisfactory QoS routing and load balancing in a static network. However, MIR is completely different from alternate path routing – only one path is calculated for a given node pair. When a network component fails, one or more working paths will be affected, and path calculation and establishment have to be reinvoked.

In summary, although the existing work is related to routing reliability, it does not exactly fit our goals of improving reliability by using non-disjoint alternate paths. Therefore, we have to develop new routing algorithms for these purposes.

2.5 Related Work on Utility-optimal Admission Control with Hard QoS Guarantees

Hard or absolute QoS guarantees for multimedia service require end-to-end guarantees covering the server, network and client. The server should have enough resources to deliver multiple multimedia streams with the desired QoS demands. The network should provide reliable connections with enough bandwidth, low enough latency and jitter. The client's machine should be powerful enough to play the multimedia streams with the desired QoS demands.

The admission controller in [Kha97, Kha98] works as a resource manager by allocating the resources such as network bandwidth, CPU cycles, I/O bandwidth, memory, etc. of the server to the user when his/her multimedia session starts. It also performs QoS adaptation dynamically by upgrading or downgrading a session in progress.

As a resource manager for the bandwidth of the links of a network, SLAOpt [Wat01] is an SLA-based admission controller for *Enterprise Networks* (EN). An EN is defined as a network with a limited number of nodes (typically less than *100)* and links administered by a single organization or an autonomous subsidiary of an organization. The objective function of SLAOpt is to optimize the utility (often revenue) of the system by applying the Utility Model [Kha98].

Based on the Utility Model, utility-optimal admission control can be mapped onto an MMKP [Kha98, Akb01b], a variant of the classical *0-1 Knapsack Problem* [Mar90]. An exact solution of the MMKP, a NP-hard problem, is not suitable for the real-time admission control problem. Hence, the Admission Controller provides a fast near optimal solution to the MMKP, from time to time, in order to determine which sessions to upgrade or admit at which QoS levels. Three heuristics namely, M-HEU, I-HEU and C-HEU [Akb01b, Akb02b], exist for solving the MMKP for real-time admission control.

However, SLAOpt is a static admission controller when dealing with network topology changes. SLAOpt does not consider failures of the network. SLAOpt focuses on performance optimization in bandwidth allocation and assumes that the network topology is static. This is not true in practice. More research work is necessary to meet reliability requirements in the admission controller, while still optimizing system utility. New mechanisms to respond to network component failures have to be developed.

3. Alternate Path Routing Mechanisms

In alternate path routing, multiple paths (i.e., a path group) for a given source-destination node pair are typically precomputed for a session request. Among the calculated paths, a preferred one can be chosen as the primary path according to a certain policy, and the others are used as alternate paths. Ranking the alternate paths is also a policy issue.

There are two ways to use the alternate paths: the *"parallel"* scheme and the *"primary/backup"* scheme. In the first scheme, the alternate paths work together with the primary path for balancing traffic load in a flow-based or packet-based mode. Load balancing is the main goal. In the second scheme, the alternate paths work as backups. Hence path recalculation is not necessary if there are alternate paths available when failures occur in a network. This scheme can significantly improve reliability. Our analysis of reliable routing focuses on the primary/backup scheme.

3.1 Classification of Alternate Paths

Traditionally, alternate paths can be calculated by two categories of algorithms – disjoint path algorithms and non-disjoint path algorithms. Much research has been done on developing algorithms to calculate disjoint paths [Suu74, Has85, Gol88, Che97].

Non-disjoint paths can also be used for alternate path routing. The calculated paths in this case can exhibit a limited number of shared nodes and shared edges. The candidate algorithms for calculating non-disjoint paths are diverse [Kub97, Lee99, Epp99, Pu01b]. Typically, *K*-shortest paths algorithms [Yen71, Mar99, Jim99] are commonly used to calculate such paths.

3.1.1 Disjoint Paths

Disjoint paths are often used for load balancing or as backup paths. Generally, a max set of disjoint paths for a given node pair can be obtained by calculating the maximum flow [For56, Din70, Gol98, Nag01] between a source-destination pair. The *Disjoint Paths Problem* can be described as:

> *For a graph G(V, E) and a given node pair (s, t) ∈ V, compute a maximum set of disjoint paths from s to t.*

By definition, no shared edges or nodes will exist among the paths calculated by the disjoint path finding algorithms. This property can significantly improve routing reliability. However, drawbacks still exist: disjoint backup paths may not always exist for a primary path. Also, in some network topologies, we may not find as many disjoint paths as desired, or the calculated disjoint paths may be too long. Moreover, the shortest path may not be included in the output set of disjoint paths.

3.1.2 K Best Disjoint Paths

The *K Best Disjoint Paths Problem* [Suu84, Cas90, Nik97] improves on the Disjoint Paths Problem by incorporating additional optimization. The problem is:

> *For a graph G(V, E) and a given node pair (s, t) ∈ V, compute K paths $\{P_1, P_2, \ldots P_K\}$ from s to t, subject to the following constraints:*
>
> - *Path $\{P_1, P_2, \ldots P_K\}$ are disjoint with respect to each other*
> - *$\Sigma L(P_i)$ is minimum. $L(P_i)$ is length of P_i, $i \in \{1, 2, \ldots K\}$*

The same drawbacks exist for the *K* best disjoint paths as in the previous case:

1). We may not find enough disjoint paths,

2). The calculated paths may be too long,

3). The shortest path may not be included in the calculated paths.

3.1.3 K Shortest Paths

Existing *K*-shortest paths algorithms [Epp99, Mar99] rank paths by their lengths in ascending order. The problem of *K Shortest Paths* calculation can be described as:

> *For a graph G(V, E) and a given node pair (s, t) ∈ V, compute K paths {P_1, P_2, ... P_K} from s to t. Let Q be a set including all possible paths, and set Q_K = {P_1, P_2, ... P_K} ⊂ Q. The calculated K paths must satisfy:*
>
> - $L(P_i) \leq L(P_{i+1})$, $L(P_i)$ *is length of* P_i, $i \in \{1, 2, \ldots K\}$
> - $L(P_K) \leq L(q)$, *for any path* $q \in \{Q - Q_K\}$

That is, when running a *K*-shortest paths algorithm, not only is the shortest path to be determined, but also the second shortest, the third shortest, and so on up to the K^{th} shortest path.

The *K*-shortest paths algorithms can minimize alternate path lengths and can find relatively large numbers of paths, because shared edges or nodes are tolerated. These properties of the *K* shortest paths solve the problems that are encountered in the disjoint paths. However, the *K*-shortest paths algorithms also have drawbacks when used directly in a routing protocol. There may be too little variation (i.e., too much overlapping) among the calculated paths. Hence, for reliability reasons, the *K* shortest paths sometimes cannot be directly used as alternate paths.

3.2 Adaptive Selection of New Paths When Failures Occur

Link or node failures may occur in networks. Although failures are currently much less common than those in the past due to wide usage of fibre links, failures are still not rare. The consequences of a failed link are very serious, owing to the large number of

sessions which may be affected, due in turn to the very large capacity of fibre links. A suitable response to such a failure in dynamic routing protocols is to quickly find new optimal paths and reroute the traffic via the new paths (called *fault recovery),* followed as soon as possible by repair of the failed component. The recovery mechanism used to bypass the failures should be as fast as possible for minimizing down time and use as few network resources as possible to minimize costs. A similar process also occurs when link lengths grow to unacceptable values due to increases in their traffic load.

Convergence time is one of the important metrics used to measure reliability of a routing protocol. During the interval of convergence time, the protocol has to find a new path; however, *black holes* (i.e., network partitioning) or path cycles may occur. Routing during convergence time may be unstable and inconsistent.

When the network topology changes, a conventional way to perform routing recovery in single path routing is to recalculate new paths, i.e., on-demand (on-the-fly) path calculation. In alternate path routing, the alternate paths can be precalculated and preestablished. The following subsections overview these two methods of path calculation.

3.2.1 On-demand Path Calculations

This is the first path calculation method mentioned above. It calculates new paths when needed. In link-state routing protocols, the simplest way to bypass failures is to rebuild the entire *Shortest Path Tree* (SPT) and the routing table after a node receives the topology update messages. Some implementations of OSPF [Nex01] deploy this method to respond to link-state updates. Although it is the easiest way to adjust the routing table according to the latest network topology, rebuilding the whole routing table will always be expensive, and will result in long convergence times. Routing cycles may be easily introduced during such a long convergence time. Hence, it is not preferable to constantly rebuild the entire SPT and the routing table.

To relieve the burden of rebuilding computations, partial rebuilding of the SPT is often applied in routing protocols [Mcq80, Nar99a, Nar00b]. The motivation for this

approach is to reduce recalculation overhead. It tries to dynamically maintain the SPT after a failure occurs, instead of completely recalculating the entire SPT.

McQuillan [Mcq80] modified Dijkstra's shortest path algorithm, so that changes in network topology require only some incremental calculations. His algorithm intends to identify the affected and unaffected nodes when failures occur, and then partially rebuild the SPT including the affected nodes. Because the SPT is just partially rebuilt, computational overhead of the McQuillan's approach is reduced.

Narvaez et al. presented another dynamic SPT algorithm [Nar99a, Nar99b] that makes use of the structure of the previously computed SPT. In their Ball-and-String model, the increase (or decrease) of an edge weight in the SPT corresponds to the lengthening (or shortening) of a string. Based on this model, they derived an efficient algorithm that propagates changes in distances to all affected nodes, in a natural order and in an economical way. The priority of node relabelling can be determined by using new notions describing the maximum decrement or increment of path length. Although this approach can speed path recalculations when failures occur, a certain amount of computation is still necessary.

After a new path is recalculated, it has to be established by underlying signaling protocols before data transmission. This is a non-avoidable step in the on-demand path calculation mode. Along the new path, all intermediate nodes will install the path and reserve the resources required by the incoming data flow. Obviously, the time cost of path establishment is a significant part of the convergence time, and this cost varies with network size.

3.2.2 Precalculated Alternate Paths

As stated, certain computational overhead is unavoidable in on-demand path calculation, and it causes a longer convergence time. The use of precalculated and preestablished alternate paths is an approach frequently used to improve the reliability of routing in networks [Wan90, Bah92]. Multiple paths are calculated and established beforehand for a given node pair, say x and y. When node x detects that the working path

to y has failed, node x will perform a path switch, meaning that the failed path will be replaced by a precalculated alternate path. Such a path switching process is obviously much faster than recalculating a new path to node y followed by path establishment, because the required computation is only a table lookup.

As stated, the alternate paths may work in the primary/backup scheme or in the parallel scheme with the primary path. The primary/backup scheme is feasible whether the alternate paths (including the primary path) are disjoint or non-disjoint. On the other hand, the parallel scheme may balance the traffic load among links and thus may not need path switching when failures occur. This scheme is only feasible when the calculated paths for a given node pair are disjoint. For non-disjoint alternate paths, there have by definition shared edges or nodes, which may create bottlenecks and cause congestion. Resolving these problems in the parallel scheme is still an open topic in the area of routing protocols and is beyond the scope of this dissertation.

The major benefit of using the alternate paths in routing protocols is the improvement in convergence time when failures occur. The small time needed for table lookup will significantly reduce calculation and require less of other system resources. For comparison, in the previously discussed approach of partially rebuilding the SPT, a certain amount of computation is always required to maintain the affected SPT when failures occur. However, three drawbacks exist for the alternate path approach:

1). Reliability improvement in alternate path routing has the cost of more resources reserved on the alternate paths. If the alternate paths are seldom used, the extra resources reserved on these paths will be wasted.

2). The alternate paths are precalculated, therefore these paths cannot respond to the latest network topology. In other words, we may use suboptimal alternate paths as new working paths after multiple failures occur.

3). When all available alternate paths are affected by failures, path recalculation has to be performed. Generally, to calculate one primary path and multiple alternatives for each destination is more costly than to calculate one shortest path for each destination. Furthermore, setting up multiple paths may take more time than setting

up only one path. Therefore, in alternate path routing, path recalculation and reestablishment must be infrequent.

If we can provide viable solutions to the above problems, the alternate path routing approach will be an efficient way to shorten routing convergence time and improve routing reliability. This is one of the focuses of this dissertation.

3.3 Fault Restoration Methods

Several fault restoration techniques [Che00, Sta99, Vee99] have been developed for fast bypassing of a failed link or node in routing protocols. Some techniques require precalculated alternate paths, and some do not. In either case, precalculated alternate paths can speed fault recovery. In this section, we overview these techniques.

3.3.1 Link Restoration

Link restoration [Che00] finds an alternate path that reconnects the two end nodes of a failed link $(w \rightarrow v)$, as shown in Figure 3.1. The affected working path is $s \rightarrow w \rightarrow v \rightarrow t$, and the back up path is $s \rightarrow w \rightarrow x \rightarrow y \rightarrow v \rightarrow t$. Although one rerouted path is shown in the figure, all working paths which utilize the failed link must be rerouted at the same time. The new paths can be calculated on-demand after the failure occurs, or precalculated as backups. For restoration speed and efficiency, the alternate paths should be precalculated and preestablished, at the same time as the primary path.

Bandwidth reservation along the alternate path can be performed during the step of path preestablishment, in order to make it available for active traffic when the working path fails. If bandwidth is not reserved, it is possible that the preselected alternate paths may have inadequate bandwidth available when the failure occurs.

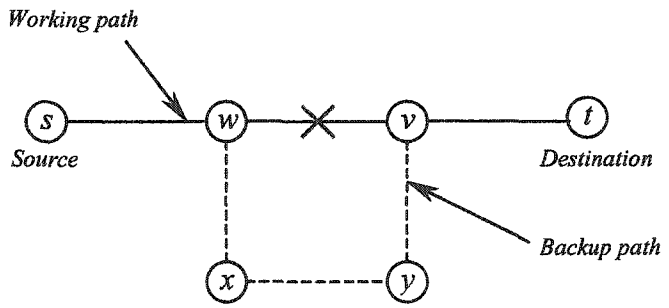


Figure 3.1. Link restoration.

Narvaez proposed a fault-tolerant link-state routing protocol without flooding [Nar00a]. The idea is to construct a shortest restoration path for each unidirectional link fault by performing only local updates on the affected nodes. Wu extended this idea to bi-directional link faults, and proposed a new protocol [Wu01] that can efficiently update the routing tables of nodes on the restoration path. The construction of the restoration path is initiated at both end nodes of the faulty link.

Link restoration is a simple and fast recovery technique, because no fault notification messages need to be sent to other nodes. The end nodes of the failed link initiate the path restoration process, which is transparent to the other nodes in the network. However, there are several disadvantages to the link restoration technique:

1). It cannot efficiently handle node failures. For example, assume node v fails in Figure 3.1, then node w initiates link restoration process. The new path from node s to node t will be $s \rightarrow w \rightarrow x \rightarrow y \rightarrow v \rightarrow t$. However, this path does not work because it contains the failed node v.

2). When multiple link failures occur, the final path from the source to the destination becomes quite complex and end-to-end delay may grow significantly; even worse, routing cycles may occur.

3.3.2 Partial Path Restoration

Partial path restoration [Che00] attempts to find an alternate path from the source, say *s*, to the downstream end node *v*, of the failed link *(w→ v)*, as shown in Figure 3.2. This technique can restore any possible link or node failures between *s* and *v*. The restoration process will be initiated by *s*, after *s* receives the notice of the failure.

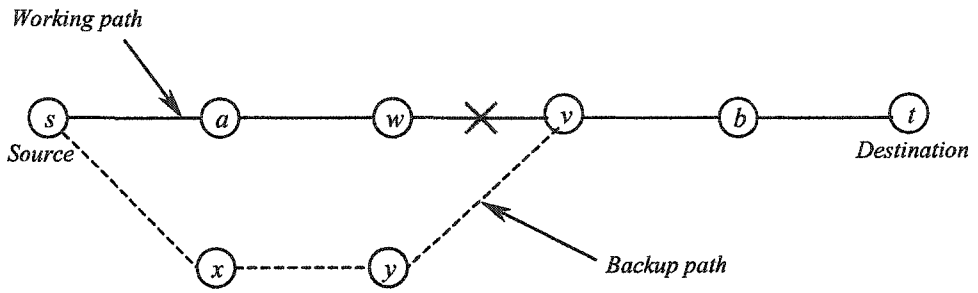


Figure 3.2. Partial path restoration.

However, response time to the failures in partial path restoration will be longer than that of the link restoration technique. It will take a certain amount of time for the source *s* to receive notice of the failed link *(w, v)* after the failure occurs. Also, *s* has to precalculate alternate paths to each of the intermediate nodes, and store them for future use. This creates a burden of computational and storage overheads. Moreover, the end-to-end delay guarantee may be broken if the alternate paths are used, because the complete path after restoration is composed of two segments: *(s→ x→ y→ v)* and *(v→ b→ t)*. Similar problems as with the link restoration technique exist in the case of multiple failures. Also, the complete path from *s* to *t* after the alternate paths are used may become too long and cycles may occur.

3.3.3 Path Restoration

In partial path restoration, the segment of the original working path downstream from the failure to the destination is unaffected by the rerouting process. For more flexibility, it might be desirable to reroute the entire failed path to another path between the source *s*

and destination *t*. Shown in Figure 3.3, the new path after restoration at source *s* is *(s→ x→ y→ t)*. This technique is called path restoration [Che00]. In this technique, the alternate paths from *s* to *t* must be precalculated and preestablished. Similar to partial path restoration, response time to the failures in path restoration is longer than that of link restoration.

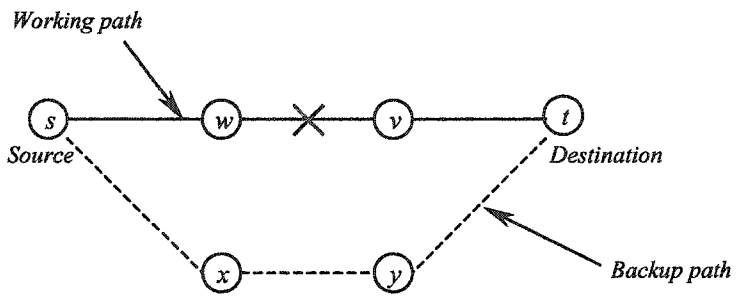


Figure 3.3. Path restoration.

In the context of alternate path routing, path switching represents the process of performing path restoration at source *s:* identifying the failed working path, switching the failed path to a backup, and resuming data transmission along the backup path.

Path restoration offers more flexibility than the link restoration or partial path restoration technique. Any alternate paths from *s to t* can be used for rerouting, subject to the requirements of QoS routing. This technique is fairly economical and can reroute the paths more efficiently. Important features of the path restoration technique are:

1). It can guarantee end-to-end delay after an alternate path is selected.

2). The complete path after path switching is predictable. Routing cycles cannot be formed.

3). It reduces computational and storage overheads as compared to the previous two path restoration techniques.

3.3.4 Virtual P-Cycles

Grover and his research group have developed a path restoration technique called the *Virtual Protection Cycle* (p-cycle) [Sta00a, Sta00b, Gro02] for fast restoration from failures. As a ring protection technique, p-cycles were initially conceived as preconfigured undirected protection cycles for use in *Wavelength-Division Multiplexing* (WDM) and *Synchronous Optical NETwork* (SONET) transport in mesh networks, and were extended to IP networks by implementing them as *Virtual Circuits* (VCs) [Sta99].

In the event of a failure, affected packets are encapsulated and immediately diverted by a preconfigured p-cycle, which is specified as an alternate path entry in the routing table. Those packets will travel through the p-cycle until the failure is cleared; the packets are then removed from the p-cycle and routed normally towards their destination. The p-cycle provides a kind of circular shift as an immediate detour for the packets, preventing their loss, until routing reconvergence occurs.

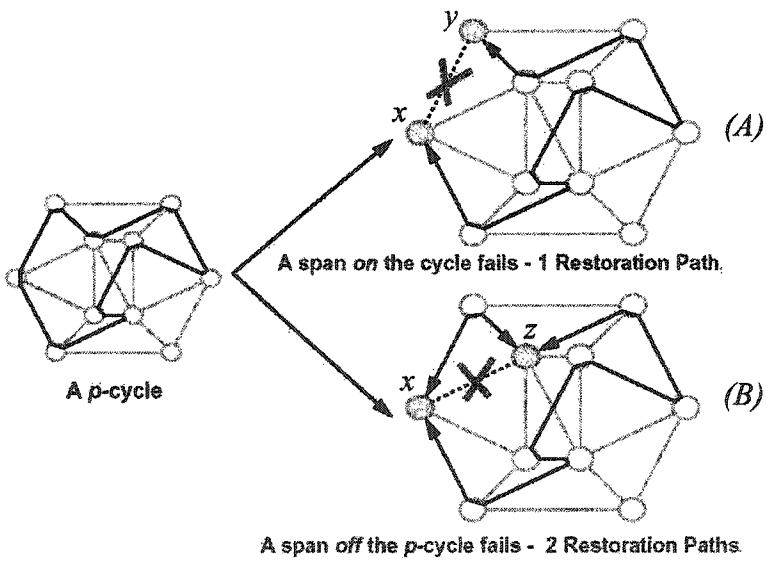


Figure 3.4. P-Cycle recovery from a span failure.

The mechanism for achieving protection by a p-cycle is illustrated in Figure 3.4 [Sta00b]. The p-cycle shown in case (A) has itself been disrupted by one failed span (a span is an optical link) on the p-cycle, while case (B) shows an straddling span failure. A

straddling span is a link not on the p-cycle but the two end nodes of the span are on the p-cycle. In both cases, the p-cycle can offer full restoration – in case A, surviving spans of the remaining p-cycle will form a restoration path and connect two end nodes of the failed span *(x, y);* in case B, there are two restoration paths to connect the failed straddling span *(x, z)* from opposite directions.

The p-cycle technique is mainly considered at the network design level to improve the survivability of the network. This technique has good restoration performance as a static technique. However, it is not easy to dynamically reoptimize the configurations of multiple p-cycles (e.g., p-cycle number and arrangement) to provide protection for every node and span based on the latest topology. Significant computation may be involved for such changes. The p-cycle technique may not provide satisfactory fault recovery in the case of multiple failures. Furthermore, the end-to-end delay of the restoration paths may vary significantly, depending on where the failures are. For example, Figure 3.4A and 3.4B show two different cases.

3.4 Analysis of Alternate Path Routing

Features of alternate path routing when responding to dynamic changes of network conditions may differ from traditional single path routing. Such differences are discussed in the following subsections. For reasons of comparability, our analysis is based on an assumption that both single path routing and alternate path routing use the same link-state protocol, e.g. OSPF [Moy98b].

3.4.1 Responsiveness

A routing protocol can work correctly only when its convergence time is smaller than the rate of changes of network topology and traffic load.

However, for extremely fast and temporary changes, both routing schemes (alternate path and single path routing) have rather limited capability to make appropriate

responses. Route updating is often slow as compared with changes in traffic intensity or load on individual links. For example, OSPF sets the interval of Hello packets to *10* seconds in a LAN [Moy98b]. Therefore, load changes may not be detected immediately. Moreover, the communication delay also imposes a limit on how fast the routing algorithm can react to traffic changes. As the network grows, so will the communication delay of flooding link-state updates. It may be possible that a temporary failure or transient congestion on a link has already recovered, when the link-state update information providing notification of the error is flooded to all nodes in the network.

For the case of long term link/node failures or slowly varying traffic load changes, both routing schemes can respond effectively. Alternate path routing yields much faster response than single path routing does. The reason is that path recalculation and reestablishment have to be performed in single path routing when failures occur, while only path switching is needed in alternate path routing.

Hence, one fundamental assumption for our work on reliable routing is that the rerouting speed of the protocols should be faster than the rate of network topology changes. Dealing with very fast traffic load variation and studying efficient congestion control mechanisms for routing protocols are beyond this research.

3.4.2 Estimation Error

It is important to be able to characterize the delay that individual packets are subjected to in their journey through the network [Wan92, Wal97]. The overall transmission delay ($T_{overall}$) can be traced to four main causes.

1). *Processing delay* (T_{proc}) – This is the time it takes to process a packet at each node and prepare it for retransmission. This delay is determined by the complexity of the protocol and the computational power available at each node.

2). *Propagation delay* (T_{prop}) – This is the time it takes a packet to actually propagate through a communication link. This delay is dictated by the length of the communication path and the transmission medium.

3). *Link service delay (T_{svce})* – or Link/Line service time. Also known as the transmission delay. This is the time that it takes to stuff an entire packet, from the first bit to last bit, into a communication link.

4). *Queuing delay (T_q)*– This is the time that a packet has to wait in a queue for link service. T_q is governed by the length of the queue on a link and the scheduling strategy used by the node.

Let n represent the number of nodes in a path P. Assume all nodes and links in the network are identical. Let T_q^i represent the queuing delay (including the packet to be served) of node i on the path P. Then, the overall delay $T_{overall}$ (indicate the delay of the path) can be estimated as:

$$T_{overall} = n \times T_{proc} + (n-1) \times T_{prop} + \Sigma T_q^i \qquad (3.1)$$

The propagation delay T_{prop} is determined by the length of the physical channels and is independent of the actual traffic on the link; likewise, the processing delay T_{proc} is determined by the available hardware and again is not affected by traffic. We hence focus on the queuing delay ΣT_q. Here, for simplicity, our discussion ignores the possibility of retransmissions, which of course would add to the overall delay. In practice, however, retransmissions are rare in most data networks (lightly loaded, optical), so this assumption may be safely made.

For a given network, the three types of delays, i.e., T_{proc}, T_{prop} and T_{svce}, may exhibit little fluctuation even though big traffic load changes occur in the network. However, the queuing delay T_q may show remarkable swings with changes of traffic load, and hence of link congestion. T_q contributes the most to variations in the overall delay $T_{overall}$.

A flooding mechanism is applied to distribute link-state updates in OSPF, from one node to all the others, so that every node has an identical copy of the link-state database. Then, the routing paths are calculated based on the contents of the database. However, because of the delay in receiving the routing data (i.e., the link metrics), the values that a remote node uses to perform the calculation may not match the latest network topology. The difference between the estimated value (i.e., the old value of the metric used to

perform path calculation, measured at time t_1) and the measured value (i.e., the real value of the metric upon the time of path calculation, measured at time t_2) is called *estimation error.*

Postponed response to failures in a routing protocol will cause estimation error. Moreover, estimation error still exists in a network with static topology (i.e., no failures) but with traffic load variation. Wang studied such estimation error [Wan92] for the shortest path algorithms by considering the overall delay of routing data through a link *i.* Let ρ_1 and ρ_2 be the network utilization factors at time t_1 and t_2, respectively. Let $\Delta\rho$ be the change of the network utilization, i.e., $\Delta\rho = (\rho_2 - \rho_1)$. The utilization of a link is determined by the load on this link.

Based on Wang's studies, Figure 3.5 plots the estimation error as a function of ρ_1 when $\Delta\rho$ is fixed. By increasing ρ_1 gradually, the estimation error remains very low until ρ_1 approaches *1.* This threshold behaviour results from the fact that the delay changes much more rapidly under heavy traffic. Figure 3.6 shows the estimation error as a function of percentage of change in the utilization factor. The estimation error is low when the traffic load is decreasing but rises sharply when the traffic load is increasing. The change in traffic load $\Delta\rho$ depends on the nature of traffic and the overall delay. When this delay is large, the difference between ρ_1 and ρ_2 tends to increase. In general, the estimation error is low when the traffic load in the network is light or decreasing.

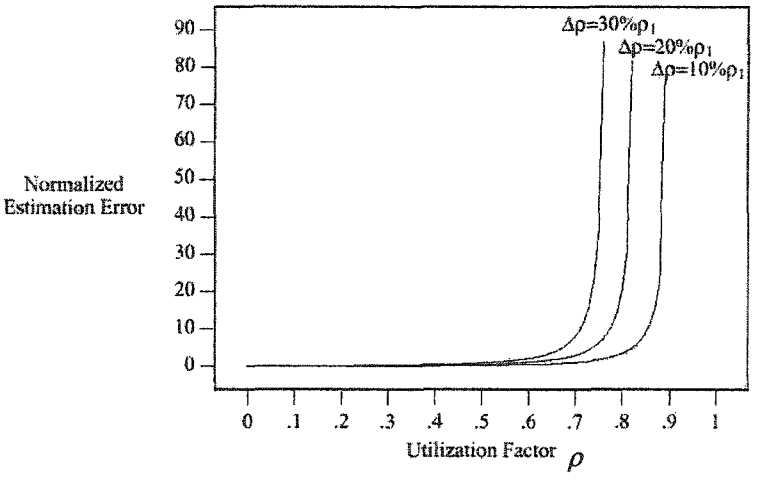


Figure 3.5. Normalized estimation error as a function of ρ_1.

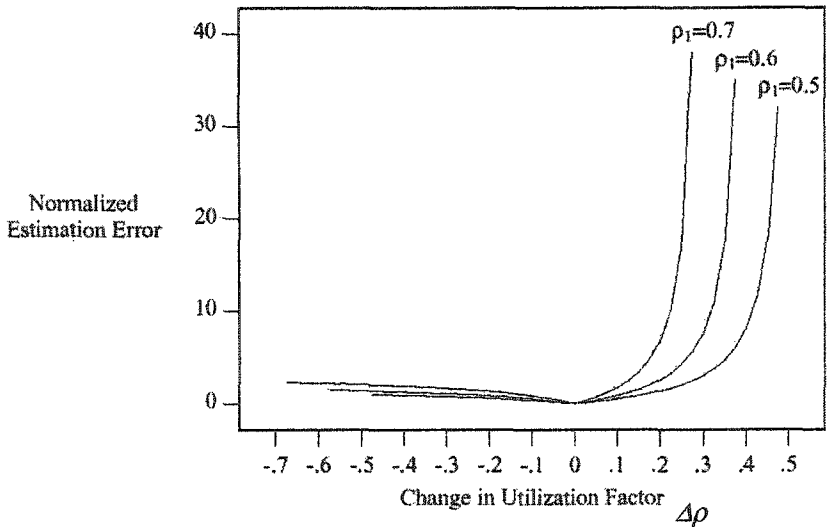


Figure 3.6. Normalized estimation error as a function of $\Delta\rho/\rho_1$.

The conclusions of Wang's studies can be used to analyze the estimation error for both single path routing and alternate path routing.

Typically, OSPF uses link delay as the metric to perform path calculation: such a metric has an indirect relation with traffic load on the link. As mentioned earlier, the queuing delay T_q depends on the utilization of the link. Under conditions of light loading, packet queuing is minimal, and thus T_q is largely negligible. In this situation, the overall link delay $T_{overall}$ will be dominated by the propagation delay T_{prop} and the transmission delay T_{svce}. The reported delay values are fairly good predictors of the delay encountered after rerouting and, in fact, routing tends to be fairly independent of traffic conditions. Under moderate loading, T_q is no longer negligible. However, routing changes result in moderate traffic shift. Therefore, T_q does not change too drastically and the delay metric remains a useful predictor of expected delay. On a heavily loaded network, queuing delay T_q may exert a significant influence on the link delay metric. In this situation, the estimation error for the metric values will be so large that the delay metric is no longer a usable predictor for the next path calculation.

A routing parameter *HelloInterval* in OSPF is used to represent the length of time between two Hello packets that the node sends to its neighbours to maintain the links and

measure the delay of the links. The typical value of the *HelloInterval* for a *local area network* (LAN) is *10* seconds, i.e., we can say that the time interval between the two path recalculations (i.e., route updating interval) T_{update} is about *10* seconds for a LAN in single path routing.

However, in alternate path routing, the frequency of path recalculations will be significantly reduced. In most cases, path switching will be used to replace affected paths with the precalculated alternatives. No path recalculation will occur if there are still available alternatives for a given node pair. Typically, the time interval between the two path group recalculations T'_{update} for alternate path routing may be much longer than T_{update} for single path routing in the same network environment. This is exactly the advantage of alternate path routing. On the other hand, within the longer interval T'_{update}, the traffic load on links may show notable changes, caused by longer accumulation, so that using precalculated alternate paths may not adapt well to the changes. During a long time interval between the two path group recalculations, the estimation error in alternate path routing would be bigger than that in single path routing. Hence, alternate paths may exhibit suboptimal performance. Only when traffic load is not very heavy can such estimation error be omitted.

Certainly methods can be used in alternate path routing to reduce the bigger estimation errors:

1). We can trace all dynamic changes of link lengths. After a source node *s* calculates multiple acceptable paths to a destination *t*, node *s* can record such changes, and update the length of affected paths in its routing table, by receiving link-state update messages. Then, a suitable path can be selected among the multiple candidate paths as the working path. For example, it could choose the path whose length is the shortest. When a path length becomes longer than the QoS requirement for end-to-end delay, that path can be marked as failed.

2). Another method to reduce the estimation error is to use admission control mechanisms at *s*, so that congestion would be avoided. As a result, the estimation error would be kept in a quite low level. In Chapter 7, we propose such an admission control mechanism when using precalculated alternate paths.

3.5 Summary

Alternate path routing mechanisms have been described in detail in this chapter. Several types of path restoration techniques which use the alternate paths were also introduced. Link restoration is the simplest of these techniques, suitable for a single link failure. Path restoration is widely used in explicit routing due to its flexibility in path calculations and efficiency in path switching at the source. P-cycle restoration is a ring protection technique, which can provide fast restoration when failures occur. However, dynamic reoptimization for p-cycle number and reconfiguration of multiple p-cycles are quite expensive.

New features when using alternate path routing are also discussed in this chapter. Alternate path routing can achieve higher responsiveness through fast path switching. However, estimation errors may be increased because the time interval between the two path group recalculations is usually much longer than that in single path routing. This means that the precalculated alternate paths may be suboptimal paths before the next path group recalculations. But, if there is little congestion in the network, or the traffic loads do not change too much, the estimation errors when using precalculated alternate paths still could be tolerated.

In the next chapter, we will present our definitions of network reliability. Also, we will illustrate different types of overlapping paths and analyze the preferred number of paths in alternate path routing, based on the tradeoff between reliability and resource consumption. Furthermore, mathematical formulae to calculate reliability when multiple paths are chosen for a given node pair will be derived.

4. New Results on Routing Reliability

Recently, a concept called *Shared Risk Link Group* (SRLG) [Cha00, Pap01] has been proposed to ensure routing diversity. SRLG is a set of optical lines sharing a common physical transmission resource (such as fiber, conduit), i.e., sharing a common risk. For instance, a set of links belongs to the same SRLG, if they are established over the same fiber segment. However, we are not interested in using SRLG to analyze reliability. We consider links or nodes shared among multiple paths, rather than a group of links sharing a common physical resource.

Improved routing reliability is one of the major potential advantages of alternate path routing, as it provides multiple paths between a given node pair *(s, t)*. We are interested in determining a small set of non-disjoint paths between *s* and *t* with a limited number of shared edges. The problems we must consider include: how reliability is defined and calculated, and how reliability is affected by the underlying alternate path finding algorithms. In this chapter, we will give definitions of network reliability and we will derive new formulae to calculate reliability in the presence of shared edges. Also, we will discuss the major factors affecting network reliability.

4.1 Introduction

Network reliability encompasses a range of issues related to the design and analysis of networks, which are subject to random component failures. In our discussion of network reliability, we mainly consider how the underlying routing protocol affects the network reliability. For a given network, the routing protocol will determine the paths, i.e., edge sets, which will be used for data transmission. Obviously, the selection of suitable edge sets has a significant influence on network reliability.

The key issues when considering reliability in alternate path routing are edge sharing and node sharing. This is because shared edges or nodes will decrease the reliability of a path set. Failures of shared edges or nodes may cause failures of multiple paths.

Only shared edges are considered in our analysis. Shared nodes can be considered as a special case of shared edges. The reasoning behind this is as follows. Assume a node v has weight w. When v is shared by multiple paths, it can be split into two vertices, v' and v'', connected by an edge (v', v'') with weight w. The node v' holds all incident edges to v, and v'' holds all out-going edges from v. By this method, a shared node can be transformed to a shared edge. Moreover, metrics of the node v, such as delay, capacity or failure probability, can also be represented as the weight of edge (v', v''). Figure 4.1 illustrates this transformation.

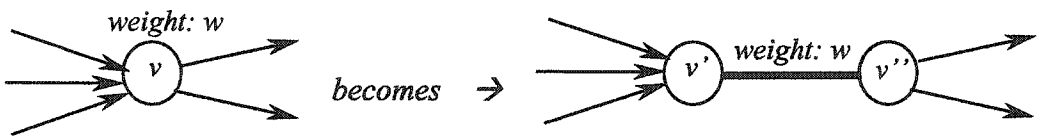


Figure 4.1. Construction of auxiliary edge and node.

In reliability analysis, network components (nodes and edges) can assume one of the two states: *operating* or *failed.* This is the so-called *two-state model.* The state of a component is by assumption a random event that is independent * of the states of the other components.

In alternate path routing, the routing reliability analysis problem can be described as follows: based on some given probabilities that each component is operating, compute a measure of reliability on the paths between a given node pair as calculated by an alternate path finding algorithm. To simplify reliability analysis in this chapter, we use the number

* This assumption can be argued in certain fiber networks. For example, links on two disjoint paths for a given node pair may share a same fiber segment. These links share a common risk and their states (operating or failed) are not independent. The concept of Shared Risk Link Group (SRLG) considers such situation. In path calculations, SRLG information can be taken into consideration [Kom02] so that the common risk will be reduced to select suitable paths.

of edges (i.e., hop count) in a path to represent the path length. In other chapters, path length represents end-to-end delay.

4.2 Definitions of Network Reliability

In the two-state model, a network component's probability of functioning correctly is called its *component reliability*. When a component fails, two types of time intervals are involved, i.e., the *mean time to failure* (called MTTF *or* T_f) and *mean time to repair* (called MTTR *or* T_r). Reliability could have one of the several possible interpretations. The most two common interpretations are:

1). the component's reliability, or

2). the component's availability.

In this dissertation, we use the term *reliability* to mean the probability that a component or system is operating. Here, we discuss a more specific definition. The term *availability* is used in the context of repairable systems. In these settings, components alternate between being in an operative state, and having failed and being currently under repair. The component availability is defined as the probability that at a random point in time the component is operating. An estimate of availability is

$$\textit{Component availability} = T_f / (T_f + T_r) \qquad (4.1)$$

The definition of component reliability does not involve consideration of repair. Rather, a length of time t is specified and the reliability of a component is defined to be the probability that the component does not fail within time t.

Components like routers and links are assumed repairable. Hence, reliability of the components is represented by the component's availability. In our analysis, the only

network components that we consider are edges (i.e., links). For simplicity, we assume that all edges have identical reliability p [*]. Failure probability of each edge is $q = (1 - p)$.

4.2.1 Network Reliability

Network reliability is defined as the probability that the network is operating (i.e., the data flows still can be correctly delivered to their desired destinations by the precalculated paths) during a specified time interval t. The following analysis of network reliability is based on literature [Bal92, Col87].

Two network reliability measures are often used. The first one is the *k-terminal* measure, represented as *Rel(s, K, G)*. Here, *G(V, E)* is the graph representing the network, and s is the source node receiving input data flows. Nodes $k = |K|$ represent the desired destinations of data packets. In a simple case, $K = (V - s)$. *Rel(s, K, G)* denotes the probability that there still exist operating paths from s to each node in K. The second measure is called *all-terminal* measure, represented as *Rel(G)*, denotes the probability that there exist operating paths between every pair of nodes in G. In this measure, any node in the network can receive data. We list definitions of the network reliability measures as follows:

Rel(s, K, G) = Pr[there exist paths from s to each node in K]

$$= \sum_{E' \subseteq E} p^{|E'|} \times (1-p)^{(|E|-|E'|)} \times S_{|E'|} \qquad (4.2)$$

Here, E' is a subset of E and includes all edges that are on every path from s to each node in set K. All paths are calculated by the alternate path finding algorithms. $S_{|E'|} = 1$ if all communication paths from s to each $v \in K$ are operational. This means that if every

[*] In a heterogeneous network, network components may have different reliability. This makes calculations of network reliability complex. Considering a simpler scenario, assume that all nodes have identical reliability p_1 and all edges have identical reliability p_2. Network components with higher reliability may be preferable in path calculations. However, calculating network reliability is still hard due to existence of common nodes and common edges. This problem is an open research topic.

edge $e \in E'$ operates, then the network operates. $S_{|E'|} = 0$ if the communication between node s to at least one node $v \in K$ fails.

$$Rel(G) = Pr[\text{there exist paths between every pair of nodes in } G]$$

$$= \sum_{E' \subseteq E} p^{|E'|} \times (1-p)^{(|E|-|E'|)} \times S_{|E'|} \qquad (4.3)$$

The definition of *Rel(G)* is similar to that of *Rel(s, K, G)*. The set E' includes all edges that are on every path between every pair of nodes in G. $S_{|E'|} = 1$ if all communication paths in the network are operational. $S_{|E'|} = 0$ if the communication between at least one pair of nodes fails.

To support the desired network reliability, explicit routing is usually required to guarantee that the calculated paths are followed at all intermediate nodes until the desired destination is reached. A source node is somewhat special; it can trace network topology changes, and perform path calculation and path switching when necessary.

However, in alternate path routing, computations of network reliability will become difficult because of the overlapping of multiple paths for each node pair. These paths can be either disjoint or non-disjoint. In this case, it is complex to calculate reliability by either *Rel(s, K, G)* or *Rel(G)*.

4.2.2 Connection Reliability

One special case of reliability is the *two-terminal* measure [Bal92, Col87] of network reliability, represented as $Rel_2(s, t, G)$. Such a measure represents the probability that data traffic can be correctly delivered from s to t. The definition of two-terminal measure of the network reliability is:

$$Rel_2(s, t, G) = Pr[\text{there exists at least an path from } s \text{ to } t]$$

$$= \sum_{E' \subseteq E} p^{|E'|} \times (1-p)^{(|E|-|E'|)} \times S_{|E'|} \qquad (4.4)$$

Here, the set E' includes all edges that are on every communication path from s to t. $S_{|E'|} = 1$ if such communication is operational, while $S_{|E'|} = 0$ if the communication between node s and t fails.

In alternate path routing domain, only a few number of paths are selected from all possible paths for the s-t node pair by path calculation algorithms. Packet forwarding in the explicit routing mode is only performed on these limited end-to-end paths. The edges/nodes which are not on the chosen paths will not be used for data delivery. Therefore, we introduce *Connection Reliability* as an estimation of routing reliability when there exist multiple paths from s to t:

Connection Reliability is the probability that at least one path from s to t is operating.

Let R represent connection reliability. R is a decreasing function of the number of common edges and of the number of edges on the paths, and an increasing function of the reliability of each edge. The common edges make the multiple paths from s to t interdependent, and the calculation of R is done using the method of inclusion/exclusion [Pu01a]. Section 4.4 gives a detailed derivation of our formulae.

In this dissertation, we use connection reliability to quantify reliability for a given node pair in alternate path routing. Without special mention, the terms reliability, routing reliability and connection reliability are interchangeable. The formulae for calculating connection reliability will be applied in our new alternate path finding algorithms for selecting suitable alternate paths that meet the desired reliability requirements.

4.3 Failure Probabilities in Networks

Failures may occur in both conventional cable-based networks and current fiber-based networks. Compared with cable-based networks, optical networks are usually more reliable due to the advanced fiber techniques. It is meaningful to estimate the failure probabilities of network components. In this section, we will do this analysis using sample data about optical networks from practice.

In an optical network, routers or switches are connected by optical transmission systems (optical links) using optical interface cards. Any of these components may fail during data transmission.

From experiences with Nortel optical networks [Bel01], almost all router failures are caused by software, and the typical value of router MTTF is about a week. Because of the software nature of router failures, the router MTTR is relatively quick (a reboot taking a few seconds to minutes). Most optical transmission system failures (link failures) are hardware failures, and almost all of those failures are caused by cable cuts. The estimation provided to us of link MTTF is in the magnitude of several years. As failures are hardware based, link MTTR is a matter of the hours required to locate and splice the broken link. For connecting routers and optical links, the interface cards (i.e., interface modules) with lasers and associated chips running at *2.5Gb/s* have a failure rate of about *1,500* failures in 10^9 hours, which means that the MTTF is about 6.67×10^5 hours or *76* years. A router would contain many such interface cards. Replacing a failed card does not require much time – it might vary from tens of minutes to several hours.

To estimate the component reliability p, we can assign sample values to MTTF and MTTF for each type of component based on above experimental data. Then, the component reliability could be computed by Formula 4.1. The component failure probability is $q = (1 - p)$. Table 4.1 shows the estimations of the q values.

Table 4.1. Estimations of failure probability to the network components.

Component	*MTTF*	*MTTR*	*Failure probability of a component*
Router	*1 week*	*40 minutes*	*0.4%*
Optical Link	*2 years*	*20 hours*	*0.2%*
Card Module	6.67×10^5 *hours*	*2 hours*	3×10^{-6}

From the above table [*], we observe that the optical links have a slightly smaller failure probability than the routers. Among the three types of components, the interface cards have the minimum failure probability.

4.4 Preferred Number of Alternate Paths

One interesting question in alternate path routing is the optimal number of alternate paths. We prefer to calculate two or three paths for a given node pair. In this section, we state the reason for this preference. For simplicity, we assume:

1). Every edge in the network is identical in length and reliability. Let p represent the reliability of an edge.

2). There are no shared edges among paths (i.e., disjoint paths).

3). All the paths have same length m (i.e., the same number of edges).

These assumptions are reasonable in practice. The number of hops in a path could be used as an estimation of the path length [Hed88, Mal98]. Moreover, in general, the number of edges shared among paths should be held to a low level, and the variance in the number of hops of the paths between two nodes is often small. Consider the relation between the connection reliability R for a given node pair (s, t) and the number of paths k. Then we have:

$$R = p^m \qquad when \quad k = 1 \qquad (4.5)$$

$$R = 1 - (1 - p^m)^k \qquad when \quad k > 1 \qquad (4.6)$$

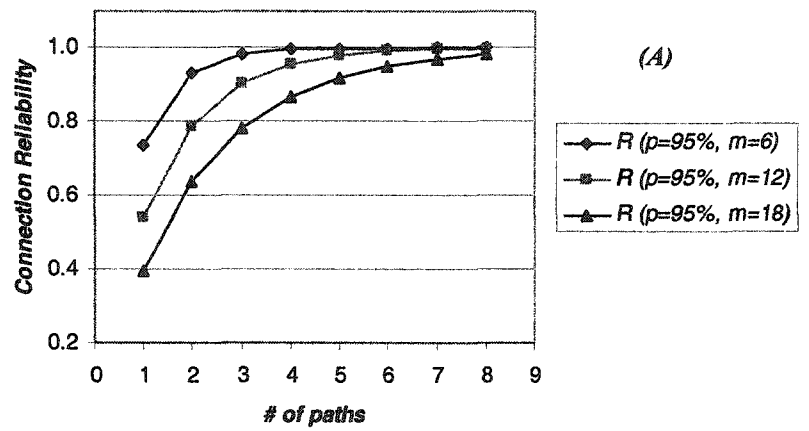
Obviously, the value of k is a measure of resource consumption. For a bigger value of k, a router has to introduce more computational overhead to calculate the desired paths,

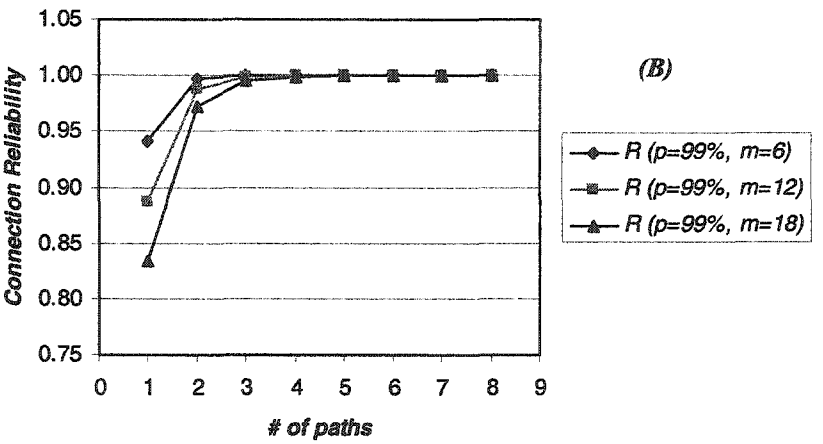
---

[*] Here, we do not consider link congestion. That is another type of problem in routing protocols. Link congestion has similar behavior to link failure. When it occurs, link length (i.e., link delay) will increase sharply. A threshold T can be used to treat congestion as a link failure when an increase of link delay exceeds the predefined T value.

and a much bigger routing table is required. When k increases, edge sharing among paths will become more complex to assess too. On the other hand, a bigger value of k means better reliability R. Therefore, our problem becomes how to best resolve the tradeoff between better reliability and excessive resource consumption.

Figures 4.2A, 4.2B and 4.2C illustrate the behavior of R as a function of k, p and m. In case A, we fix p = *95%* and set path length m as *6, 12* and *18*. In case B, we fix p = *99%* and also set path length m as *6, 12* and *18*. In case C, we fix m = *12,* and set p as *95%, 97%* and *99%*. In these figures, connection reliability R increases faster when k is smaller but grows slowly when k is larger. Put another way, the incremental improvement in reliability diminishes rapidly as k increases.





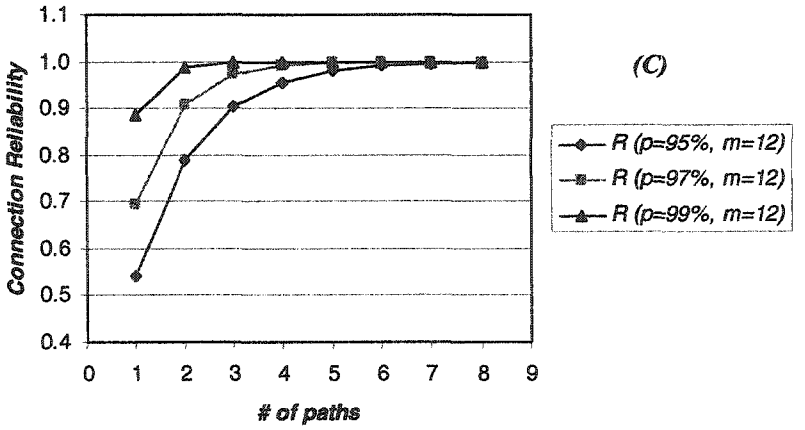


Figure 4.2. The relation between connection reliability and the number of path.

Here, we apply the *20-80 Rule* [*] [Koc98], an empirical rule of thumb in engineering and economics, to choose the optimal number of paths in alternate path routing. From Figure 4.2C, when path length is moderate ($m = 12$), and the edges are not very reliable ($p = 95\%$), the minimum reliability is $R_{min} = 54\%$ when $k = 1$. Assume the maximum reliability is $R_{max} = 100\%$ for a very large k. From the curve, $k = 3$ achieves $R' = 90.3\%$ reliability. On the other hand, according to the *20-80* rule, the expected reliability based on the tradeoff between reliability and the number of paths is $R'' = R_{min} + 80\% \times (R_{max} - R_{min}) = 90.8\%$. Hence, $R' \approx R''$ is a valid assumption, and we can say that $k = 3$ may be the best choice to balance the achieved reliability and demanded resource consumption. The same analysis can be applied to the case when $p = 97\%$ and $m = 12$ in Figure 4.2C.

---

[*] Vilfredo Pareto (1848-1923) was an Italian economist who, in 1906, observed that twenty percent of the Italian people owned eighty percent of their country's accumulated wealth. Later, this rule has come to be called Pareto's Principle [Koc98], or the *20-80* Rule. It states that a small percentage of effort, is responsible for a large percentage of the resulting effect, often in a ratio of approximately *20:80*. His rule has been applied as an empirical rule of thumb in engineering and economics - a minority of input produces the majority of results. The improvement of reliability with increase in k seems to be an example of Pareto's classic idea, as are so many other cases of engineering design.

However, in Figure 4.2B, when the edges are quite reliable *(p = 99%),* the reliability *R'* achieved by two paths *(k = 2)* is almost equal to the corresponding *R''*. In this scenario [♠], according to the *20-80* rule, *k = 2* is a better choice than *k = 3*.

Here, we do not consider shared edges amongst the paths. Shared edges will obviously reduce the value of *R*. However, for achieving high routing reliability, only a very limited number of shared edges can be allowed among the calculated alternate paths. For simplicity and brevity, we apply the above analysis to such non-disjoint alternate paths. Thus, the preferable value of *k* is *2* or *3*.

Furthermore, for the case of *k > 3,* edge sharing will become more complex. An edge can be shared by four paths, five paths and so on. The complexity of an alternate path finding algorithm when considering such types of edge sharing may significantly increase. Also, more alternate paths will require a much bigger routing table. The third drawback is that estimation errors (discussed in Section 3.4.2) may increase, because the time interval between the two path group recalculations will become longer when more alternate paths exist. These alternate paths may not match the latest network topology changes.

Setting *k = 3* can be used as the upper bound on the number of precalculated paths for a given node pair. The underlying alternate path finding algorithm may also affect the number of alternate paths that can be calculated. For certain network topologies, it is possible that only one alternate path can be calculated.

Therefore, by such selection of path number *(k = 2, 3)*, the introduced computational overhead and additional storage requirement during the routing process are not very significant, and these costs are tolerable in the current router technologies.

Finally, it is interesting to recall that alternate path routing was first used in the voice switching networks, and there the value of *k* was always *2,* one primary route and one alternate route.

[♠] It is possible that, in an optical network, the reliability of network components could be *99%* (or better). We have discussed this in Section 4.3.

4.5 Calculation of Connection Reliability

In this section, we will closely examine the important role of shared edges in the calculation of connection reliability *R*. Shared edges make the paths interdependent, because if a shared edge fails, multiple paths will be affected. In alternate path routing, the path finding algorithms will calculate two or three paths (depending on the user's specification) for each pair of nodes. Therefore, shared edges can be classified as *double-shared edges* that are common edges used by two paths, or *triple-shared edges* that are common edges used by all three paths (if there are three paths).

4.5.1 Edge Sharing of Alternate Paths

Let us consider the case of three communication paths for a given node pair *(s, t)*. It may be common that a few shared edges exist among the three paths from *s* to *t* after an alternate path finding algorithm is executed at node *s*. Figure 4.3 illustrates several sample cases of edge sharing. When only two communication paths exist from *s* to *t*, edge sharing becomes much simpler, there are no triple-shared edges.

For convenience, we introduce the following notation for our mathematical analysis. Let m_1, m_2 and m_3 represent the length (here, the number of edges) of paths P_1, P_2 and P_3 respectively. Each path involves three types of edges: unshared edges (also called independent edges), double-shared edges and triple-shared edges, as shown in Figure 4.4.

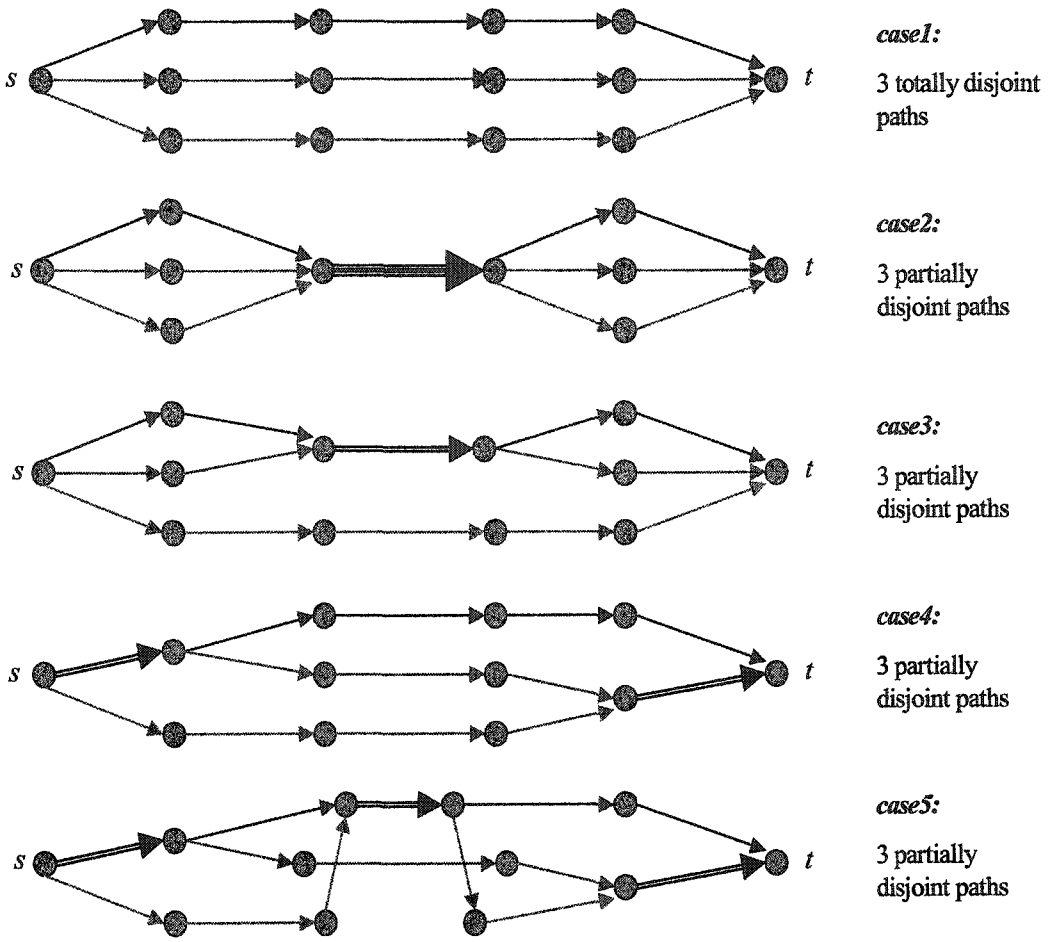


Figure 4.3. Sample cases of edge sharing among three paths.

We define:

the length of path P_1: $m_1 = L_1 + L_{12} + L_{13} + L_{123}$ *(4.7)*

the length of path P_2: $m_2 = L_2 + L_{12} + L_{23} + L_{123}$ *(4.8)*

the length of path P_3: $m_3 = L_3 + L_{13} + L_{23} + L_{123}$ *(4.9)*

where,

L_i, the number of unshared edges used only by path i; $i = \{1, 2, 3\}$

L_{ij}, the number of double-shared edges used by paths i and j; $i, j = \{1, 2, 3\}$, $i \neq j$

L_{123}, the number of triple-shared edges used by all three paths

Furthermore, assume every edge has identical failure probability q. Then, every edge has identical reliability $p = (1 - q)$. The failure probability of nodes is omitted in our calculation of connection reliability.

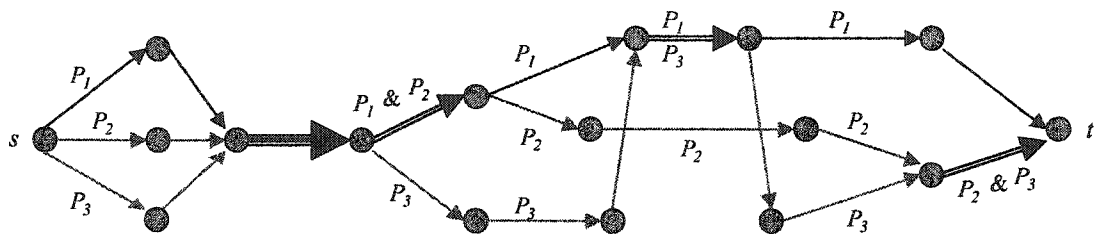


Figure 4.4. Double- and triple-shared edges among three paths from s to t.

4.5.2 Formulae to Compute Connection Reliability

If there are two paths (P_1 and P_2) from s to t, then only double-shared edges among P_1 and P_2 may exist. Using the notation described in the last section, we can quickly obtain the following formula to calculate the connection reliability R for this scenario:

$$R = p^{L12} \cdot [1 - (1 - p^{m1-L12}) \cdot (1 - p^{m2-L12})] \qquad (4.10)$$

Calculations will become complex when the number of paths increases to three. Edge sharing has three cases: independent, double-shared and triple-shared edges (shown in Figures 4.3 and 4.4). Certain mathematical methods of inclusion/exclusion have to be used to calculate R. Here, we derive the following formulae for three non-disjoint paths by applying the 3-D Venn Diagram model [Fre80], a technique to calculate occurrence probability of three interdependent events using inclusion/exclusion. The detail steps of this derivation are described in Appendix A. From Equations A.12-A.16, we have

$$R = S_0 - S_1 + S_2 \qquad (4.11)$$

Here,

$$S_0 = p^{m1} + p^{m2} + p^{m3} \qquad (4.12)$$

$$S_1 = p^{(m1+m2)-(L123+L12)} + p^{(m1+m3)-(L123+L13)} + p^{(m2+m3)-(L123+L23)} \quad (4.13)$$

$$S_2 = p^{(m1+m2+m3)-(2*L123+L12+L13+L23)} \quad (4.14)$$

In some special cases, simplified formulae can be obtained to calculate the connection reliability R, as shown below (i.e., Equations A.17-A.19 in Appendix A).

(1). The three paths (P_1, P_2 and P_3) are disjoint (case 1 in Figure 4.3)

$$R = 1 - (1 - p^{m1}) \cdot (1 - p^{m2}) \cdot (1 - p^{m3}) \quad (4.15)$$

(2). Only triple-shared edges exist among P_1, P_2 and P_3 (case 2 in Figure 4.3)

$$R = p^{L123} \cdot [1 - (1 - p^{m1-L123}) \cdot (1 - p^{m2-L123}) \cdot (1 - p^{m3-L123})] \quad (4.16)$$

(3). Only double-shared edges between P_1 and P_2 exist (case 3 in Figure 4.3)

$$R = 1 - (1 - p^{m3}) \cdot \{1 - p^{L12} \cdot [1 - (1 - p^{L1}) \cdot (1 - p^{L2})]\} \quad (4.17)$$

4.5.3 Sample Calculation Results

Formulae 4.11-4.17 were used in calculating the results shown in Figures 4.5-4.7. Figure 4.5 shows connection reliability comparisons when only triple-shared and only double-shared edges exist among the three paths (cases 2 and 3 in Figure 4.3). It illustrates that the overall connection reliability R decreases as the number of triple- or double-shared edge increases, which agrees with intuition.

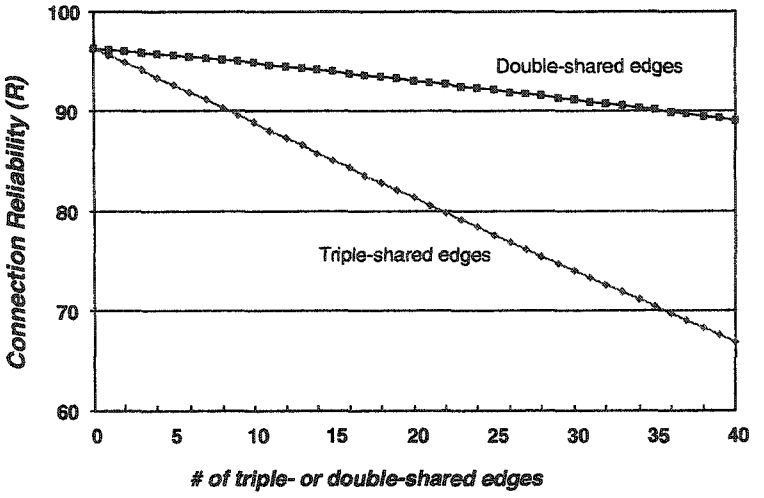


Figure 4.5. The relation between R and shared edges (path length = 40, p = 99%).

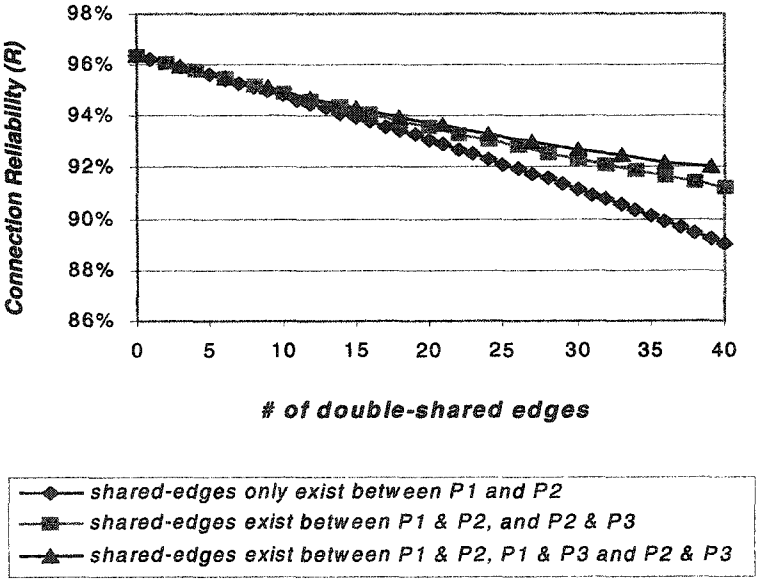


Figure 4.6. Connection reliabilities under different types of double-shared edges.

Figure 4.6 shows the variation of the R for different combinations of the double-shared edges (cases 3, 4 and 5 in Figure 4.3). These variations are relatively minor. When only double-shared edges exist, even distribution of the double-shared edges among the three paths will cause a slightly higher connection reliability R compared to the case

where the double-shared edges are mainly upon only two of the paths. In the calculation, all path lengths are fixed as *40* and reliability of every edge $p = 99\%$.

Figure 4.7 illustrates the relation of R to single edge reliability p when only triple-shared edges exist among the three paths (case 2 in Figure 4.3 with different *p).* All path lengths are fixed at *20* edges.

Obviously, a triple-shared edge is a throughput bottleneck and a critical link to the three partially disjoint paths. If possible, we should avoid picking three paths with triple-shared edges from the candidate path set.

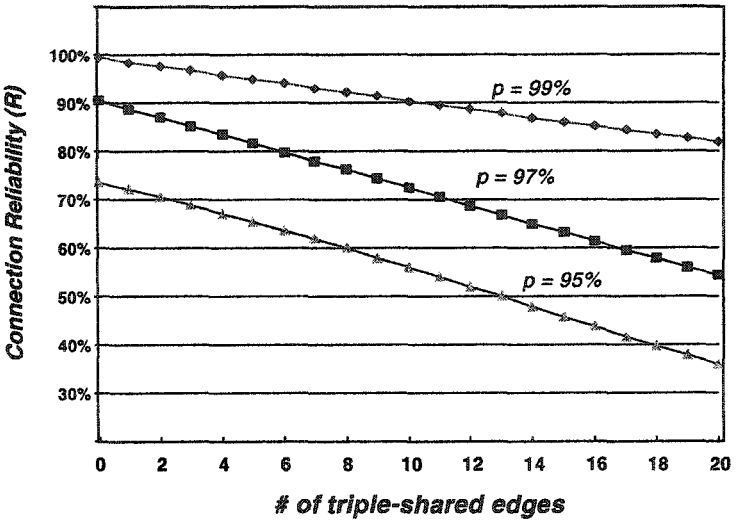


Figure 4.7. Relation of R and edge reliability p when only triple-shared edges exist.

4.6 Summary

In this chapter, we have presented the definitions of network reliability. Network reliability has been analyzed through three types of measures: the *k-terminal* measure *Rel(s, K, G), all-terminal* measure *Rel(G)* and *two-terminal* measure $Rel_2(s, t, G)$.

In alternate path routing, multiple end-to-end paths are used to deliver data packets for a given node pair *(s, t).* We introduce a concept of connection reliability, the probability that at least one path from s to t is operating. Connection reliability is a

function of edge reliability, the number of edges on the paths, and the number of edges shared among the paths. In this dissertation, routing reliability is quantified by using the concept of connection reliability.

In the presence of three partially disjoint paths between a given node pair in a network, we used the 3-D Venn Diagram to model the connection reliability. The mathematical formulae to calculate such reliability were derived. In an alternate path finding algorithm, these formulae are useful and essential to calculate R as a constraint for deciding whether a candidate path is acceptable or not.

In the next chapter, we will propose new alternate path finding algorithms based on the reliability analysis in this chapter. The new algorithms can efficiently calculate suitable alternate paths subject to predefined constraints on reliability demand and QoS guarantees, such as end-to-end delay and bandwidth requirement.

5. New Alternate Path Finding Algorithms

In the last chapter, the formulae for calculating connection reliability (i.e., reliability in our context) were derived. In this chapter, we will apply those formulae in alternate path finding algorithms. From the review of previous work in Section 2.4, existing path finding algorithms may not quite fit the needs of our reliable routing using non-disjoint alternate paths. To propose new efficient algorithms that consider reliability when calculating suitable alternate paths is necessary.

5.1 A New Alternate Path Finding Algorithm Optimizing Reliability

In this section, a new alternate path finding algorithm called OptAlt is proposed to calculate optimal alternate paths for a given node pair, by maximizing reliability subject to desired QoS guarantees on end-to-end delay and bandwidth.

5.1.1 Problem Definition

As stated in Section 4.2.2, the reliability R is the probability that there is a connection from a given source s to destination t. The selected multiple paths are called a *path group* (or *path set),* for which the reliability will be maximized. We define the *Reliability-optimal Alternate Path Finding Problem,* the problem of selecting the optimal path group, as follows:

Objective:

Find a path set (two or three paths) from s to t with maximum reliability R.

Constraints:

1). There must be at least B_{min} available bandwidth on each link of each of the paths.

2). Each path length (taken here to represent latency, i.e., end-to-end delay) is at most the specified maximum end-to-end delay value L_{max}.

5.1.2 The OptAlt Algorithm

Alternate path calculations done at *s* require a global view of the entire network, which can be provided by the underlying routing protocol QOSPF [Apo99b]. To calculate suitable alternate paths with optimal reliability from *s* to *t*, we develop a new *Optimal Alternate path finding algorithm* (called OptAlt) based on the problem definition stated in Section 5.1.1. Figure 5.1 illustrates its pseudo code.

// Network topology and link load information are obtained from the link-state
// database provided by QOSPF.

1). *Prune the links whose available bandwidth are less than the least desired bandwidth* B_{min} *;*
2). *Use the MPS K-shortest paths algorithm [Mar99] to find* ***all*** *acceptable paths whose lengths are less than the specified maximum end-to-end delay* L_{max}*. Put all these paths in a set T;*
3). *Consider all combinations of k paths from the path set T. Each combination is called a path group, and it contains k paths, where k is a prespecified number indicating the group size (k = 2, 3);*
4). *For each path group, calculate connection reliability R. Select the path group with the maximum R as the result.*

Figure 5.1. Pseudo code for the OptAlt alternate path finding algorithm.

OptAlt is a brute-force algorithm. The time complexity of the OptAlt algorithm can be analyzed from the above pseudo code. Assume that the network contains *M* links and *N* nodes. Let *H* represent the size of the set *T* containing all acceptable paths. Step *1* will take $T_{link\text{-}pruning} = O(M)$ time, and step *2* will take $T_{MPS} = O(MlogN + HN)$ [Mar99] time to calculate all *H* candidate paths that meet the end-to-end delay constraint. In steps *3* and *4*, the number of possible combinations (i.e., path groups) is ${}_HC_k$. Hence, the time needed to finish steps 3 and 4 is $T_{grouping} = O({}_HC_k)$. The entire time complexity of the OptAlt algorithm is:

$$T_{OptAlt} = T_{link\text{-}pruning} + T_{MPS} + T_{grouping} = O(M) + O(MlogN + HN) + O({}_HC_k)$$

$$= O(MlogN + HN + H^2) \qquad when\ k = 2$$

$$= O(MlogN + HN + H^3) \qquad when\ k = 3 \qquad (5.1)$$

The reasoning for optimality of our OptAlt algorithm is as follows. By the link-pruning process, step 1 in the algorithm shown in Figure 5.1, all remaining links after the link-pruning process will contain at least B_{min} bandwidth. After step 2, the K-shortest paths algorithm, all H acceptable paths from s to t meeting the end-to-end delay constraint are calculated. Then, in step 3, there are in total ${}_HC_k$ possible candidate path groups composed of the H candidate paths. Step 4 will choose a path group with maximum R from all possible candidate groups. Therefore, after these four steps, OptAlt can guarantee that the selected path group will be an optimal solution with maximum reliability.

In some cases, multiple path groups may have the same maximum reliability. In this case, we can pick the preferable group by the following rules:

1). A group containing fewer triple-shared edges is preferable.

2). Else, a group containing fewer double-shared edges is preferable.

3). Else, a group with shorter primary path is preferable.

4). Else, a group with shorter alternate paths is preferable.

5). Else, a group with more available bandwidth on the primary path is preferable.

6). If still tied, then we pick one group at random.

5.1.3 Analysis of the OptAlt Algorithm

From Formulae 4.11-4.14 for calculating reliability R, we know that R is a function of the number of edges on paths and the number of edges shared among paths. In OptAlt, we cannot predict or control the number of shared edges in the selected optimal path group. However, a network provider or a user may want to specify a particular number of shared links for the preferable alternate paths. For example, a remote medical operation

in Montréal may ask for a connection to a medical center in Los Angels, by three paths with at most two double-shared links and no triple-shared links. This shows that the number of shared edges can also be used to indirectly represent the influence of reliability.

Another drawback of new proposed OptAlt is that the number of candidate path groups will increase sharply when there are more acceptable paths. For example, let each group contain three paths, and there are *6* acceptable paths calculated by the *K*-shortest paths algorithm. We then have to check ${}_6C_3 = 20$ candidate path groups. If there are *15* such paths, the number of candidate path groups will increase to ${}_{15}C_3 = 455$. As a result, computational overhead will increase significantly.

Moreover, the shortest path for a given node pair *(s, t)* may not be included in the optimal path group selected by OptAlt. However, in some cases, the shortest path may be preferred by the user for its features – short, reliable and compatible with the currently popular shortest path routing strategy. In alternate path routing, the primary path will be used most of the time, whenever the failure probability is not too high. This is true of current optical networks.

Hence, we need to develop heuristic algorithms to reduce computational overhead in alternate path calculations under various path selection constraints. Also, the shortest path is contained in the chosen path group.

5.2 Constraints for Acceptable Alternate Paths

Like Section 5.1.1, let L_{max} represent the maximum acceptable end-to-end delay from *s* to *t*. Let B_{min} be the demanded bandwidth. Let DSE_{max} and TSE_{max} represent the maximum number of acceptable double- and triple-shared edges among paths, respectively. Obviously, there are no triple-shared edges if we only calculate two paths (i.e., the group size $k = 2$). Without loss of generality, we assume here that we will calculate three acceptable paths: one primary path, P_0, and two alternate paths, P_1 and P_2. A heuristic alternate path finding algorithm should calculate an acceptable path group, in which each path P_i, $i = 0, 1, 2$, must satisfy the following constraints:

1). Available capacity of each link on path P_i is at least B_{min}.

2). P_0 is a shortest path from s to t.

3). Length of path P_i is at most L_{max} for all i.

4). Number of edges (i.e., edge count or hop count) of P_i is at most H_{max}.

5). Number of double-shared edges among all calculated paths is at most DSE_{max}.

6). Number of triple-shared edges among all calculated paths is at most TSE_{max}.

The users can specify all of these path selection constraints or some of them in path calculations. For the first constraint, the link-pruning process in OptAlt can be used in the proposed heuristic path finding algorithms to meet the demanded bandwidth, i.e., the links with insufficient bandwidth are pruned.

The alternate paths that are calculated by the heuristic path finding algorithms subject to predefined path selection constraints are termed the *acceptable alternate paths.*

5.3 Calculating Acceptable Alternate Paths

As discussed before, the connection reliability for a node pair *(s, t)* is a function of the number of shared edges among paths and the edge counts in the paths. There is a tradeoff between these two types of parameters. For achieving the required reliability, an alternate path finding algorithm may emphasize either aspect: calculating longer alternate paths with fewer shared edges, or shorter alternate paths with more shared edges.

The proposed OptAlt algorithm can calculate optimal path group with fewer shared edges and shorter path lengths, but it has high time complexity. Hence, we propose three faster new algorithms to calculate the acceptable alternate paths subject to the constraints described in Section 5.2.

In certain networks, there may exist multiple equal-length shortest paths for a given node pair. If this is the case, we can choose one containing the minimum edges as the primary path, as it minimizes system costs. If several equal-length shortest paths have the same minimum edges, we break the tie by a random choice among the paths. After the

primary path is selected, the left equal-length shortest paths can be used as acceptable alternate paths if they meet the predefined constraints. However, if we still cannot find enough paths, the alternate path finding algorithms have to be used.

5.3.1 An Existing Algorithm to Calculate Alternate Paths

In this section we look at an existing alternate path algorithm that is closely related to our work.

Using multiple alternate paths, Lee and Gerla proposed an approach [Lee01] for improving fault tolerance in QoS provisioning. This approach was briefly descried in Section 2.4. They developed an alternate path finding algorithm by extending Cavendish's single path solution [Cav98], which is based on Bellman-Ford's shortest path algorithm [Cor90a], to calculate suitable alternate paths for a given node pair *(s, t)* subject to multiple QoS constraints. For simplicity, we denote Lee and Gerla's algorithm as L&G in this dissertation.

L&G selects acceptable alternate paths from all possible paths by a BFS-like (Breadth First Search) method when increasing the hop number of temporal paths (described by *QoS descriptors* [Lee01]) starting from source *s*. Such paths are extended in a hop-by-hop manner. If destination *t* is reached and the complete path from *s* to *t* satisfies the path selection constraints, then this path is an acceptable path. The time complexity of L&G with N nodes is:

$$T_{L\&G} = O(N^3) \qquad (5.2)$$

One benefit of L&G is that it can optimize the number of hops of the calculated alternate paths. However, L&G is complex and inefficient for its high time and space costs. The algorithm may often have to calculate and store large number of valid temporal paths from *s* to every intermediate node, which are costly operations.

In the following subsections (Sections 5.3.2-5.3.4), three new algorithms are developed to speed the calculations of the acceptable alternate paths.

5.3.2 An Extension to the MPS *K*-Shortest Paths Algorithm (EMPS)

The *K*-shortest paths algorithms calculate a set of paths for a given node pair, which are ranked by path lengths in ascending order. However, path overlapping is omitted in the *K*-shortest paths algorithms.

Among the numerous *K*-shortest paths algorithms, we choose Martins' MPS algorithm [Mar99] for its ease of implementation and absence of complex data structures. MPS uses a deviation approach to calculate paths in a directed or undirected graph. It calculates and ranks the *deviation paths* ♣ between a given node pair based on the shortest path. Appendix D presents pseudo code of the MPS algorithm in detail.

To calculate acceptable alternate paths, we can extend the MPS *K*-shortest paths algorithm by considering common edges among paths, based on the path selection constraints. Two extra processes in our new algorithm considering demands of QoS guarantees and reliability requirements are added. The first process is the *link-pruning process,* which deletes the links with insufficient bandwidth. The second process is the *path-checking process,* which determines whether the calculated i^{th} *(i = 2, 3... K)* shortest path is acceptable, subject to the constraints of path lengths and the number of shared edges among the i^{th} path and the already selected paths.

Our alternate path finding algorithm is an extension of the MPS algorithm. We call it EMPS. Assume that we want to compute three paths (P_0, P_1 and P_2) from *s* to *t* by EMPS, where path P_0 is the shortest path from *s* to *t*. The major steps to calculate the two alternate paths (P_1 and P_2) are shown in Figure 5.2.

// the link-pruning process

1). Prune the links with insufficient available bandwidth .

2). Set the number of already found alternate paths numOfAltPath = 0 .

3). Calculate the i^{th} shortest path P_i by MPS .

♣ Let node *v* be an intermediate node of the i^{th} shortest path $P_i(s, t)$. Assume path $P(v, t)$ is a new shortest path among all possible paths from *v* to *t*, $P(v, t) \not\subset P_i(s, t)$. A deviation path $P'_i(s, t)$ of $P_i(s, t)$ at node *v* is a concatenated path: $P'_i(s, t) = P_i(s, v) \,||\, P(v, t)$. Here, node *v* is called a deviation node of path $P_i(s, t)$.

// the path-checking process, (if i = 1, increase i by one and repeat step 3)

4). Perform the path-checking process to determine whether P_i is acceptable or not, subject to the constraints of path lengths and the number of shared edges .

5). If P_i is acceptable, then we have found an alternate path and we increase numOfAltPath by 1 .

6). If numOfAltPath == 2, then the algorithm stops .

7). Repeat from step 3 .

Figure 5.2. Pseudo code for the EMPS alternate path finding algorithm.

Step *4* in EMPS performs the path-checking process for each new path calculated by MPS. For the first alternate path P_1, there are only double-shared edges between P_0 and P_1. Triple-shared edges may exist only when we calculate the second alternate path P_2.

Here, we examine the time complexity of the EMPS algorithm. The link-pruning process will take $T_{link\text{-}pruning}$ time. Let H represent the number of the shortest paths (calculated by MPS) that meet the end-to-end delay constraint. The path-checking process introduces an extra $T_{path\text{-}checking} = O(H)$ time cost in the calculation. Therefore, the total time cost of the EMPS algorithm T_{EMPS} is:

$$T_{EMPS} = T_{link\text{-}pruning} + T_{MPS} + T_{path\text{-}checking} = O(M) + O(MlogN + HN) + O(H)$$
$$= O(MlogN + HN) \qquad (5.3)$$

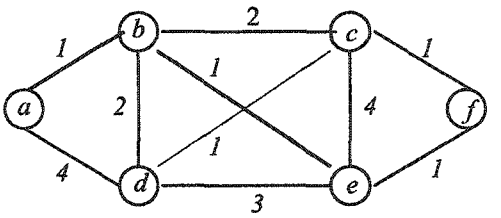
In EMPS, the acceptable alternate paths are sequentially selected from the ranked H shortest paths. Shared edges among paths are used as the path selection constraints. Hence, we can say that EMPS considers the path length at first, then the shared edges. In other words, EMPS finds the acceptable alternate paths that are as short as possible.

Using a sample weighted graph shown in Figure 5.3A, we illustrate major steps (Figure 5.3B) of how to select three acceptable paths from source node *a* to destination node *f* by EMPS. In this example, we assume that each link has sufficient bandwidth, and the path selection constraints are:

1). The maximum number of double-shared edges $DSE_{max} < 3$.

2). There are no triple-shared edges (i.e., $TSE_{max} = 0$).

3). The maximum path length $L_{max} < 7$.



A). The sample weighted graph

B). Steps to select the acceptable alternate paths

Path calculated by MPS	*Complete path*	*Path length*	*The number of double-shared edges*	*The number of triple-shared edges*	*Acceptable?*
The 1st shortest path	*a→b→e→f*	*3*	*0*	*0*	*Yes (Path P_0)*
The 2nd shortest path	*a→b→c→f*	*4*	*1 (a→b)*	*0*	*Yes (Path P_1)*
The 3rd shortest path	*a→b→d→c→f*	*5*	*1 (c→f)*	*1 (a→b)*	*No*
The 4th shortest path	*a→d→c→f*	*6*	*2 (a→b) and (c→f)*	*0*	*Yes (Path P_2)*

Figure 5.3. Major steps to calculate the acceptable alternate paths by EMPS.

5.3.3 An Edge Weight Increasing Alternate Path finding Algorithm (EWI)

The *Edge Weight Increasing alternate path finding algorithm,* called EWI [Pu01b], is another new heuristic algorithm we developed. EWI is intended to calculate acceptable alternate paths between two nodes that will have as few shared edges as possible. This algorithm considers the shared edges at first to maximize the diversity among the paths (i.e., minimize the shared edges), and then the path lengths are applied as the additional path selection constraints.

In Dijkstra's algorithm [Dij59], if we set the weight $w(i, j) = INFINITY$ for an edge (i, j), then this edge will not be added to the shortest path P_{st}. Tanenbaum [Tan88] indicates that alternate paths may be obtained by deleting all edges of path P_{st} and recalculating a

new shortest path P'_{st} by reinvoking Dijkstra's algorithm. This is equivalent to setting $w(i, j) = INFINITY$.

Heuristically, a new shortest path P'_{st} will have fewer common edges * with the previous shortest path P_{st} if P'_{st} is recalculated after incrementing every edge's weight of P_{st} by a larger value ΔW. Such value can be treated as a penalty upon each edge on the previously selected path. Rouphail [Rou95] developed a pretrip route-planning scheme by providing tourists alternate routes using similar ideas. After the shortest path P_0 was calculated, he increased each edge's weight in P_0 by *20%, 50%* and *100%* of its original weight. The best alternate route for the tourists could then be chosen from the candidate paths as recalculated by Dijkstra's algorithm.

Our new algorithm also applies this concept of edge weight penalty (ΔW) and calculates alternate paths by recursive invocation of Dijkstra's algorithm. However, ΔW is updated (increasing or decreasing) logarithmically for speeding calculation.

Let P_0 be the shortest path from s to t. We start with a large $\Delta W = W_0$ on P_0 to ensure a minimum number of shared edges between a new calculated shortest path P_1 and the existing P_0. If P_1 does not meet the desired constraints, ΔW is decreased by half $(W_0 / 2)$, and then a new P'_1 can be calculated. If P'_1 fails the tests again, ΔW will be further decreased.

On the other hand, if P_1 satisfies the desired constraints, then P_1 is a candidate alternate path. In this case, we will increase ΔW by $W_0 / 4$, and then calculate a better P'_1 ♥ with fewer shared edges. Such increasing and decreasing operations are similar to a binary-search method shown in Figure 5.4. When P'_1 and P_1 calculated using different ΔW are the same, or the change of ΔW is less than a predefined value, we stop and pick the previously candidate path as the acceptable alternate path. By the same steps, the second alternate path P_2 can be calculated based on the previously calculated P_1 and P_0.

---

* There is one exception. If an edge (i, j) is a cut edge connected nodes s and t, this edge will appear in both paths P_{st} and P'_{st}.

♥ The key point in EWI is to find a biggest edge weight penalty ΔW $(0 < \Delta W \leq W_0)$, so that the newly calculated shortest path may have the fewest shared edges with the previously selected paths. Whether the new path is acceptable is determined by the path-checking process.

```
// These two functions will return a new edge weight penalty ΔW' ( 0 < ΔW' ≤ W0 ) .
// ΔW is the old edge weight penalty
// p and r are the lower bound and higher bound of adaptive edge weight penalty, respectively.
// In initialization, we set p = 0, r = W0 .

increase_edgeweight_penalty(p, r, ΔW)
{
    If (ΔW == r)
        Stop ;
    else {
        ΔW' = ΔW + (r - ΔW) / 2 ;
        p = ΔW ;
    }

    return (ΔW') ;
}

decrease_edgeweight_penalty(p, r, ΔW)
{
    If (ΔW == r)
        ΔW' = ΔW / 2 ;
    else {
        ΔW' = ΔW - (ΔW - p) / 2 ;
        r = ΔW ;
    }

    return (ΔW') ;
}
```

Figure 5.4. Update edge weight penalty Δw in a binary-search method.

In EWI, the initial value of ΔW is set to $W_0 = \sum w_i$, the sum of the weights of all edges in the undirected graph (or arcs in the directed graph). For example, in Figure 5.5, the primary shortest path P_0 between nodes a and c is $a \rightarrow c$. If we choose ΔW as the maximum edge weight in the graph, i.e., $\Delta W = 3$, the next shortest path P_1 will still be $a \rightarrow c$ with length 4. Therefore, ΔW has to be much bigger. If edge (a, c) increases its weight by $\Delta W = W_0$ (here, $W_0 = \sum w_i = 6$), then we can get the desired alternate path P_1 $(a \rightarrow b \rightarrow c)$. Obviously, if we choose $\Delta W >> W_0$, the calculation for P_1 is exactly as same as choosing $\Delta W = W_0$.

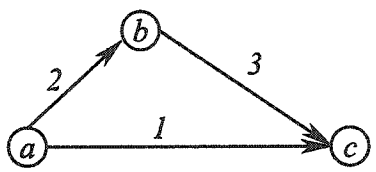


Figure 5.5. Choose the appropriate initial value ΔW.

EWI calculates alternate paths in a series of phases. In every phase, we keep probing one valid path by the *edge-weight-increment* process, in which ΔW logarithmically dithers between W_0 and 0. Every path calculated by Dijkstra's algorithm is tested to see if

it meets the desired constraints to determine whether it is acceptable. At most one alternate path is generated within each phase. When there are multiple acceptable paths found in a phase, the path calculated by the larger value of ΔW is preferable. Therefore, the final path set (three paths) would include a minimum number of common edges. The pseudo code of the new EWI algorithm is illustrated in Figure 5.6.

```
prune links whose available bandwidth are less than the least desired bandwidth B_min ;
calculate the primary shortest path P_0 by Dijkstra's shortest path algorithm ;
numOfAltPath = 0 ;
P_cur = P_0 ;              // P_cur – the working path where edge-weight-increment is performed
ΔW = W_0 ;                 // W_0 = Σw_i
p = 0, r = W_0 ;
repeat until (numOfAltPath == 2) {
FINDING_NEW_ALTERNATIVE :                          // a label for goto statement
    Q = Φ ;                // Q is a path set to store all candidates when seeking an alternate path
    P_alt = null ;         // P_alt represents one acceptable alternate path found
    repeat {
SEARCHING_BETTER_CANDIDATE :                       // a label for goto statement
        increase all edges' weight on P_cur by ΔW ;
        recalculate a new shortest path P_new by Dijkstra's algorithm ;
        decrease all edges' weight on P_cur by ΔW ;        // restore link weights on P_cur
        ΔW' = ΔW ;                                         // save the value of this ΔW
        if (P_new satisfies all constraints) {             // P_new is an acceptable candidate
            if (ΔW < W_0) {
                ΔW = increase_edgeweight_penalty(p, r, ΔW) ;        // increase ΔW
                Δ = |ΔW - ΔW'| ;                           // Δ - the change of ΔW values
                // Testing: Repeat calculations of seeking acceptable candidates?
                if ((Δ > T) and (P_new ∉ Q)) {             // T is a preset threshold
                    put P_new into the path set Q ;
                    goto SEARCHING_BETTER_CANDIDATE ;
                }
            }
            P_alt = P_new ;                                // we find an alternate path
        }
        else {                                             // P_new is not acceptable
            ΔW = decrease_edgeweight_penalty(p, r, ΔW) ;        // decrease ΔW
            Δ = |ΔW - ΔW'| ;                               // Δ - the change of ΔW values
            if (Δ > T)                  // T – threshold, an end condition of path seeking
```

```
            goto SEARCHING_BETTER_CANDIDATE ;
        else                                          // we still find an alternate path
            P_alt = pick the path calculated by maximum ΔW from set Q ;
    }
    if (P_alt is not null) {                          // P_alt is the alternate path we need
        numOfAltPath ++ ;
        // impose the maximum edge weight penalty on old working path P_cur
        reset ΔW = W_0 ;
        increase all edges' weight on P_cur by ΔW ;
        P_cur = P_alt ;                  // set the newly selected alternate path as working path
        goto FINDING_NEW_ALTERNATIVE ;   // start to calculate a new path
    }
  }
}
```

Figure 5.6. Pseudo code for the EWI alternate path finding algorithm.

Figure 5.7 illustrates how EWI works in a sample network. Nodes s and t are the source and the destination. The total weight of all arcs in the graph is $W_0 = \sum w_i = 20$. Assume that the only constraint on the alternate paths is $H_{max} = 3$ (i.e., the number of edges on an alternate path is at most *3)*. Other constraints, such as path length and shared edges are omitted for simplicity.

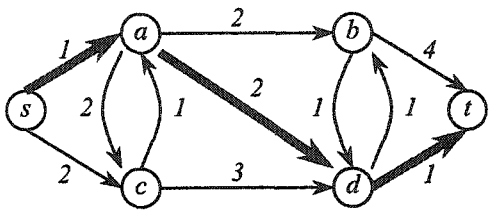
Step 1. In case A, a shortest path P_0 *(s→ a→ d→ t)* is calculated and it is selected as the primary path. The length of P_0 is *4.*

Step 2. In case B, every edge on P_0 has its weight increased by $\Delta W = W_0 = 20$ *(i.e.,* $w_i' = w_i + \Delta W$*).* Then, we reinvoke Dijkstra's algorithm to obtain the 2nd path P_1 *(s→ c→ a→ b→ t).* The length of P_1 is *9.* However, P_1 has *4* edges, which violates the constraint: $H_{max} = 3$. Therefore, path P_1 has to be discarded, although it has *0* common edges with P_0.

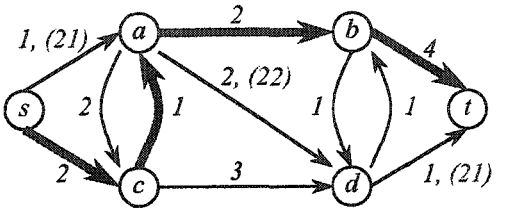
Step 3. In case C, we now try to logarithmically reduce ΔW on path P_0 and recalculate the next shortest path. When $\Delta W = 2$, the second acceptable path P_2 *(s→ c→ d→ t)* can be obtained. P_2 has three edges and has a real path length of *6.* Because no

other acceptable paths can be found for any $\Delta W > 2$, P_2 is chosen as the first alternate path, although it has one common edge with P_0.

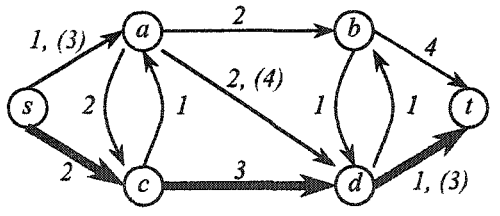
Step 4. In case D, we have already found two valid paths P_0 and P_2. Thus, the edge-weight-increment process is performed on both paths. By invoking Dijkstra's algorithm again, a third acceptable path P_3 is obtained as the second alternate path. It has one common edge with P_0 and no common edges with P_2. The real path length of P_3 $(s \rightarrow a \rightarrow b \rightarrow t)$ is *7*, and P_3 also satisfies the constraint on the number of edges.



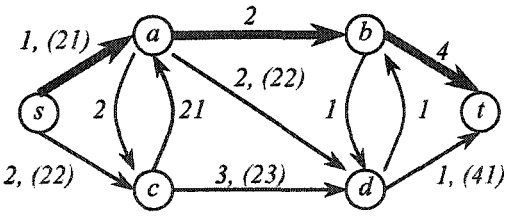
A). P_0: $s \rightarrow a \rightarrow d \rightarrow t$, path length = 4
P_0 is selected as the primary path



B). P_1: $s \rightarrow c \rightarrow a \rightarrow b \rightarrow t$, path length = 9
(Increase all edges in P_0 by $\Delta W = W_0 = 20$, and then recalculate path). But, P_1 violates $H_{max} = 4$ and it has to be discarded.



C). P_2: $s \rightarrow c \rightarrow d \rightarrow t$, can be accepted.
The real path length is 6. (Increase P_0 by $\Delta W = 2$, and then recalculate path)



D). P_3: $s \rightarrow a \rightarrow b \rightarrow t$, can be accepted.
The real path length is 7. (Increase P_0 and P_2 by $\Delta W = W_0 = 20$, and then recalculate paths)

P.S. For each edge, weight values in braces are fake (adjusted by the edge-weight-increment process).

Figure 5.7. Major steps to calculate the acceptable alternate paths by EWI.

Let us examine the time complexity of the EWI algorithm. The link-pruning process takes $T_{link\text{-}pruning}$ time. Dijkstra's algorithm runs in time $T_{Dijstra} = O(MlogN)$ in a graph with M edges and N nodes. For the inner loop in the pseudo code, we increase or decrease

ΔW in logarithmic steps between W_0 and 0; therefore, Dijkstra's algorithm will be called at most $(logW_0 + 1)$ times to calculate one acceptable alternate path. So, the time complexity of the EWI algorithm is:

$$T_{EWI} = T_{link\text{-}pruning} + 2 \times (logW_0 + 1) \times T_{Dijstra} = O(M) + 2 \times (logW_0 + 1) \times O(MlogN)$$
$$= O(MlogN \times logW_0) \qquad (5.4)$$

5.3.4 An Extension to the Bak's Algorithm (EBAK)

Bak's approach is to achieve load-balanced routing [Bak99a, Bak99b, Bak00]. By randomly choosing an intermediate node v from a selected set of network nodes, an alternate path from a source s to a destination t can be constructed by a concatenation of two segments – a shortest path from s to v and a shortest path from v to t. Node v is called a *pivot* node. We can reuse the idea of path concatenation to construct the alternate paths. However, we have to consider the path selection constraints described in Section 5.2, to calculate acceptable alternate paths.

Based on the main idea in Bak's approach of achieving load balance, we develop an algorithm, called EBAK *(Extension to Bak's algorithm)* to calculate acceptable alternate paths for a given node pair in an undirected graph. The pseudo code of the EBAK algorithm is illustrated in Figure 5.8:

1). Prune the links whose available bandwidth are less than the least desired bandwidth B_{min} ;
2). Construct the shortest path tree (SPT) rooted at the source node s, and the SPT rooted at the destination node t ;
3). Obtain the shortest path (the primary path $P_0(s, t)$) from s to t ;
4). Put node $x \in [V -$ all nodes on $P_0(s, t)]$ into a node set T. we call T the set of pivot nodes, which contains all nodes in the graph that are not on $P_0(s, t)$;
5). For each $x \in T$, construct a path $P'(s, t) = P_0(s, x) \,||\, P_0(x, t)$. Put $P'(s, t)$ into a path set H. If $P'(s, t)$ contains a cycle, or $P'(s, t)$ is a duplicate path in H, then we do not put $P'(s, t)$ into H ;
6). Sort the path set H in ascending order according to the path length ;
7). Choose two acceptable alternate paths from H by considering the predefined constraints for path lengths and the number of shared edges ;

Figure 5.8. Pseudo code for the EBAK alternate path finding algorithm.

In step 5 in Figure 5.8, there may exist multiple equal-length shortest paths from *s* to *x*, i.e., $P_0^i(s, x)$, $i = 1, 2, ...m$, and multiple equal-length shortest paths from *x* to *t*, i.e., $P_0^j(x, t)$, $j = 1, 2, ...n$. If this is the case, we construct a complete path $P'^r(s, t)$ for each possible combination of $P_0^i(s, x)$ and $P_0^j(x, t)$, i.e., $P'^r(s, t) = P_0^i(s, x) \parallel P_0^j(x, t)$. Here, $r = 1, 2, ... (i \times j)$. Then we put a $P'^r(s, t)$ into the path set *H* if $P'^r(s, t)$ does not contain a cycle, or $P'^r(s, t)$ is not a duplicate path in *H*.

MBAK differs from the Bak's approach as follows:

1). Bak's approach focuses on improving load balance without considerations of routing reliability and QoS guarantees. The shared edges among paths are also not considered. On the contrary, MBAK includes the link-pruning and path-checking processes to calculate acceptable alternate paths subject to the predefined path selection constraints.

2). Bak's approach uses a small node set chosen by certain randomization. However, in MBAK, all nodes that are not on the primary path $P_0(s, t)$ are used as pivot nodes, for calculating enough alternate paths.

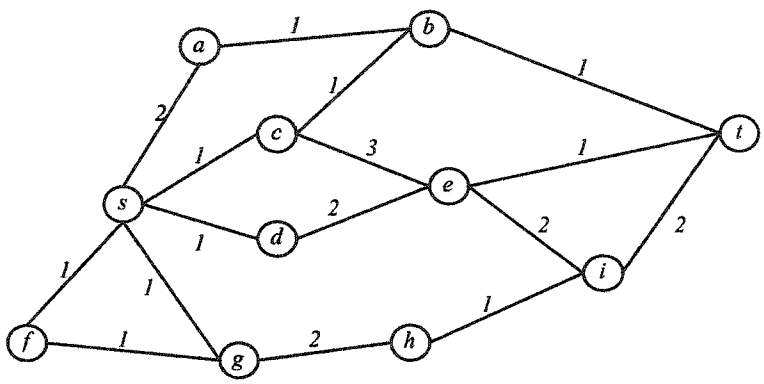
3). Every pivot node determines a candidate path from *s* to *t* by path concatenation. The acceptable alternate paths are selected from all appropriate candidate paths (containing no cycles) after they are sorted in ascending order.

4). In Bak's routing scheme, data packets are distributed on the paths in a random fashion for balancing traffic. However, EBAK is developed for alternate path routing. At any time instance, there is only one working path for a given node pair and the other paths are used as backups. Path switching occurs only when the working path is affected by the failures.

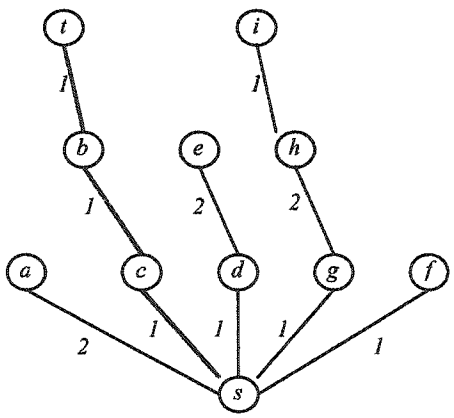
From the pseudo code shown in Figure 5.8, we can analyze the time complexity of EBAK. Steps 4 and 5 will take $T_{build\text{-}H} = O(N)$ time because there are at most $(N - 2)$ concatenated paths in set H. The path-sorting process in step 6 will take $T_{path\text{-}sorting} = O(NlogN)$ time, and the path-checking process in step 7 will take $T_{path\text{-}checking} = O(N)$ time. Therefore, the entire time complexity of the EBAK algorithm T_{EBAK} is:

$$T_{EBAK} = T_{link\text{-}pruning} + 2 \times T_{Dijkstra} + T_{build\text{-}H} + T_{path\text{-}sorting} + T_{path\text{-}checking}$$
$$= O(M) + 2 \times O(MlogN) + O(N) + O(NlogN) + O(N)$$
$$= O(MlogN) \qquad (5.5)$$

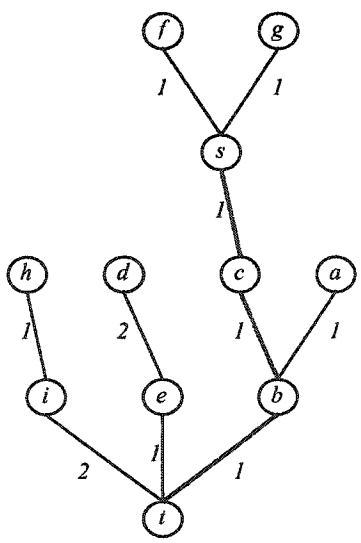
Figure 5.9 shows an example of two Shortest Path Trees (SPTs), one SPT is rooted at the source and the other is rooted at the destination. In Table 5.1, major steps for how to select the acceptable alternate paths by the EBAK algorithm are illustrated.



A). the sample graph



B). the SPT tree rooted at node *s*



C). the SPT tree rooted at node *t*

Figure 5.9. The shortest path trees rooted on the source and the destination.

Table 5.1. Construct the acceptable alternate paths by the EBAK algorithm.
(The primary path P_0 from s to t (i.e., the shortest path) is $s \rightarrow c \rightarrow b \rightarrow t$ with length 3.)

Pivot node	*Complete path*	*Path length*	*# of shared edges with P_0*	*Has cycle?*	*Acceptable?*
a	$s \rightarrow a \rightarrow b \rightarrow t$	*4*	*1*	*No*	*Yes*
d	$s \rightarrow d \rightarrow e \rightarrow t$	*4*	*0*	*No*	*Yes*
g	$s \rightarrow g \rightarrow s \rightarrow c \rightarrow b \rightarrow t$	*5*	*3*	*Yes*	*No*
f	$s \rightarrow f \rightarrow s \rightarrow c \rightarrow b \rightarrow t$	*5*	*3*	*Yes*	*No*
e	$s \rightarrow d \rightarrow e \rightarrow t$	*4*	*0*	*No*	*Duplicate*
h	$s \rightarrow g \rightarrow h \rightarrow i \rightarrow t$	*6*	*0*	*No*	*Yes*
i	$s \rightarrow g \rightarrow h \rightarrow i \rightarrow t$	*6*	*0*	*No*	*Duplicate*

5.4 Discussion

Four new alternate path finding algorithms are proposed in this chapter, namely, OptAlt, EMPS, EWI, EBAK. These algorithms have different optimizing objectives while calculating acceptable alternate paths. Comparisons among our algorithms and L&G [Lee01] are shown in Table 5.2.

Table 5.2. Comparisons of the alternate path finding algorithms.

Algorithm	*Objectives to be optimized (to the alternate paths)*	*Time complexity in the worst case*
OptAlt	*maximize reliability*	$O(MlogN + HN + {}_HC_k)$ ♣
EMPS	*minimize path lengths*	$O(MlogN + HN)$ ♣
EWI	*minimize shared edges*	$O(M\ logN \times logW_0)$ ♥
EBAK	– *	*O(MlogN)*
L&G	*1). minimize the number of edges on paths* *2). minimize path lengths*	$O(N^3)$

♣ *H* represents the candidate paths (calculated by the *K*-shortest paths algorithm) whose lengths satisfy the desired end-to-end delay constraint. *k* represents the number of paths to be calculated for a given node pair, i.e, $k = 2$, or *3*.

♥ W_0 is the sum of the weights of all edges in the graph.

* Better load balancing may be a side benefit of EBAK, which derives from the Bak's algorithm. However, we do not consider load balancing in this work.

Although the OptAlt algorithm provides the best reliability for alternate path routing, its overhead of computation is often significant.

The EWI algorithm tries to maximize diversity among the calculated paths, and then applies constraints of path lengths. In its logarithmic penalty-updating process, a maximum penalty will be chosen, by which the calculated alternate path may have a minimum number of shared edges with the previously calculated paths.

The EMPS algorithm selects acceptable alternate paths from the K shortest paths. Its path-checking process performed upon the ranked paths ensures that short alternate paths are selected. The edges shared among paths are constraints in this path-checking process. Compared with EWI, the acceptable alternate paths calculated by EMPS may have more shared edges.

Similar to EMPS, L&G is also based on a shortest path algorithm and selects the acceptable paths subject to the predefined constraints. Hence, paths calculated by L&G will have shorter path lengths but more shared edges among paths than EWI does. Another benefit of L&G is that there may be a smaller number of hops on the calculated paths when compared with EMPS. This feature comes from its BFS-like method, with the price of significant overhead of computation.

The alternate paths calculated by EBAK may have moderate path lengths and numbers of shared edges. There is no single objective optimized by this algorithm. EBAK stands somewhere between EWI and EMPS, and is the simplest and fastest algorithm in Table 5.2 to calculate the acceptable alternate paths.

By comparing the worst-case time complexities of $T_{L\&G}$, T_{EMPS}, T_{EWI} and T_{EBAK}, we see that our new heuristic algorithms can calculate acceptable alternate paths much faster than L&G.

The primary and alternate paths can be affected by network component failures. When all paths for a given node pair fail, a path recalculation becomes necessary. However, failures are not the unique reason to trigger a path recalculation. When accumulated changes of link lengths on a path achieve a high level (e.g. exceeding certain threshold), we can also treat the path as "failed". With this method, alternate paths can be

recalculated to accommodate link delay (including queuing delay) changes. This method will be used in the proposed R-MPLS in Section 6.3.

5.5 Simulations

We implemented the L&G, OptAlt, EMPS, EWI and EBAK algorithms. Simulations were performed to compare the performances of these algorithms. For each node pair in the graph, we tried to calculate three acceptable alternate paths use these algorithms.

In all simulation tests, we fixed the predefined constraints for selecting the alternate paths as follows:

- The maximum number of edges on each path $H_{max} = 10$
- The maximum number of double-shared edges $DSE_{max} = 4$
- The maximum number of triple-shared edges $TSE_{max} = 1$

We implemented a random graph topology generator for the simulation purposes. In Appendix B, we describe the popular models for generating random graphs, and illustrate our approach of random graph generator in detail. In a generated graph, each edge will be assigned a length, which is the distance between the two end nodes of the edge. The relations among graph parameters are shown in Equations B.5-B.7. Such graph parameters are the number of nodes N, the number of edges M, and average degree $avgD$ of all nodes in the graph.

In the simulations, we fixed graph size as $N = 50$ and changed the value of M.

5.5.1 Achieved Optimality

The new OptAlt algorithm calculates path groups with optimal reliability R_{opt}. Let R' represent the reliability calculated by a heuristic algorithm. Hence, optimality in reliability achieved by the heuristic algorithm can be estimated by R'/R_{opt}, which is a

measure of how close the heuristic solution is to the optimal solution. In the simulations, we assume that each edge has identical reliability $p = 99\%$. Table 5.3 shows the optimality comparisons.

Table 5.3. Optimality achieved by the heuristic algorithms

avgD	*R'/R_{opt} (%), the optimality of the heuristic algorithms (R_{opt} – reliabilities calculated by OptAlt)*			
	L&G	*EMPS*	*EWI*	*EBAK*
2.94	*98.72*	*98.63*	*99.07*	*98.24*
4.90	*98.78*	*98.62*	*99.05*	*98.28*
7.35	*98.81*	*98.58*	*99.01*	*98.31*
9.80	*98.75*	*98.64*	*99.03*	*98.27*
14.70	*98.76*	*98.61*	*99.06*	*98.33*

From the above table, we can observe that all of these heuristic algorithms can find solutions at least *98%* of the optimal. EWI shows the best performance among the four algorithms. The reason is that there are less shared edges in the alternate paths calculated by EWI. We also observe that L&G has slightly better reliability than EMPS, because the alternate paths calculated by L&G tend to have slightly fewer edges. In the above table, EBAK shows the least reliability of calculated paths, but the decrease in reliability when compared with the other algorithms is not great.

5.5.2 The Number of Calculated Paths

We attempt to select three paths for each source-destination pair. It is useful to know the number of paths calculated by the underlying alternate path finding algorithms under same network conditions, because a sufficient number of calculated paths improves the reliability. Table 5.4 shows the simulation results for the average number of paths calculated by the L&G, EWI, EMPS and EBAK algorithms. Here, we again observe that EMPS calculates almost the same numbers of alternate paths as L&G. Both find slightly more paths than EWI and EBAK.

Table 5.4. Average number of paths calculated by the heuristic algorithms.

avgD	*Calculated number of paths (three paths are expected)*			
	L&G	*EMPS*	*EWI*	*EBAK*
2.94	*2.82*	*2.83*	*2.71*	*2.53*
4.90	*2.96*	*2.97*	*2.94*	*2.65*
7.35	*2.99*	*2.99*	*2.96*	*2.78*
9.80	*3*	*3*	*2.99*	*2.84*
14.70	*3*	*3*	*3*	*2.91*

From the table, both EMPS and EWI algorithms can still find nearly three paths for every pair of nodes in most cases. When the input graph is sparse, the algorithms determine a small set of alternate paths. For example: for *avgD = 2.94,* EWI gives an average of *2.71* paths; while for EMPS, the value is *2.83.* Both numbers are close to *3.* Under the same network conditions, EBAK may find slightly fewer paths, on average *2.53* paths. This is because there are only limited numbers of candidate paths constructed by path concatenation at the pivot nodes, and one pivot node can only decide one candidate path.

5.5.3 The Numbers of Shared Edges

Shared edges have significant influences on reliability. Hence, it is interesting to compare the number of edges shared among paths calculated by the different algorithms. Tables 5.5 and 5.6 illustrate the simulation results for numbers of double- and triple-shared edges, respectively.

Table 5.5. Average numbers of double-shared edges among paths calculated by the heuristic algorithms.

avgD	The number of double-shared edges			
	L&G	EMPS	EWI	EBAK
2.94	2.61	2.64	0.51	1.62
4.90	2.31	2.36	0.44	1.51
7.35	2.14	2.18	0.37	1.38
9.80	1.73	1.80	0.32	1.21
14.70	1.62	1.64	0.28	1.07

Table 5.6. Average numbers of triple-shared edges among paths calculated by the heuristic algorithms.

avgD	The number of triple-shared edges			
	L&G	EMPS	EWI	EBAK
2.94	0.45	0.47	0.09	0.23
4.90	0.39	0.41	0.08	0.19
7.35	0.33	0.36	0.07	0.16
9.80	0.26	0.27	0.07	0.15
14.70	0.22	0.23	0.06	0.12

For all of the algorithms, fewer shared edges will exist as the graph becomes denser. The main reason may be that when the graph becomes denser, every node will have more neighbors, and there will be more choices for the alternate successors at each node. Therefore, for each given node pair, the algorithms tend to find acceptable paths with fewer shared edges.

From the above two tables, we also observe that EWI is calculating paths with the fewest shared edges when compared with the other algorithms. EBAK stands between EWI and EMPS. The alternate paths calculated by EMPS and L&G contain more shared edges when compared with EWI and EBAK. Moreover, EMPS results in almost the same number of shared edges as L&G. The simulation results meet our expectation. The reason is that the objective of EMPS and L&G is to minimize path length, whereas EWI seeks to minimize the number of shared edges.

5.5.4 Lengths of Alternate Paths

Simulations were performed to investigate the lengths of alternate paths calculated by the heuristic alternate path finding algorithms. When the graph contains $N = 50$ nodes and each node has about *3* edges *(avgD = 2.94)*, we observe that the average length of the first alternate path calculated by EWI is about *28%* longer than the first alternate path selected by EMPS. For the second alternate path, the value becomes about *37%*. This indicates that EWI is finding longer alternate paths than EMPS does. We also observe that, alternate paths calculated by EBAK have path lengths lying between the lengths of alternate paths calculated by EMPS and EWI. Comparing EBAK to EMPS, the first alternate path is about *15%* longer and the second alternate path is about *22%* longer.

In the simulations, we observe that the alternate paths calculated by EMPS and L&G have almost the same path lengths. However, the average number of edge on acceptable alternate paths calculated by these two algorithms is slightly different. L&G selects paths with about *12%* fewer edges than EMPS does.

5.6 Summary

In this chapter, a new alternate path finding algorithm (OptAlt) optimizing reliability subject to QoS guarantees was proposed. Moreover, three new heuristic alternate path finding algorithms (EMPS, EWI and EBAK) have also been developed to speed up calculations of acceptable alternate paths. The paths calculated by these heuristic algorithms may be disjoint or partially disjoint; however, they satisfied the constraints for selecting acceptable alternate paths, such as bandwidth demands, end-to-end delays and bounds on the number of shared edges.

The proposed heuristic algorithms have different features, which may satisfy diverse demands on end-to-end delay and the number of shared edges. EMPS tends to select shorter path lengths (i.e., end-to-end delays), while EWI tends to result in fewer shared

edges among paths to achieve higher reliability. EBAK stands somewhere between EWI and EMPS. EBAK may be preferable for its simplicity.

One existing alternate path finding algorithm, Lee and Gerla's algorithm (L&G), was also reviewed. This algorithm is similar to EMPS. However, the time complexity of L&G is much bigger than that of EMPS. Simulations were designed to compare L&G with our algorithms. The results showed that our algorithms could efficiently calculate the acceptable alternate paths.

We will apply our new alternate path finding algorithms to MPLS. In the next chapter, a reliable MPLS called R-MPLS will be proposed.

6. Reliable MPLS (R-MPLS)

MPLS [Ros01] is a currently emerging routing architecture. It describes a label substitution protocol that is used for routing data packets through predetermined paths and can be used to implement other desired routing features, such as QoS guarantees. MPLS can be viewed as a service that provides a path or circuit-switched construct in addition to the usual datagram service, for IP networks. MPLS is driven by industry players who realize that a path-switched service carrying IP datagrams is an essential supplement to datagram switching for many multimedia and other future applications.

To speed up the restoration of routes after failures occur, alternate path routing can be used in MPLS [Has00, Kin01]. Multiple explicit paths for a source-destination pair can be set up by MPLS signaling protocols [Awd01, Jam02]. The source nodes can preestablish alternate paths for improvement of routing reliability. Existing research effort of improving reliability for MPLS has been reviewed in Section 2.4.

In this chapter, we will propose a new extension to MPLS called *Reliable MPLS* (R-MPLS), derived by deploying alternate path routing and path restoration. Also, we will apply our new path finding algorithms to R-MPLS, because they can calculate acceptable alternate paths meeting the bandwidth and end-to-end delay constraints essential to multimedia applications.

6.1 MPLS Overview

The essence of MPLS is the generation of a short fixed-length *label* [Ros01] that acts as a shorthand representation of an IP packet's header. The label is used to identify a *Forwarding Equivalence Class* (FEC), by which a group of IP packets are forwarded over the same path. Thus labels correspond to paths. In MPLS, these labels are attached to IP packets by the first MPLS device they encounter as they enter the network. Such a device is called an ingress *Label Edge Router (LER).* The LER analyzes the contents of

the IP header and selects an appropriate label (indicating a desired path) with which to encapsulate the packet. Within the network, only the MPLS label is used to make forwarding decisions. Finally, as MPLS labeled packets leave the network, an egress LER removes the labels.

In MPLS, packets are forwarded along a *Label Switched Path (LSP),* fixed for each label, by a label-driven relay mechanism. Each node on the LSP is called a *Label Switching Router (LSR)* *. At each hop, the LSR strips off the existing label in a packet and applies a new label which tells the next-hop LSR how to forward the packet. MPLS thus provides a mechanism for sending packets independent of the IP routing tables in the routers, namely *label-driven routing.* Each packet is routed through a predefined path, which is determined before data transmission. The conventional IP routing mechanism is not used in MPLS. Figure 6.1 illustrates a LSP and the related forwarding tables (i.e., routing tables) on LSRs.

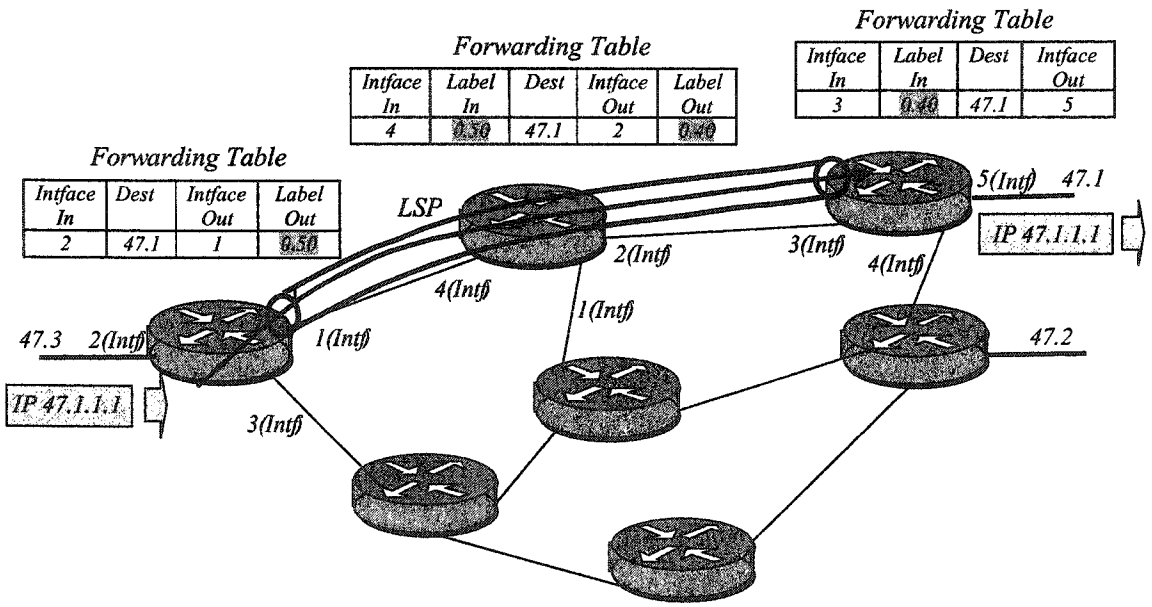


Figure 6.1. An established Label Switched Path (LSP).

* An ingress or egress LER is also a LSR.

6.2 Explicit Routing in MPLS

Although hop-by-hop routing dominates the current Internet, explicit or source routing, where the source node specifies a path from itself to a given destination before data flows, is becoming a necessity in latency sensitive applications such as audio and video streaming – one example of the QoS requirements of multimedia applications.

As a non-Explicit Routing mode in MPLS, hop-by-hop routing binds network resources such as link and switch capacity to data flows too late to permit absolute QoS guarantees. The link restoration technique (Section 3.3.1) can be used as an adaptive routing scheme to quickly bypass failures or congestion. However, such routing scheme is not suitable in QoS routing with reliability requirements. The reasons are:

1). Route calculations and label binding at each node are independent. A source cannot control the end-to-end routes to a desired destination. As a result, the end-to-end delay of the alternate routes may not be guaranteed.

2). In hop-by-hot routing, the connection reliability calculated at the source is not the real estimation of the reliability because routes chosen independently at intermediate nodes may not be consistent.

3). When path switches occur at intermediate nodes due to failures, end-to-end delay bounds may not be guaranteed and path cycles may be formed.

Precalculated routes as provided by source routing are necessary to permit sufficiently early binding of resources to flows and to allow us to achieve the desired QoS guarantees. In explicit routing, a path calculated at a source guides the flow of packets through all intermediate nodes to the destination. This feature of explicit routing provides a support to guaranteed services. Explicit routing is generally used to improve path controllability, eliminate cycles in the paths, support QoS demands [Awd01, Jam02], implement policy routing [Ste93a, Ste93b] and support traffic engineering [Awd99, Awd02].

Explicit routing is well supported in MPLS. In the label-driven routing mechanism in MPLS, labels are distributed between LSRs by a *Label Distribution Protocol (LDP).* An

Explicit Route (ER) can be established by installing a *Constraint-based Routed LSP (CR-LSP)* from an ingress LSR to an egress LSR. There exist two signaling protocols to establish an ER on all intermediate LSRs along the route: *CR-LDP* [And01, Jam02] and *RSVP-TE* [Awd01, Zha93], either is sufficient in setting up the path for a given source-destination pair.

The complete specification of the selected ER is provided by the ingress LSR in a setup message and the route specification is installed on all intermediate nodes by either one of the two MPLS signaling protocols. All the nodes along the ER follow the route specification and use this route as a basis for label request, label mapping, and data transfer. A label request message containing the entire path information is used to establish the end-to-end path. The direction of a label request is from the ingress LSR to the egress LSR along the route, while the direction of label mapping is from the egress LSR to the ingress LSR along the reverse route. Binding between the labels and FECs will be completed after this label mapping process. Resource reservation may or may not be performed at all intermediate LSRs. After the ER is established, explicit routing can be easily achieved by an operation of label lookup at each of the intermediate LSRs.

Label-driven routing in MPLS can be adaptive, because CR-LSP rerouting is supported by the signaling protocols in MPLS [*]. In general, LSP rerouting pertains to the provisioning of a new route for an old LSP on failure notification, topology change, or for optimization purposes.

6.3 The Proposed R-MPLS

We have developed a reliable alternate path approach for MPLS called *Reliable MPLS* (R-MPLS) [Pu02]. The main idea of R-MPLS is to apply alternate path routing and fast path restoration at source nodes to MPLS, for fast fault recovery in case of failures. The main features of R-MPLS are:

---

[*] Conventional MPLS based on OSPF single path routing will recalculate a new path and then establish it as a new ER in response to a failure. Hence, it will take a certain amount of time to get the new ER ready at the ingress LSR. Extensions to MPLS signaling protocols [Has00, Kin01] support preestablishment of multiple alternate paths for better reliability.

1). R-MPLS is an extension to MPLS for improving reliability. Our new alternate path finding algorithms are applied in R-MPLS. To consider reliability, we assume that all links have the same reliability. We are addressing homogeneous networks.

2). R-MPLS uses QOSPF (QoS extension to OSPF) [Apo99b] as the underlying routing protocol, which traces all changes (i.e., failures and link delay changes) on network topology and traffic load. Path calculations are based on QOSPF link-state database.

3). Two or three paths (i.e., the suitable path group) can be precalculated by our alternate path finding algorithms. Among these calculated paths (disjoint or partially disjoint), the shortest path is the primary path and the others are alternate paths.

4). All calculated paths, including the primary and alternate paths, will be preestablished as explicit routes (ERs) with resource reservation by a LDP before starting transmission. For a DWDM fiber network, one lambda (λ) can be used to represent one route in generalized MPLS [Ash03, Ber03]. Therefore, we can reserve one lambda for an alternate path to reduce cost of resource reservation.

5). When failures occur, the paths affected by the failures are marked, then replaced by available alternate paths without path recalculation. This is the path switching process in R-MPLS.

6). In case of link length (i.e., link delay including queuing delay) changes, the precalculated paths containing the links with modified lengths are identified. For each such path, the path length (i.e., end-to-end delay) will be adjusted at the source based on the latest link length information. In other words, R-MPLS will trace the link delay changes and keep the path lengths in the routing table updated.

7). When a path length becomes too long to satisfy the end-to-end delay demand, that path will be treated as a failed path. If such a path is a current working path, then path switching will be initiated at the source.

Due to time consumption of dissemination of routing information and end-to-end path establishment, scalability in MPLS and R-MPLS is poor when using explicit routing in a network. Hence, R-MPLS is applicable to a core network of limited size, e.g. an Enterprise Network. Also, we do not consider edge networks connecting end users.

In R-MPLS, a path group recalculation is unnecessary in case of a working ER failure. The ingress LSR will be notified, with failure information, and it will then switch to a preestablished backup ER. A path group recalculation is performed at the ingress LSR, only when all three paths to the same egress LSR fail.

Both path calculation and MPLS label calculation are done beforehand. Therefore, when a failure occurs, the ingress LSR will take less time to identify the affected paths and switch to backups (i.e., path switching in R-MPLS) than to recalculate new shortest paths (i.e., the conventional way).

Let M and N represent the number of edges and the number of nodes in a graph $G(V, E)$, respectively. There will be at most $3 \times (N - 1)$ entries in the routing table at the source in R-MPLS. Hence, path switching will take $O(N)$ time for a table lookup process. However, the conventional path recalculation when MPLS uses single path routing will take $O(MlogN)$ time by using Dijkstra's shortest path algorithm.

In MPLS, when a new ER is calculated in response to failures, the ingress LSR will send tear-down messages to delete the affected ER and then try to setup a new ER with resource reservation. In order to establish the new path, the path setup process has to be initiated again by the ingress LSR. Consequently, significant message exchange overheads will be introduced.

We propose a different scheme for R-MPLS. The MPLS signaling protocol can be extended so that it will not send the tear-down messages when a working ER fails. Instead, such an ER is simply marked as failed in the routing table without actually removing it. We still keep the entry of the failed ER in the routing table. After path switching, the ingress LSR will start to use an available preestablished alternate ER. As a result, routing convergence time can be significantly reduced.

This scheme is also helpful in failure recovery. If a previous failure is recovered, the ER affected by the failure can be quickly restored by changing the status of this path in the routing table. There are three policies to reuse the recovered path if its length (i.e., end-to-end delay) is shorter than the currently working path. In the first policy, a shorter path is always preferred to be a working path. Hence, path switching will be performed to use the recovered path. However, this policy may introduce frequent path switches (called *path flapping).* The second policy intends to reduce path flapping when a shorter affected path recovers. Path switching will only be performed when the working path fails. This will minimize path flapping, but cannot take advantages of using the shorter paths. In the third policy, a percentage threshold can be predefined to determine whether path switching is necessary. If the ratio of path lengths between the recovered path and the current working path exceeds such threshold, we will switch the working path to the recovered path with much shorter length.

Only when all paths for a given source-destination pair fail, the ingress LSR will have to remove all paths at once, and ask for a new path group recalculation. Hence, R-MPLS can take full advantage of alternate path routing and path restoration technique.

6.4 Simulations

We developed a simulation platform (i.e., a simulator) of alternate path routing whose intend is to study R-MPLS properties. The heuristic alternate path finding algorithms (i.e., L&G, EWI, EMPS and EBAK) were used in the simulations. We mainly focused on performance testing of the algorithms and the effectiveness of path switching routing scheme in R-MPLS.

6.4.1 Simulation Platform Structure

Figure 6.2 gives a flowchart for the main structures of the simulator. Alternate path finding algorithms build of routing tables based on the input graph topology (e.g., a sample graph or a random graph generated by our graph generator, described in

Appendix B). Moreover, the random failure generator can generate three types of random link-state updates (i.e., link failures, node failures and link length changes) as needed. A node failure can be treated as a node whose all associated edges fail. In the simulations, we assume that all nodes in the graph are notified with update events immediately and simultaneously.

Simulations have been performed on a sample optical network (a representative of a class of current enterprise networks), which is shown in Figure 6.3. This network contains $N = 31$ nodes and $M = 38$ edges. The average degree of all nodes is $avgD = 2.45$. In the simulations, we assume that each edge has the same capacity, link delay and identical reliability $p = 99\%$. Every edge is assigned an identical length *1*. For simplicity, we fix node *0* as the source node and all the other nodes are destinations. We preset the same constraints for all heuristic alternate path finding algorithms as follows:

- The maximum length of an acceptable path $L_{max} = 12$
- The maximum number of double-shared edges $DSE_{max} = 4$
- The maximum number of triple-shared edges $TSE_{max} = 1$

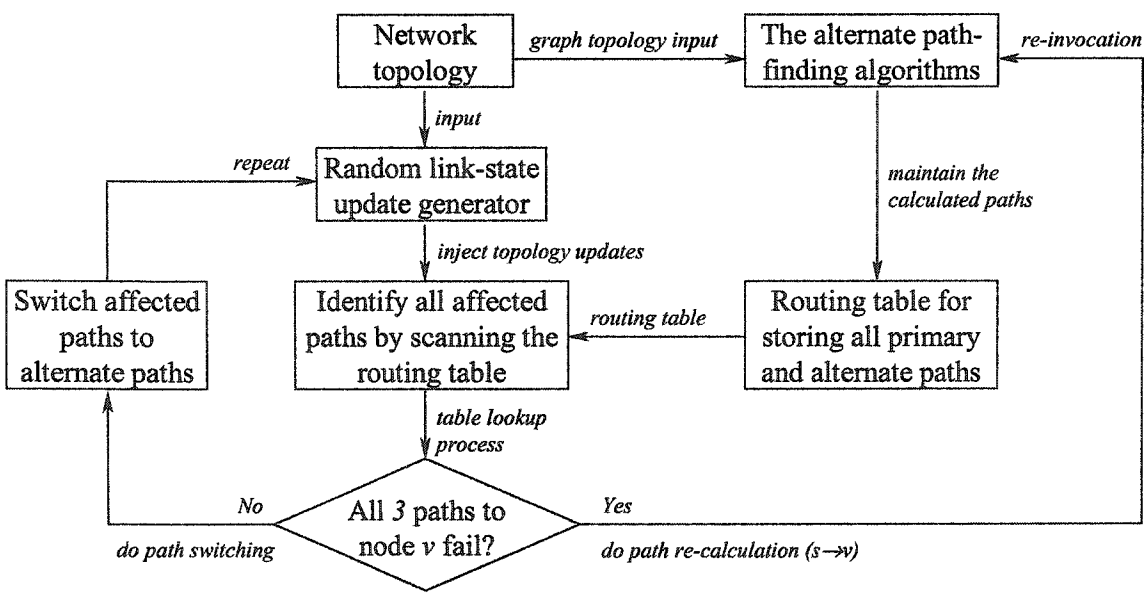


Figure 6.2. The main structure of the R-MPLS simulation platform.

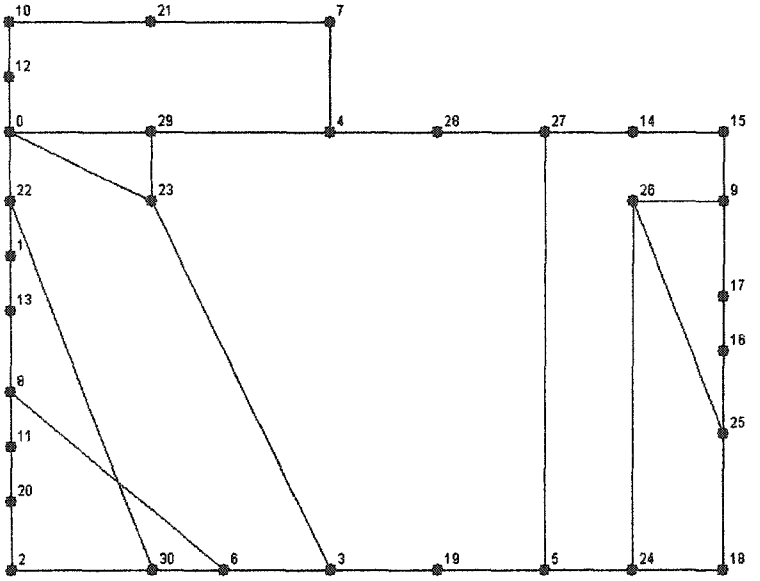


Figure 6.3. A sample 31-node network.

For each source-destination pair, we try to calculate three paths (one primary path and two alternate paths). Furthermore, to deal with the link length changes, a predefined threshold T_{chg} is used to decide whether path switching should be performed. If a current working path increases its length more than T_{chg}, we consider that this path has failed. In our simulations, we set $T_{chg} = 50\%$.

6.4.2 Features of the Calculated Paths

The features of the paths calculated by the heuristic alternate path finding algorithms are illustrated in Table 6.1. The column labeled *Reliability* shows the average connection reliability achieved by the algorithms. The column labeled *avgPNum* shows the average number of calculated paths. The columns labeled *avgDSE* and *avgTSE* show the average number of double- and triple-shared edges among the calculated paths, respectively.

In the simulations, we observe that EMPS and L&G calculate alternate paths with similar properties. This is expected.

Table 6.1. The paths calculated by the alternate path finding algorithms.

Algorithm	*Reliability R (%)*	*AvgPNum*	*avgDSE*	*AvgTSE*
L&G	*98.65*	*2.69*	*2.70*	*0.30*
EMPS	*98.62*	*2.67*	*2.73*	*0.31*
EWI	*98.78*	*2.58*	*1.86*	*0.15*
EBAK	*98.25*	*2.61*	*2.34*	*0.26*

When OptAlt, the algorithm optimizing reliability, is used in this sample network, the maximum reliability achieved is about *99.54%*. When using single shortest path (calculated by Dijkstra's shortest path algorithm) routing for each source-destination pair, the average reliability is about *96.03%*. Column *Reliability* in Table 6.1 shows that, like L&G, our algorithms can calculate the acceptable alternate paths with much higher reliability (Column *Reliability)* than the single shortest path algorithm does.

The above table also indicates that all the algorithms can calculate enough number of paths. EWI tends to calculate the acceptable alternate paths with the fewest shared edges among all of these algorithms. In the contrary, paths calculated by EMPS and L&G tend to contain the most shared edges. Moreover, in the simulations, we observe that the first alternate path calculated by EWI is about *31%* longer than that by EMPS. For the second alternate path, the value is about *40%*. Comparing EBAK to EMPS, the first alternate path is about *17%* longer and the second alternate path is about *26%* longer. Hence, EBAK stands between EWI and EMPS. The simulation results support our analysis about features of the algorithms in Section 5.4.

6.4.3 Path Switching

To investigate the dynamic properties of R-MPLS, we simulated the dynamic routing process in the presence of failures. Three types of link-state update events were simulated: node failure/recovery, edge failure/recovery and edge length change. We randomly injected the link-state update events of desired types. Each injected event is detected with probability *1* by the neighboring nodes, and all other nodes are notified

immediately. Then, the source node starts the scanning process to determine all paths affected by this update event. By counting the number of affected paths to each destination, we determine whether path switching or path recalculation should be performed at the source to bypass the failure. In R-MPLS, path switches will suffice to bypass failures in most cases.

For simplicity and efficiency, node failure events were used in the simulations of path switching (for R-MPLS) and path recalculation (for MPLS). As we discussed before, path switching in R-MPLS is much faster than path recalculation in conventional MPLS using single path routing. The simulation results also indicate this – it will take about

$$t_{switch} = 0.14\ ms$$

to identify the affected paths at node *0* by scanning the routing table, and then pick entries of the available precalculated alternate paths in R-MPLS; and it will take about

$$t_{recal} = 5.2\ ms$$

to recalculate new shortest paths by Dijkstra's algorithm in MPLS. Here, *"ms"* means millisecond. The above two values are the time costs of software execution [♣]. We also observe that the values of t_{switch} for our new algorithms are almost the same. The reason is that t_{switch} only depends on the number of path entries in the routing table.

Let t_l represent time interval to receive the failure notice. In OSPF, when a node detects a link failure, it will compose a LSA and notify such failure to every node in the network [♥]. In a broadcast network, dissemination of the routing information takes about *1.3ms, 14.7ms* and *49.5ms* in average when the numbers of nodes in the networks are *10, 50* and *100,* respectively [Sha01]. For our *31*-node network, it may take about $t_l \approx 9ms$ by interpolation.

[♣] Our simulations were implemented in C (Microsoft Visual C++® 6.0), and ran on a desktop computer (650MHZ Pentium III CPU, 128MB main memory, and Windows 2000 operating system).

[♥] OSPF uses an adjustable parameter *"MinLSInterval"* (often *5* seconds) to reduce routing messages and path recomputations. It is the minimum time between two distinct originations of LSA for a node. For two continuous failures around a same node with very short interval, origination of LSA to the second failure may be delayed. But, this scenario is rare. We can decrease *MinLSInterval* for quicker response.

In R-MPLS, there is no time cost for establishing a chosen alternate path. Hence, the entire convergence time to respond to a failure is

$$t_{R\text{-}MPLS} = t_1 + t_{switch}$$

When single path routing is used in MPLS, the process of reestablishing a new calculated path is unavoidable. Let t_2 represent the *LSP Establishment Latency* (i.e., the time cost to establish a new path). The entire convergence time to respond a failure in conventional MPLS is

$$t_{MPLS} = t_1 + t_{recal} + t_2$$

To establish an explicit path and allocate required resources at every intermediate node, two end-to-end steps are needed by an MPLS signaling protocol: propagate a label request from the source to destination, and perform label mapping reverse from the destination to source. As an estimation of time cost to establish the new path, t_2 would typically have a big contribution [♣] to the total convergence time t_{MPLS}. Moreover, t_2 increases when the size of network grows. As a conclusion, $t_{R\text{-}MPLS} << t_{MPLS}$ is true based on these data, i.e., R-MPLS would have much smaller convergence time than conventional MPLS using single path routing.

6.4.4 Path Recalculation

R-MPLS is proposed for improving the reliability of MPLS by applying the alternate path scheme. It is meaningful to compare reliability for R-MPLS and MPLS under identical failure conditions in the network. We performed corresponding simulations to obtain the ratio of numbers of required path recalculation R_{recal} when the source node

---

[♣] Example 1, in the New Jersey LATA test network (*10* nodes and *23* links), the time cost demanded to establish a new path (i.e., call setup) could be up to hundreds of milliseconds [Cho93].

Example 2, Ali et.al. designed a MPLS performance test suite [Ali02] by using a IW95000 network analyzer and a UNIX workstation. There was only one link in this two-node network. According to their simulation results, it took up to *12ms* in average to establish a LSP between the two nodes. Obviously, such time will increase quickly when the LSP has more links. Hence, path establishment is costly.

uses the alternate path finding algorithms (i.e., in R-MPLS) and the Dijkstra algorithm (i.e., in conventional MPLS). Here, a path recalculation in R-MPLS means a path group (the primary path and the acceptable alternatives) recalculation. A smaller value of R_{recal} denotes that more failures can be tolerated before next path recalculation is required. The simulations are based on the same link-state database. In R-MPLS, the database is used for the initial alternate path calculations as well as to respond to failures.

Table 6.2 illustrates the results of the ratios R_{recal} when injecting the same sequence of link-state update events to R-MPLS and MPLS during the simulations. From this table, we observe that the number of path recalculations in R-MPLS is much smaller than in conventional MPLS. Such results match the analyses in Section 6.3 and show that we can significantly improve the reliability of R-MPLS by using alternate paths. This table also suggests that, when EWI is used, R-MPLS will need to do about half as many path recalculations as the conventional single path routing in MPLS for the node failures or the link failures, and about *83%* fewer for the weight changes (Row 3). Similar conclusions can be made for L&G, as well as the EMPS and EBAK algorithms.

Table 6.2. The ratios of numbers of path recalculations between R-MPLS and MPLS.

Algorithm	*The ratio of path recalculation R_{recal} (%)*		
	Node failure & recovery	*Edge failure & recovery*	*Edge length changes*
L&G	*62.24*	*51.43*	*19.23*
EMPS	*62.85*	*51.89*	*19.38*
EWI	*53.10*	*42.75*	*16.72*
EBAK	*59.23*	*49.17*	*18.11*

Furthermore, Table 6.2 shows that using EWI in R-MPLS will lead to slightly fewer path recalculations than the other algorithms. Paths calculated by EWI tend to contain the fewest shared edges, so that more failures may be tolerated. We also observe that node failures cause bigger R_{recal} values than either the edge failures or weight length changes do. The main reason is that a node failure can be treated as several edge failures and will

therefore affect more paths, while a certain number of edge length changes (i.e., enough to reach the threshold) will result in one or multiple path failures.

6.5 Summary

In this chapter, we proposed a Reliable MPLS (R-MPLS) based on alternate path routing. Our new alternate path finding algorithms are used in R-MPLS.

R-MPLS uses precalculated and preestablished alternate paths for improving reliability in case of failures of network components. Fast path switching can significantly reduce the response time to failures. Furthermore, the alternate path routing scheme gives R-MPLS a capacity to tolerate more failures before the next path recalculation becomes necessary. Only when all paths from s to t fail, a new path group recalculation is initiated. As a direct conclusion, under identical failure conditions in the network, there will be a significant reduction of path recalculations in R-MPLS, as compared to conventional MPLS using single path routing.

Simulations have been performed to analyze features of R-MPLS. The results indicated that, under identical failure conditions in a network, R-MPLS could achieve fast recovery from failures, and reduce path recalculations. The simulation results support our analysis to R-MPLS.

In the next chapter, we will propose a new admission control scheme based on R-MPLS and an existing admission control model. As an application of alternate path routing, the new scheme combines routing reliability with utility-optimal admission control. Furthermore, strategies to reduce idles of the preestablished alternate paths (i.e., a waste of resources) will be studied in the next chapter too.

7. Unification of Our Work on Reliability with Previous Work on Admission Controls for Optimal Network Performance with QoS Guarantees

Providing guaranteed *Quality of Service* (QoS) for multimedia applications has become an area of increasing interest [Cra98, Seg98, Apo99b]. To achieve such guarantees, the allocation and reservation of resources become necessary, and this must be done before the session requiring the resources is admitted and the guarantee of QoS is issued. Otherwise, we risk violating that guarantee.

CPU cycles and I/O bandwidth in clients and servers, link bandwidth in networks, buffers in each node, may all be considered as system resources. In a network oriented towards multimedia applications, some form of admission control must be used to admit or deny sessions, reserve resources and issue QoS guarantees: if we admit all applicants, we may over-consume resources and thus violate the guarantees. Admission control based on *Service Level Agreements* (SLAs) is an efficient technique for achieving these goals [Mul99, Iwa00, Liu01]. An SLA is a contractual agreement between customers and network providers, in which the customer requests service at a desired level of QoS and for a proposed price, and the provider agrees to provide the level of QoS at the agreed price, or does not agree.

In this chapter, we apply the alternate path routing mechanism and R-MPLS to this admission control problem, and develop a reliability-conscious SLA-based admission controller [Pu03a, Pu03b] for data networks. We assume that all links in the network have the same reliability. Our work is an extension to previous work on SLA-based admission control [Kha98, Wat01, Akb02a] and focuses on improving reliability while

achieving desired QoS levels, subject to all system resource constraints. Alternate path routing is used for this purpose because of its fast response when a working path fails.

The basic idea is this: we precalculate multiple alternate paths for each session and perform resource reservation [♣] on all these paths. In the event of node or link failure, a session can then be quickly switched to one of the alternate paths, maintaining the guaranteed QoS without having to run the full admission algorithm again.

The main drawback of our approach is *the idle resource problem* – resources allocated to the alternate paths may be wasted if these paths are seldom used when the network is quite reliable. To resolve this problem, we propose several schemes of resource reservation on the alternate paths in Section 7.3.6.

7.1 The SLA Optimization Problem

An SLA details the price offered and the level of QoS requested for the transmission of a multimedia flow through a network, and encapsulates specifications for this flow [♥]. In an SLA-based system, an *Admission Controller* considers incoming SLA requests and assigns available network resources in order to maximize network utility (often net revenue) while respecting all QoS guarantees. Khan & Manning [Kha97, Kha98, Kha02] solved the admission control problem, by showing that it could be mapped to a variant of the combinatorial knapsack problem, called the Multiple-Choice Multi-Dimension Knapsack Problem (MMKP). Based on this work, Watson developed an Admission Controller called *SLAOpt* [Wat01] to process and admit SLAs to a network.

However, Watson's admission controller assumes that the network topology is static. An initial snapshot of the network topology is used for all subsequent path calculations. Hence their work cannot deal with path failures. These are caused by node or link failures, which are assumed not to occur.

[♣] Efficient resource reservation strategies are worthy for future research. This should be investigated or developed by technologies such as AI (Artificial Intelligence), to adaptively use various algorithms under different conditions and preferences.

[♥] Several levels of QoS may be proposed, with an offered price for each, in which case the Admission Controller is free to select one of proposed levels at which to admit the flow.

Consideration of node or link failures motivates our proposal to extend SLAOpt, an extension which attempts to improve reliability in the face of failures, and which adapts to topology changes as are caused by node or link failures. Our extension is called *Reliable SLAOpt* (R-SLAOpt). Precalculated and preestablished alternate paths are used in R-SLAOpt, so that reliability will be significantly improved. Here, reliability refers to the probability that a particular SLA session will remain unaffected by phenomena such as network failures. For this purpose, a new alternate path finding algorithm was developed and new path switching mechanisms were introduced.

Each admitted SLA in R-SLAOpt is associated with both a *primary path* and multiple *alternate paths*. The primary path will be initially used as the *working path*, the path that carries the packets of a customer's flow. If the working path is disrupted by a failure of one of its constituent nodes or links, the SLAs associated with the flows conveyed by the failed path are called *affected SLAs*. The source node will reroute these SLAs to an available alternate path, so that the QoS guarantees for all the affected flows can be maintained. This process is referred to as *path switching*, and has been discussed in detail in Chapter 6.

Four sub-problems were identified for R-SLAOpt:

1). Calculating acceptable primary and alternate paths and grouping them as multiple *path sets* (also called *path groups*) for each SLA request.

2). For each admitted SLA, ensuring that all calculated paths meet the specified bandwidth and end-to-end delay constraints.

3). Performing admission control and QoS adaptation by running efficient algorithms to solve the MMKP. This includes selecting an optimal path group to serve an admitted SLA.

4). Switching affected flows from the failed working paths to suitable preestablished alternate paths when failures occur.

The main implementation task in R-SLAOpt was to integrate algorithms and software previously developed by Watson and Akbar et. al., for SLA admission control with the

software and algorithms developed for path switching under failure conditions. Put another way, our goal was to unify our work on R-MPLS with Khan et. al.'s work on utility-optimal admission control.

7.2 Related Work

In this section related work is presented, including previous work on multimedia adaptation by SLA admission control.

7.2.1 Existing Systems without Guarantees

The best-effort datagram service model of the current Internet does not require resource reservation prior to data transmission. When a packet arrives at a router, and buffer space and link capacity on the outgoing link are available, the packet is forwarded to the next router; otherwise, the incoming packet may be delayed, or even dropped. It is therefore, in general, impossible to predict or guarantee the bandwidth or latency experienced by a stream. Additionally, since each packet of a session is forwarded through the network independently, packets may arrive at the destination out of order. This service model has many advantages, but it is unworkable for real-time multimedia applications requiring absolute standards of performance, such as guaranteed end-to-end bandwidth, or strict delay constraints.

To support an absolute guarantee of QoS, it is necessary to have explicit routing (so that the links used are identified), resource reservations on those links (so that bandwidth is in fact available as promised along the selected route), and admission control (to prevent over-consumption of resources). These issues are discussed in the following sections.

7.2.2 Service Level Agreement (SLA)

A Service Level Agreement (SLA) is a contract between a network service provider and a customer. It is an agreement to provide a certain level of service, described by a bandwidth requirement with a delay bound from a source *s* to a destination *d,* for a certain period of time, at a given price [Akb02b]. Delay jitter bound may be also specified in the SLA. Jitter is related to buffer sizes allocated on each intermediate node from *s* to *t,* which could be considered as one resource constraint when performing admission control ♣. An SLA could specify additional conditions such as:

1). Time-dependent service (e.g., *2Gb/s* next Tuesday from *2* to *6* pm, for a videoconference)

2). Repetitive service (the above, but every Tuesday)

3). Multiple service/price options (e.g., *1Mb/s @ $10/hour* or *2 Mb/s @$15/hour)*

4). Service auction and bidding prices (If a customer cannot get the service that she wants at *$10/hour,* she may choose to raise her bid to *$12).*

For the original Utilty Model and SLAOpt, Kahn [Kha98] and Watson [Wat01] defined SLAs as specifying a source node, a destination node and one or more QoS levels. Each QoS level defines an offered utility (i.e., cost or revenue), a bandwidth requirement and an end-to-end delay constraint.

R-SLAOpt aims to guarantee reliability. Accordingly, the SLA specification was extended to contain reliability requirements. The expected minimum reliability (labeled as R_{min}) can be treated as an additional QoS parameter ♥ in the SLA, similar to the end-to-end delay requirement. Any set of path groups created to serve a particular SLA request must satisfy the constraint on reliability:

$$R \geq R_{min}$$

♣ Delay jitter is also related to the scheduling and forwarding strategies used at a node. To guarantee jitter, particular strategies are necessary at each intermediate node, as well as admission control at the source.

♥ When desired, the number of double- and triple-shared edges can be also used as the parameters indicating reliability, instead of using R_{min}.

It is reasonable that reliability is one of the important factors influencing pricing in an SLA specification: customers should expect to pay more for higher reliability.

Another newly introduced parameter is the number of paths (denoted as *k)* in each path group used to serve a particular SLA request. This is referred to as *group size* in the context of R-SLAOpt. When group size $k = 1$, R-SLAOpt is equivalent to SLAOpt. More alternate paths in each path group will permit higher reliability, but will require more resources and therefore should influence pricing.

7.2.3 The Utility Model

The Utility Model [Kha98] was proposed for the admission and adaptation of sessions in a multi-session multimedia service provider (i.e., a server). For each session described by an SLA, there is a set of QoS levels, e.g., Gold, Silver and Bronze, and the corresponding prices offered by the user. Each session i is either rejected or admitted at one of the offered QoS levels Q_i. The chosen QoS level determines both the *session utility* $u_i(Q_i)$, and the *session resource usage* $r(Q_i)$. QoS adaptation is performed by adjusting the sessions' QoS levels upward or downward. The objective of QoS adaptation and admission control is to maximize the system utility obtained, subject to honoring all QoS guarantees to the sessions admitted. This implies observing the resource constraints of the system - communication link bandwidths and latencies, CPU cycles, memory and I/O bandwidths. Figure 7.1 shows the relation between the system and session utility, and between resource mapping and related constraints.

The main contribution of the Utility Model was the solution of the admission control problem by mapping it to the MMKP [Kha98, Akb01b], where we wish to select at most one stone from each of several piles of stones, maximizing the total weight of the chosen stones while respecting multiple volume constraints on the knapsack. The mapping from the SLA admission control problem to the MMKP is:

1). A stone represents a single QoS level, using a particular path, for an SLA.

2). A pile of stones represents an SLA (with all possible QoS levels).

3). The weight of the knapsack represents total utility (revenue).

4). The dimensions of the knapsack volume represent the network resource constraints.

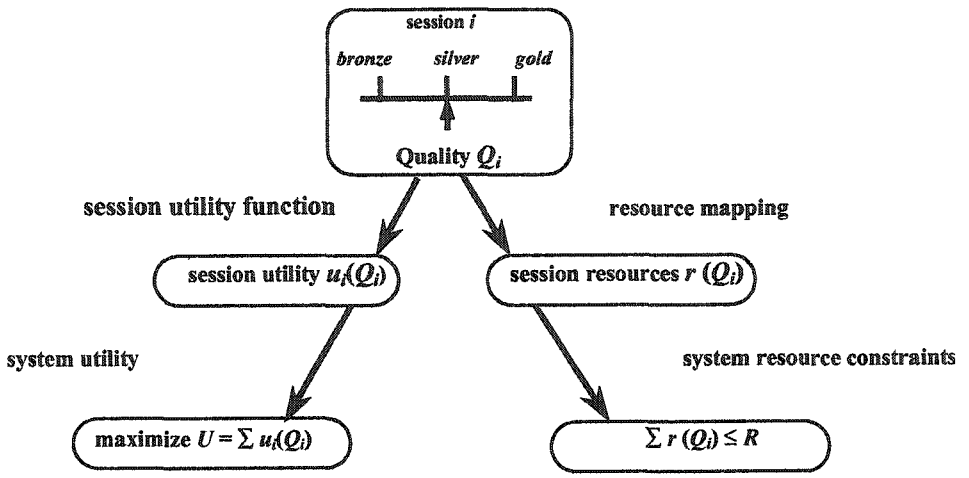


Figure 7.1 Relations between system and session utilities, and between resource mappings and constraints

The MMKP is NP-hard and, it turns out, impractical for real time decision making for cases of realistic size. However, heuristic algorithms [Kha02, Akb02a] have been developed to solve the MMKP. One such algorithm is I-HEU [Akb01b, Akb02b] for QoS adaptation. I-HEU is an incremental heuristic; it finds a solution of the MMKP using the results of the previously determined solution. I-HEU generally achieves about *97%* of the utility of the optimal solution. The first iteration of I-HEU takes a considerable amount of time. However, later iterations require much less time than the initial run – about *50%* to *90%* less, due to its incremental nature.

7.2.4 SLAOpt

SLAOpt is a *Service Level Agreement Optimizer* which was developed as a prototype Java implementation of the Utility Model applied to admission control of a data network [Wat01]. SLAOpt is an admission controller that seeks to maximize utility for a network,

subject to the constraints imposed by the QoS guarantees offered to each admitted session. Applying the Utility Model to a network introduced the problem of matching paths in the network to a QoS level described in an SLA. Note that SLAOpt makes no provisions for failure conditions. It simply assumes the admission control algorithm will be run again. This may lead to degrading or ending some of the previously admitted sessions.

SLAOpt uses the MPS *K*-shortest paths algorithm [Mar99] to find multiple suitable paths for each SLA. For every admitted SLA after QoS adaptation, one optimal path within such candidate paths will be chosen to serve the SLA session, so that the system utility is maximized. The complexity of the MPS algorithm is $T_{MPS} = O(MlogN + KN)$, where M is the number of edges and N is the number of nodes in a graph, and K is the number of paths found.

7.3 Reliable SLAOpt (R-SLAOpt)

Usually, optimizing performance and reliability are two different and competing goals for an admission controller. Hence it is more complex to simultaneously optimize them both. Optimization of performance may lead to a loss of reliability, and vice versa. Hence, R-SLAOpt is designed to optimally accommodate the SLAs requests with the desired QoS levels, while obeying a constraint on reliability.

7.3.1 Design Issues of R-SLAOpt

As mentioned earlier, alternate path routing is employed by R-SLAOpt to improve reliability. Mechanisms of path switching for SLA flows in case of failures are also proposed here. In this implementation, R-SLAOpt uses the I-HEU algorithm to solve the MMKP for QoS adaptation.

1). The MMKP is solved periodically; the interval between solutions is called an *epoch.* SLAs arriving during the epoch are batched for processing at the epoch's

end. Figure 7.2 shows an example of the epoch and the associated computational timelines in R-SLAOpt.

2). At the end of each epoch, the routing algorithm calculates paths from the latest description of the network topology, which reflects all changes due to failure, repair or provisioning of desired QoS guarantees during the epoch *. Then, the admission algorithm processes batches of SLAs for optimal adaptation.

3). Dynamic path switching in response to failures may occur at any time during an epoch. When a failure occurs during path finding or admission determination, that failure will be ignored until the computation finishes. Then, based on the newly calculated paths, the failure will be handled by a path switch.

4). In the event that the alternate paths are all unavailable, path recalculation is necessary. The failures and link length changes (e.g., caused by congestion) during an epoch will be taken into account at the end of the next epoch, when the latest topology is used to calculate new paths.

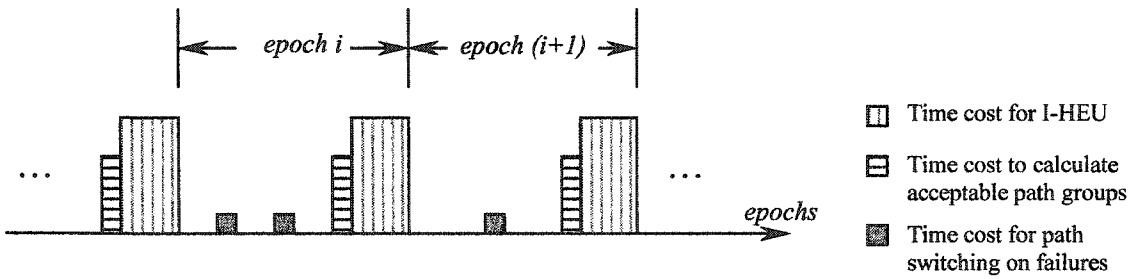


Figure 7.2. A timeline showing periodical calculations and path switching in R-SLAOpt.

The following assumptions apply to R-SLAOpt:

1). The underlying routing protocol will provide all nodes in the network (and the Admission Controller itself) with identical and updated copies of the network topology database.

* Path calculations are not necessary if a previously admitted SLA still has an available path from its source to destination.

2). All failures in the network will be detected, and each copy of the topology database will be updated immediately.

7.3.2 The Structure of R-SLAOpt

R-SLAOpt extends SLAOpt with structural changes. In R-SLAOpt, a source node *s* receives the QoS adaptation results (the calculated path groups) from the Admission Controller. Node *s* then assigns the flows with itself as source *s* (specified by the SLAs) to the selected working paths. When necessary, *s* will perform path switching. In other words, SLA rerouting in response to failures can be performed very quickly at *s*, without having to consult the Adapting Module. The structure of the proposed reliable Admission Controller based on R-SLAOpt is illustrated in Figure 7.3.

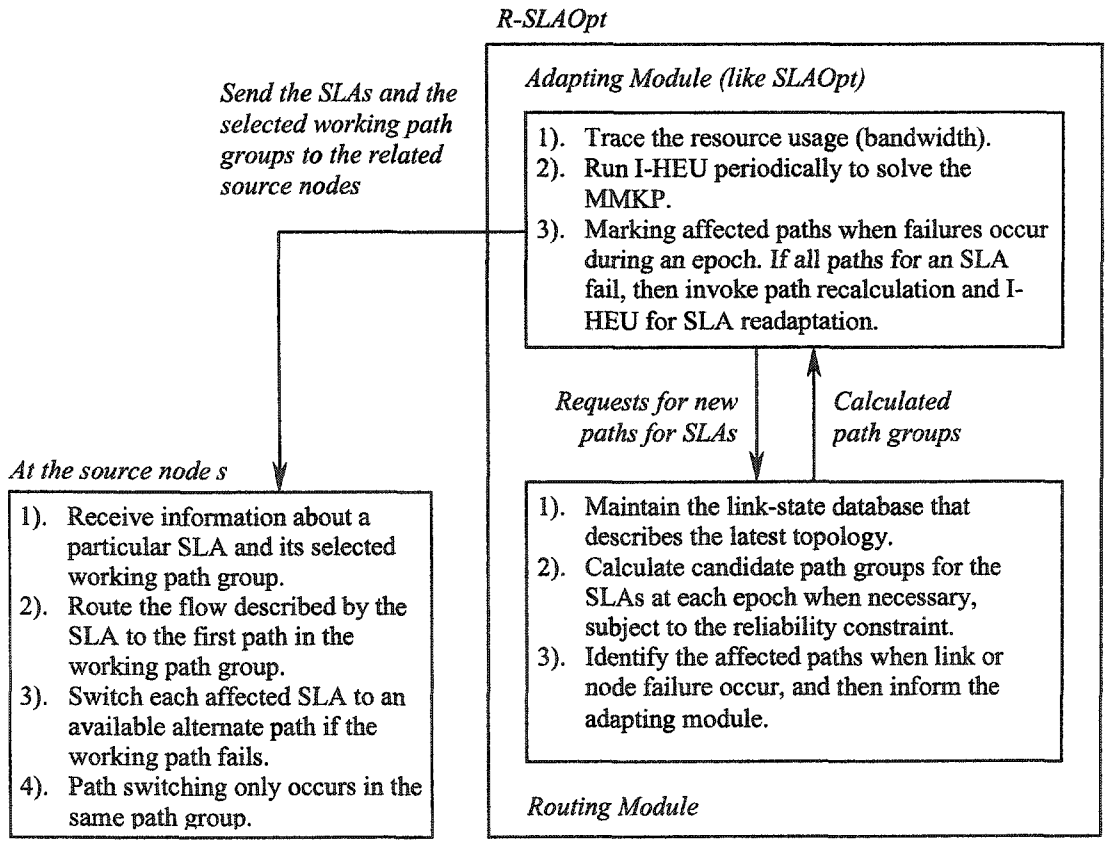


Figure 7.3. The architecture of the new reliable admission controller (R-SLAOpt).

In SLAOpt, QoS adaptation considers **multiple single** paths. In R-SLAOpt, QoS adaptation considers **multiple path groups**, each of them consisting of **multiple paths**.

At the end of each epoch, the Routing Module will calculate new paths if necessary, and the Adapting Module will process newly arriving SLAs and the affected SLAs based on the latest topology.

R-SLAOpt differs from SLAOpt in that each of the alternate paths in a group must meet the end-to-end delay constraint. Let D_{SLA} denote the end-to-end delay constraint for a given SLA. Let path length represent the total delay from the source to the destination. In SLAOpt, a path whose length $Len \leq D_{SLA}$ is eligible to participate in QoS adaptation. However, in R-SLAOpt, path groups are used to accommodate incoming SLAs. Let $Len(P_0)$, $Len(P_1)$ and $Len(P_2)$ be the length of path P_0, P_1 and P_2 in group $g(i)$, respectively. Let $Len(max)$ denote the maximum path length among $Len(P_0)$, $Len(P_1)$ and $Len(P_2)$. If and only if the delay criterion $Len(max) \leq D_{SLA}$ is satisfied can group $g(i)$ be eligible to participate in QoS adaptation. Due to this rule, when path switching is performed at a source node, the end-to-end delay constraint can still be met by any of the alternate paths. Also, of course, each of the alternate paths must have sufficient unallocated bandwidth on all of their links to accommodate the SLA in question.

7.3.3 GAPA – A New Algorithm to Calculate Acceptable Path Groups

In the MPS *K*-shortest paths algorithm [Mar99], path overlapping is not considered. In practice, shared links among different paths must be considered if reliability is to be adequately addressed. A failure on a shared link may affect multiple paths; hence, the number of shared links has a significant impact when attempting to meet a reliability constraint. A new algorithm is therefore needed to calculate multiple path groups associated with different reliability constraints subject to the QoS demands on bandwidth and end-to-end delay. During this research work, we could find no such existing algorithm.

A new alternate path finding algorithm called the *Grouped Alternate Path finding Algorithm* (GAPA) was therefore invented for R-SLAOpt. GAPA generates candidate groups of paths that are provided to the admission controller. Using the latest network topology, GAPA selects appropriate path groups from all possible candidates, subject to the constraints of a specified upper bound on path length and a lower bound on connection reliability for each path group. Three was chosen as the upper bound for the number of paths in a group, as larger groups were found to introduce complexities in the calculation of R, and may waste too much reserved bandwidth on the alternate paths if those paths are not used. The only way to **guarantee** sufficient bandwidth on an alternate path when we switch to it is to reserve all of the (potentially) required bandwidth on that path.

GAPA has three major steps. First, the MPS K-shortest paths algorithm is used to calculate the K acceptable paths. The latency of each path P_i, $i \in [1, K]$, must satisfy the end-to-end delay constraint specified in the SLA, i.e., the path length $Len(P_i)$ must meet the requirement for the maximum end-to-end delay for all specified QoS levels of the flow. Secondly, we compose all possible path groups from the calculated K candidate paths [*]. Finally, the acceptable path groups (after sorting) are chosen by selecting the ones whose connection reliability R satisfies the customers' demands. The time complexity of GAPA is

$$T_{GAPA} = T_{MPS} + O(r) + O(h\log h)$$

where,

$$r = {}_KC_3$$

is the number of all possible path groups, and h represents all acceptable groups whose R is greater than the minimum reliability R_{min} specified in the SLA. GAPA also works well when the specified group size $k = 2$. The algorithm complexity is almost the same as for $k = 3$, except that the number of candidate path group is ${}_KC_2$. The pseudo code for the GAPA algorithm is illustrated in Figure 7.4.

---

[*] When each group contains three distinct paths, there will be at most ${}_KC_3$ or $K(K-1)(K-2)/6$ such path groups. For example, let $K = 6$, we then have ${}_6C_3 = 20$ candidate path groups.

```
// The Routing Module (RM) receives an SLA from the Adapting Module (AM),
// RM extracts information of the s-d pair, end-to-end delay bound L_max,
// and the lowest reliability requirement R_min in the SLA

// Calculate K shortest paths that meet the end-to-end delay constraint
 Run the MPS K-shortest paths algorithm;
Take the first K paths, each of them satisfies path length ≤ L_max;

// Here, assume group size is 3. There are a total r = KC3 possible candidate path groups.
Compose all possible path groups, each of them contains 3 paths;

// Find h acceptable path groups from all r candidates
Q = null;                          // Q is a set containing all acceptable path groups
For each of the path group i, do {
    Calculate connection reliability R(i);
    If (R(i) ≥ R_min)
    Insert group i into Q;
}

Sort Q in descending based on reliability R;        // Q contains h path groups
Put the first x path groups from Q into a set Z;    // We may just use the first x groups
return Z;
```

Figure 7.4. Pseudo code for the GAPA algorithm.

7.3.4 Path Switching

Fast path switching at the source nodes is the most significant advantage of R-SLAOpt when compared to SLAOpt. We assume that the controller has been notified of the exact nature of the failure by the underlying network infrastructure and thus can easily determine a path's validity. There are two steps that the admission controller must execute when notification of failures is received – locating the affected paths and then switching these failed paths to available alternates when necessary. Locating the affected paths is essentially a linear operation in the controller, where every path in the working group for each active SLA is checked to determine if it is still valid. If a working path is invalidated due to a failure, a valid path can be selected quickly from the precalculated alternate paths. Note that path switching can only occur within the same path group.

With this path switching mechanism, R-SLAOpt can use a precalculated and preestablished path to quickly replace a failed one without recalculation and reestablishment. Our simulations also support this. Hence, R-SLAOpt could be a good candidate for supporting applications with real-time constraints. The pseudo code for performing path switching is shown in Figure 7.5.

In practice, however, the time required for path switching would most likely be limited by the underlying network infrastructure. The speed of the mechanisms used to signal the failure of network components would be a major factor limiting the time taken for path switching.

```
// The following process will be activated after a notification of failure is received
// Let a set Q contain all previously admitted SLAs
For each SLA in Q, do {
    // Identify all affected paths caused by the failures
    Obtain the current working path group associated with the SLA ;
    For each path Pi in the working group, do {
        Scan all nodes and links of Pi to determine whether Pi is affected ;
        If Pi is affected, then mark path Pi as "failed" ;
    }

    // Check whether the working path group is still available
    If all paths in the working group are marked as "failed", then {
        The SLA is affected ;
        Path recalculation and SLA readaptation for the SLA become necessary ;
    }

    // Do path switching if necessary (among currently available alternate paths)
    If the current working path P0 in the working group is marked as "failed", then {
        Find the first available alternate path Palt (i.e., the shortest one) ;
        Use Palt for all subsequent data packets belonged to the SLA ;        // switching
    }
}
```

Figure 7.5. Pseudo code for performing path switching.

7.3.5 Resource Reservation Scheme

SLAOpt was originally designed to accommodate SLAs from a set of multiple single paths. After an optimal path P_{opt} for an SLA was established, the required resources would be reserved on each link along path P_{opt}. However, in R-SLAOpt, QoS adaptation is based on multiple path groups, each of which contains multiple paths. Hence, resource reservation has to be modified after an optimal path group is selected.

When considering a given SLA, the Admission Controller will make bandwidth reservations on all links of all paths of the path group selected. The shortest path of the group, say P_0, can be selected as the primary path. If P_0 fails during an SLA session, the working path will be switched to P_1, the first alternate path, if P_1 is still available. For a same admitted SLA set, bandwidth allocated in R-SLAOpt will be double (i.e., $k = 2$) or triple (i.e., $k = 3$) when compared with SLAOpt. However, redundant resource reservations benefit fault recovery and improve reliability. Because sufficient bandwidth resources have already been reserved, the affected SLA can be appropriately accommodated on P_1. The same situation would occur with P_2, the second alternate path, in the event that P_1 also failed.

7.3.6 Pessimistic versus Optimistic Reservation of Resources

In R-SLAOpt, the admission controller reserves resources on all three paths in the selected working path group. This pessimistic scheme ensures that all QoS requirements will be guaranteed. However, this introduces significant waste of bandwidth resources on the alternate paths. P_0 (i.e., the shortest path in the group) is always selected as the working path. When failures are rare on P_0, path P_1 and P_2 are much less likely to be used. Hence, reserving resources on all the links on P_1 and P_2 may not be necessary, as these resources will be idle.

To solve this problem, the reliability criterion on P_1 and P_2 could be relaxed in the case where P_0 is very reliable. A kind of optimistic scheme might be used wherein a percentage of the total required resources is reserved on the links in paths P_0, P_1 and P_2.

For P_0, this parameter would be $p = 1$; for P_1 and P_2, $p < 1$. Obviously, this approach would introduce a risk of QoS downgrade when alternate path routing was applied. Finding appropriate p values for the alternate paths P_1 and P_2, and investigating the influences of QoS degradation caused by changes of these p values are both interesting problems requiring further study.

There is another possible solution to the above problem: resource reservation on alternate paths could only apply to SLA flows. Non-SLA flows, those which do not have QoS requirements (such as FTP or email traffic), could be transmitted on the bandwidth reserved for these paths while they were not in use for SLA flows. However, SLA flows would have priority access to the resources on the alternate paths. When an alternate path was activated due to failure of the primary path, the non-SLA flows would be displaced, so that the SLA flows could be accommodated.

A business-related question raised by R-SLAOpt is what to do with an SLA after path switching has occurred. The SLA now has fewer alternate paths remaining, and it could be argued that it is thus less reliable. One possible policy is to calculate new path groups during the next adaptation phase and switch the SLAs to these, thus restoring the reliability. An alternate policy is to reduce the utility cost to the SLA (i.e., charge the user less) to match the reduction in reliability. The charge for an SLA would thus be proportional to the number of paths reserved for the SLA. For example an SLA might be paying *$0.30* per minute for a given QoS level when three paths are reserved for its use. If the failure of a link makes one path invalid then the user would pay *$0.20* a minute. A counter argument to this policy is that the path failure is essentially transparent to the users, and since they see no changes in the QoS, they should see no change in billing. This is a policy issue and this work only addresses mechanisms to implement decisions.

7.4 Discussion

The lengths (i.e., end-to-end delays) of precalculated paths can be statically maintained in the proposed R-MPLS and R-SLAOpt. The complete paths and the associated path lengths are stored in the routing table at the source. When a link length

change occurs, the underlying routing protocol will report such update. Then, the source can modify the path lengths of the affected paths in its routing table according to the update. By this method, dynamic traffic and queuing delay can be considered in our reliable routing scheme.

Both R-MPLS and R-SLAOpt are based on alternate path routing and redundant resource reservations. To improve reliability is one of our main goals. R-MPLS and R-SLAOpt are practical in a network when the bandwidth capacity is not a limitation for network applications. For example, in current fiber networks, shortage of bandwidth capacity on links is rare.

R-SLAopt is designed for a core network with limited size (e.g., an enterprise network). However, it is common that network components are reliable (i.e., failures occur rarely), but incoming traffic is dynamic. Moreover, in wireless networks, channel characteristics (e.g., bandwidth) is variable during communications. Link or node congestion may be easily introduced within an epoch. To solve this, we have to extend R-SLAOpt. For example, historical information or experimental data about input traffic and channel characteristics can be used in admission control calculations, congestion control mechanisms are necessary to relieve congestion, and the epoch interval in R-SLAOpt may be shorten. Such extensions are considered as future work for R-SLAOpt.

In R-SLAOpt, system utility is optimized in admission control subject to QoS guarantees and reliability requirements. However, R-SLAOpt can be extended with diverse purposes for various preferences, such as optimize reliability or optimize resource usage. These extensions need further research.

7.5 Simulations

R-SLAOpt unifies performance optimization and reliability provision in an SLA admission controller. Several simulations were done to compare the performance of R-SLAOpt and SLAOpt. These included performance analysis of the routing algorithm and of the QoS adaptation algorithm used for solving the MMKP in R-SLAOpt. The engineering tradeoff between reliability and performance was also analyzed to measure

the quantitative effect of the reliability constraint on utility. Timing measurements in the simulations were taken to estimate the execution times of the algorithms. Moreover, artificial failures were injected into the network to investigate the reliability features of R-SLAOpt. Figure 7.6 illustrates the major software components of the R-SLAOpt simulator.

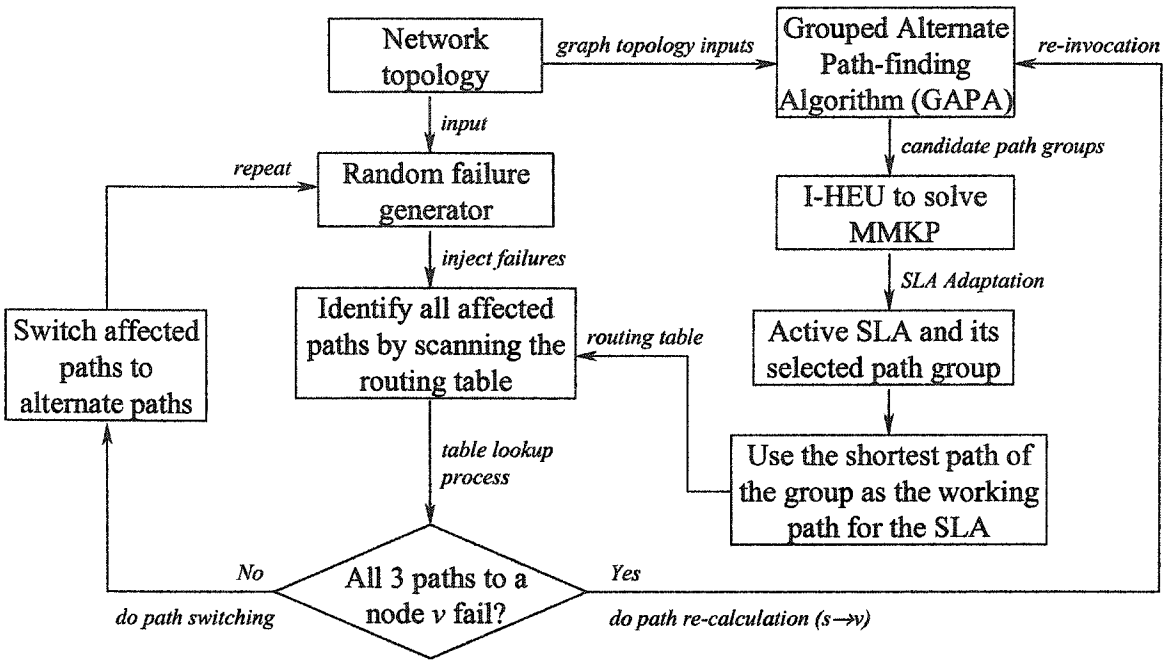


Figure 7.6. The software structure of the R-SLAOpt simulator.

All simulations were based on a sample optical network, a global network with components representing cities in North America, Australia and Africa. The network consists of $N = 25$ nodes and $M = 38$ links. In the simulations, all links were assumed identical and assigned with the same parameter values:

1). Link capacity: $c = 100{,}000$ *Mbps*

2). Link delay: $l = 10$ *millisecond (ms)* [♣]

3). Link reliability: $p = 99\%$

The simulations described here are aimed at investigating reliability issues that arise when failures are injected into an admission-controlled network. In Appendix C, more simulations which investigate features of the proposed GAPA path finding algorithm, and the performance of QoS adaptation in R-SLAOpt are described.

7.5.1 SLA Test Sets

Performance data were collected from generated SLA batches of different sizes. The characteristics of these batches were generally chosen to illustrate the performance of R-SLAOpt over a range of conditions. As an SLA-based system of the type R-SLAOpt would support has yet to be deployed, actual measurements of expected traffic are unavailable. The test sets used here were randomly generated in the same manner as those used by Akbar in [Akb02b].

The size of the batches was varied over a wide range of values *(150 – 1800* SLAs) to explore R-SLAOpt's performance. The bandwidth requirements for the QoS levels of an SLA were set to vary with the total size of the SLA batch so that there would always be mild contention for network bandwidth. In other words, these parameters were set so that there were never enough resources to accommodate all the SLAs seeking admission to the system. Moreover, the SLAs were generated so that higher QoS levels would be associated with bigger bandwidth requirements and shorter end-to-end delays.

The same SLA test sets were used in SLAOpt and R-SLAopt for all comparisons. Utility values were not adjusted to reflect the higher cost of reliable service to the users. This allowed analysis of the reliability and performance tradeoff.

[♣] Assume that the photon speed in fiber is about $0.6c$ ($c = 3\times10^8$ meter/second). Then, an optical link with *10ms* delay is about *1800* kilometers in length.

7.5.2 Reliability versus Utility Tradeoff

The computational time necessary to solve the MMKP by I-HEU will sharply increase when SLAs have a great number of possible QoS choices upon different paths. Hence, in the simulations, it is more efficient to use a small number of candidate paths (noted as *K)* for SLAOpt, or of candidate path groups (noted as *G)* for R-SLAOpt. To compare SLAOpt and R-SLAOpt, K and G should be the same. In previous work [Akb02a, Wat01], $K = 5$ was used in simulations.

In order to be comparable with previous works, we still use $K = 5$ for SLAOpt and $G = 5$ for R-SLAOpt in our simulations. The MPS K-shortest paths algorithm calculates *5* acceptable shortest paths for SLAOpt and R-SLAOpt. For simplicity, we assumed that two alternate paths were used as backups for each incoming SLA in R-SLAOpt – the group size was set as $k = 3$ for GAPA. Thus there would be a total of ${}_5C_3 = 10$ possible path groups composed from the *5* acceptable paths. The first $G = 5$ path groups with maximum connection reliability R were selected as the candidate groups for QoS adaptation by the admission algorithm.

SLAOpt does not consider reliability, allocating only a single path for each admitted SLA to achieve maximum utility. In R-SLAOpt, each admitted SLA is associated with multiple paths for high reliability. Obviously, system utility is decreased due to the higher consumption of resources. In order to compare SLAOpt and R-SLAOpt, we can apply the concept of the connection reliability to SLAOpt and calculate its reliability based on its single path data delivery.

Table 7.1 shows a comparison of system utility and reliability between SLAOpt and R-SLAOpt for the adaptation of one initial batch of SLAs. The simulation results were gathered from one admission operation at the first epoch. Note that the utility values offered by the SLAs are not adjusted to reflect higher or lower reliability. The smaller utilities obtained by R-SLAOpt occur because extra resources on each alternate path in the working group must be reserved, and thus fewer SLAs can be admitted. This is expected, and the percentage loss of utility can be compared to the corresponding improvement in reliability. It is a truism that reliability costs money; here we are actually able to quantify the utility cost of various degrees of network reliability.

Table 7.1. Comparisons of utility and reliability between SLAOpt and R-SLAOpt.

m (# of SLA)	*U: utility for SLAOpt*	*U': utility for R-SLAOpt*	*R: reliability for SLAOpt*	*R' : reliability for R-SLAOpt*
150	*407*	*315*	*0.9620*	*0.9944*
300	*736*	*613*	*0.9605*	*0.9938*
450	*1085*	*920*	*0.9589*	*0.9938*
600	*1463*	*1239*	*0.9600*	*0.9945*
750	*1880*	*1569*	*0.9595*	*0.9941*
900	*2197*	*1866*	*0.9602*	*0.9937*
1050	*2589*	*2161*	*0.9593*	*0.9945*
1200	*2975*	*2518*	*0.9593*	*0.9939*
1350	*3314*	*2791*	*0.9587*	*0.9941*
1500	*3690*	*3136*	*0.9597*	*0.9942*
1650	*4098*	*3441*	*0.9606*	*0.9944*
1800	*4465*	*3740*	*0.9586*	*0.9939*

7.5.3 Reliability when Network Topology Changes

In R-SLAOpt, the recalculation of new paths and readaptation of SLAs due to topology changes only becomes necessary when all paths in a working group fail. However, in SLAOpt an SLA has only one unique working path. If this path fails, recovery steps must be performed immediately. To analyze this improvement in reliability, two values of interest were investigated through simulation. First, the time required by the admission controller to handle failures was tested. This was the time taken by the controller upon notification of a failure to check if paths are valid and to signal path changes. Also of interest was the number of SLAs requiring recalculation (noted as *nAffected)* for SLAOpt and R-SLAOpt under the same failure conditions.

For simplicity and efficiency, random node failures were used in the simulations.

7.5.3.1 Path Switching

Figure 7.7 shows the time * required for performing path switching in case of node failures. It is the time, for each previously admitted SLA, that R-SLAOpt needs to identify all affected paths and selects available precalculated alternate paths when necessary. For comparisons, Table 10.3 in Appendix C illustrates time consumption of path calculations and QoS adaptation between SLAOpt and R-SLAOpt.

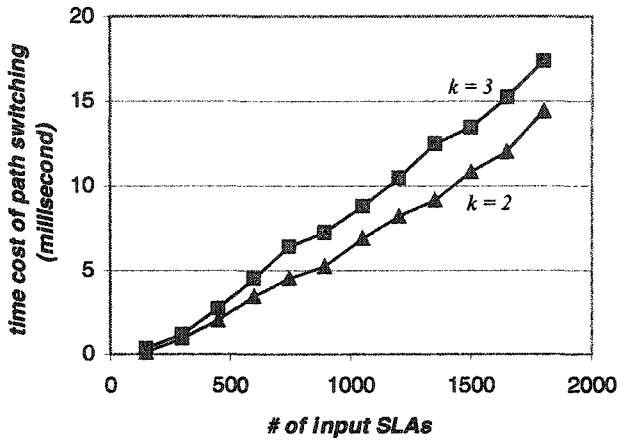


Figure 7.7. Time cost of path switching in R-SLAOpt.

In the simulations, the time for path switching is indeed very small as compared to those for path recalculation and SLA readaptation. For example, in SLAOpt, when there are *1500* SLAs in the system, path recalculations by the MPS algorithm will take about *50.2ms,* and SLA readaptation will take about *625ms* (from Table 10.3). Moreover, to establish the newly calculated paths SLAOpt will need extra time. Section 6.4.3 has shown that the process of path establishment is very costly. On the other hand, based on Figure 7.7, switching the failed paths to backups in R-SLAOpt will only take about *10.9ms* and *13.5ms* for $k = 2$ and $k = 3$, respectively. Hence, response to failures in SLAOpt is much slower than that in R-SLAOpt.

---

* The simulations of SLAOpt and R-SLAOpt were implemented in Java, and ran on a desktop computer (650MHZ Pentium III CPU, 128MB main memory, and Windows 2000 operating system) and Java™ 2 Platform, Standard Edition (J2SE™) 1.4.1_01.

As stated, the time taken by the controller in R-SLAOpt to select alternate paths is comparatively small. The simulation results indicate that the R-SLAOpt can more quickly obtain a new path in case of failures than SLAOpt. As a result, R-SLAOpt is worth further investigation as an appropriate candidate for supporting real-time applications requiring SLA type control.

7.5.3.2 Affected SLAs

A network component failure may affect transmission of certain SLAs if their working paths contain the failure. In SLAOpt, the affected SLAs have to recalculate new paths and then perform QoS adaptation. However, in R-SLAOpt, path switching is used in most cases to respond the failure because recalculation becomes necessary only when all paths for an SLA fail. Under the same failure conditions for SLAOpt and R-SLAOpt, the smaller number of SLAs requiring recalculation *(nAffected)* implies higher reliability.

To measure *nAffected* (the number of affected SLAs requiring path recalculation due to failures), a series of simulations were designed to incorporate random node failures. In the simulations, the group size k was varied between *1* and *3* for different test runs, and the system was subjected to the same sequence of failures. R-SLAOpt was equivalent to SLAOpt when $k = 1$. Random node failures were injected into the system, at intervals (denoted as *H)* of *5, 20* and *100* epochs – one node failure was injected into the network every H epochs.

In a real SLA admission controller, the system is dynamic – some existing SLAs are removed and new SLAs are added to the system as time elapses. It was meaningful to simulate such a dynamic process in our simulations of SLA admission control. Therefore, we designed the simulations as follows: several (approximately *10%* of the initial batch size) SLAs would be removed from the system and new ones (approximately *12.5%* of initial batch size) would be submitted in each epoch following the initial adaptation.

Table 7.2 illustrates the total number of affected SLAs *nAffected* as a percentage of the total number of admitted SLAs (denoted as *nAccepted)*, i.e., percentage P = *nAffected*

/ nAccepted (shown in Columns 2-4). A smaller *P* value indicates that there were fewer active SLAs affected by failures during the *H* epochs.

Table 7.2. The affected SLAs in case of node failures.

H (we inject one node failure every H epochs)	*P: the affected SLAs as a percentage (%) of the admitted SLAs*		
	k = 1	*k = 2*	*k = 3*
5	*9.41*	*5.35*	*4.41*
20	*2.34*	*1.28*	*1.04*
100	*0.51*	*0.28*	*0.22*

It can be seen that, when using alternate paths to respond to failures, the value of *P* drops quickly. For *H = 5,* there is an average *44%* drop in the value of *P* from *k = 1* to *k = 2,* and a further *17%* drop from *k = 2* to *k = 3*. Similar improvements occur when *H = 20* or *100.*

These results demonstrate that there are far fewer SLAs requiring recalculations due to network topology changes when using R-SLAOpt in a long term environment. In other words, R-SLAOpt can significantly improve reliability. This meets the design criterion and expectations for R-SLAOpt.

7.6 Summary

In this chapter, we have unified our work on reliability through alternate path routing with previous work by our colleagues on network admission control. We have presented a reliable SLA admission controller called R-SLAOpt. Precalculated alternate paths are used to improve the reliability of SLA flows.

A new alternate path algorithm called GAPA was developed to calculate multiple acceptable path groups for each SLA. SLA admission control is achieved by solving the MMKP admission problem with existing algorithms, considering path groups instead of single paths. Each path in a group satisfies the end-to-end delay guarantee, and sufficient

bandwidth is reserved on all the links on the paths in the selected group for the admitted SLAs. Consequently, when failures occur, the source node can quickly shift an affected SLA flow from a failed working path to an available alternate path, without violating any QoS guarantees for end-to-end delay or bandwidth.

Simulations to investigate the features of R-SLAOpt were presented. The simulation results showed that the GAPA algorithm can calculate suitable path groups to accommodate incoming SLAs, and that R-SLAOpt can improve reliability while doing QoS adaptation. Moreover, in the simulations of path switching, failures were injected and path switching was performed when the working path failed. The results showed that the time cost for path switching was very small when compared with the time cost of QoS adaptation. It was also observed that the number of SLAs affected by failures significantly decreased when there were more alternate paths in each group.

8. Conclusions

8.1 Major Contributions

Our work concentrated on improving routing reliability by using precalculated and preestablished alternate paths for sessions with requirements for guaranteed QoS. Routing reliability is quantified as connection reliability in this approach. The sets of calculated alternate paths may contain limited number of shared edges, but must meet the reliability requirements and QoS guarantees. Furthermore, we unified our research on routing reliability with previous work by colleagues on the SLA-based admission control problem. The MPLS explicit routing mode played a key role in this unification.

The major contributions of our work are:

1). We developed new strategies to improve routing reliability by adapting to packet networks the alternate path routing mechanism, which was first introduced to the switched voice network over *50* years ago. Path restoration, a fast and efficient failure recovery technique, and precalculated alternate paths are used as the basis of reliable routing.

2). We examined how to use non-disjoint alternate paths in alternate path routing for higher reliability. For such purposes, the effect on reliability of link sharing between non-disjoint paths for a given node pair was analyzed in detail.

3). The preferable number of paths for alternate path routing was evaluated by applying the *20-80* rule. This number can thus be reasonably chosen as three, based on the tradeoff between achieved reliability and system resource consumption.

4). An analysis method for routing reliability was proposed, to quantify routing reliability by using connection reliability for a given node pair. We calculated such reliability (i.e., the overall success probability of the multiple disjoint or non-disjoint paths), and investigated how double- and triple-shared links affect the

dynamic routing process, by applying the 3-D Venn Diagram model [Fre80]. General mathematical formulae to calculate connection reliability were derived.

5). An alternate path finding algorithm (i.e., OptAlt) that optimizes reliability subject to the path-selection constraints of QoS guarantees was developed. The calculated path group for a given node pair will have maximum possible reliability.

6). For simplicity and efficiency, three new heuristic algorithms (i.e., EMPS, EWI and EBAK) were also developed to calculate acceptable alternate paths that meet the requirements on path lengths and the number of shared links, as well as the bandwidth demands. Each alternate path finding algorithm has distinct features and may satisfy particular requirements.

7). Simulations were performed to validate the alternate path finding algorithms. For this purpose, we implemented a random graph topology generator based on Waxman's model [Wax88]. The simulation results suggested that our algorithms can calculate a sufficient number of alternate paths, which can provide the QoS guarantees and meet the reliability requirements.

8). We applied our alternate path finding algorithms and analysis of routing reliability to MPLS. Reliable MPLS (R-MPLS), an extension to MPLS, was proposed to improve system reliability based on MPLS explicit routing mechanisms.

9). We developed a simulator for alternate path routing in R-MPLS and injected artificial failures. The simulation results showed that, compared with the fault recovery process in MPLS using single path routing, path switching in R-MPLS is a very fast process and reliability can be significantly improved.

10). Furthermore, we reviewed the problem of SLA-based admission control using the MMKP to achieve maximum revenue. An existing approach is SLAOpt. To improve reliability, node or link failures needed to be considered in admission control. Based on SLAOpt, we proposed a reliable SLA-based admission control scheme (i.e., R-SLAOpt), by applying SLA-based QoS adaptation together with R-MPLS, thus unifying our work and this previous work on utility-optimal admission control. Related resource reservation methods were also carefully examined.

11). A new alternate path finding algorithm called GAPA was developed for R-SLAOpt to calculate multiple acceptable paths groups, each of which meets the constraints on reliability and QoS requirements. Each path group may contain two or three paths. An optimal group was selected as the working path group for an admitted SLA to achieve maximum revenue.

12). We developed a Java simulation for R-SLAOpt, the reliable SLA-based admission controller. The simulator was used to simulate network component failures. The simulation results demonstrated that R-SLAOpt has much higher reliability than SLAOpt does under the same failure conditions.

8.2 Future Work

Some future work on routing reliability and its applications is suggested as follows:

1). In our analysis of reliable routing, only shared links were considered. Future work can study the problem of node sharing among paths in alternate path routing.

2). We only deployed the technique of path restoration to speed the convergence time in case of failures. Research on fast failure recovery techniques based on other restoration techniques, such as link restoration and partial path restoration, subject to QoS constraints, may be an interesting topic to study.

3). Three new heuristic algorithms were developed to calculate acceptable alternate paths. However, in these algorithms, the shortest path is always used as the primary path, and alternate path calculations are based on this path. In some cases, a path set which does not include a shortest path may be a better choice, e.g., there may be fewer shared edges among paths and shorter alternate path lengths. A search for alternate path finding algorithms that address this problem is interesting.

4). So far, we have focused on achieving the reliability requirements and QoS guarantees by explicit routing. Currently, hop-by-hop routing still dominates in IP networks. Hence, improving reliability in hop-by-hop routing subject to the QoS guarantees may be a useful research topic.

5). R-SLAOpt adapts the QoS levels of incoming SLAs among multiple selected path groups by solving the MMKP. However, other types of bandwidth allocation solutions, instead of mapping admission control to the MMKP, could be a research direction. Objective functions may also be diverse – including but not limited to maximizing reliability or minimizing resource usage.

6). One major drawback of alternate path routing is the idle resource problem. The alternate paths may often idle when the network is reliable, so that the prereserved resources on these paths are wasted. We have proposed several methods to handle such situations. However, more research for developing efficient and practicable strategies to balance the tradeoff of reliability and resource consumption is worthwhile.

7). Multicasting is a capability to deliver data from a source to a group of receivers. It is a kind of emerging multimedia-based applications. Improving reliability in multicasting (estimated by *Rel(s, K, G))* with QoS guarantees is still an open topic. How to introduce multicasting capability to R-MPLS and R-SLAOpt is a meaningful work. Based on the idea of alternate path routing, multiple multicast trees from the source to the group of receivers are essential for high reliability. Then, when failures or congestion occur, alternate paths can be quickly activated and used. More research work on multicasting extension to R-MPLS and R-SLAOpt is necessary.

8). Very similar to the multicasting extension, our work can be also extended to n-to-1 (multipoint-to-point) network applications. However, congestion may be easily introduced due to traffic merging. Therefore, new resource reservation schemes are necessary and certain congestion control is required.

9). Our research on routing reliability using non-disjoint alternate paths now focuses on existed networks. It is reasonable to extend our work to network design before a network is built. Common links and nodes shared by the alternate paths should be considered in the network design level so that reliability can be optimized.

9. References

[Akb01a]. M. M. Akbar, E. G. Manning, G. C. Shoja, *Admission Control and QoS adaptation in Distributed Multimedia Server System,* ITCom 2001, Denver, USA, August, 2001.

[Akb01b]. M. M. Akbar, E. G. Manning, G. C. Shoja, S. Khan, *Heuristic Solutions for the Multiple-Choice Multi-Dimension Knapsack Problem,* International Conference on Computational Science, San Francisco, USA, May 2001.

[Akb02a]. M. M. Akbar, *The Distributed Utility Model Applied to Optimal Admission Control & QoS Adaptation in Multimedia Systems & Enterprise Networks,* Ph.D. Dissertation, Department of Computer Science, University of Victoria, 2002.

[Akb02b]. M. M. Akbar, E. G. Manning, R. K. Watson, G. C. Shoja, S. Khan, K. F. Li, *Optimal Admission Controllers for Service Level Agreements in Enterprise Networks,* 6th SCI Conference, Orlando, FL, July 2002.

[Ali02]. K.A. Ali, H.T. Mouftah, J. Tur and D. Fraser, NetTest, *MPLS Signaling Performance Testing,* The 9th IEEE Workshop on Computer-Aided Modeling, Analysis and Design of Communication Links and Networks (CAMAD), New York city, New York, May 2002.

[And01]. L. Andersson, P. Doolan, N. Feldman, A. Fredette and B. Thomas, RFC3036, *LDP Specification,* January 2001.

[Apo98]. G. Apostolopoulos, R. Guérin, S. Kamat and S. K. Tripathi, *Quality of service based routing: a performance perspective,* Proceedings of the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication, pp. 17-28, Vancouver, Canada, August, 1998.

[Apo99a]. G. Apostolopoulos, S. Kamat, and R. Guerin, *Implementation and Performance Measurements of QoS Routing Extensions to OSPF,* Proceedings of the Conference on Computer Communications, IEEE Infocom, New York, March 1999.

[Apo99b]. G. Apostolopoulos, RFC 2676, *QoS Routing Mechanisms and OSPF Extensions,* August 1999.

[Ash03]. P. Ashwood Smith and L. Berger, RFC3472, *Generalized Multi-Protocol Label Switching (GMPLS) Signaling Constraint-based Routed Label Distribution Protocol (CR-LDP) Extensions,* January, 2003

[ATM02]. The ATM Forum Technical Committee, *Private Network-Network Interface Specification v.1.1*, http://www.atmforum.com, April 2002.

[Awd01]. D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow, RFC3209, *RSVP-TE: Extensions to RSVP for LSP Tunnels,* December 2001.

[Awd02]. D. Awduche, A. Chiu, A. Elwalid, I. Widjaja and X. Xiao, RFC 3272, *Overview and Principles of Internet Traffic Engineering,* May 2002.

[Awd99]. D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell and J. McManus, RFC 2702, *Requirements for Traffic Engineering Over MPLS,* September 1999.

[Bah92]. S. Bahk and M. E. Zarki, *Dynamic Multi-path Routing and How it Compares with other Dynamic Routing Algorithms for High Speed Wide Area Networks,* Proceedings of the ACM Symposium on Communications Architectures & Protocols, Baltimore, USA, pp. 53-64, August 1992.

[Bak00]. S. Bak, *Load-balanced Routing,* PhD dissertation, Department of Computer Science, University of Houston, 2000.

[Bak99a]. S. Bak, J. A. Cobb and E. L. Leiss, *Load-Balanced Routing via Randomization,* XXV Conferencia Latinoamericana de Informatica (CLEI'99), Asuncion, Paraguay, September 1999.

[Bak99b]. S. Bak, J. A. Cobb and E. L. Leiss, *Randomized Distance-Vector Routing Protocol,* XXV Conferencia Latinoamericana de Informatica (CLEI'99), Asuncion, Paraguay, September 1999.

[Bal92]. M. O. Ball, C. J. Colbourn and J. S. Provan, *Network Reliability,* Technical Report, Department of Computer Science, University of North Carolina, June 1992.

[Ban01a]. G. Banerjee and De. Sidhu, *Label Switched Path Restoration under two Random Failures,* Proceedings of IEEE GLOBECOM 2001, San Antonio, Texas, USA, July 2001.

[Ban01b]. G. Banerjee and D. Sidhu, *Multi-Constrained Path Computation for Traffic Engineering in MPLS Networks,* Proceedings of SCS SPECTS 2001, Orlando, Florida, USA, November 2001.

[Ban01c]. G. Banerjee and D. Sidhu, *Path Computation for Traffic Engineering in MPLS Networks,* Proceedings IEEE ICN 2001, CREF, Colmar, France, July 2001.

[Bel01]. J. Bell, C. Hobbs, *WiseNet Architecture,* Documents of Nortel Networks (Architecture SN2001-001, Issue A), November 2001.

[Ber03]. L. Berger, RFC3473, *Generalized Multi-Protocol Label Switching (GMPLS) Signaling Resource ReserVation Protocol-Traffic Engineering (RSVP-TE) Extensions,* January, 2003

[Cal97]. K. L. Calvert, M. B. Doar and E. W. Zegura, *Modeling Internet Topology,* IEEE Communications, 35(6), pp. 160-163, June 1997.

[Cas90]. D. Castanon, *Efficient Algorithms for Finding the K Best Paths through a Trellis,* IEEE Trans. AES 26, pp. 405-410, 1990.

[Cav98]. D. Cavendish and M. Gerla, *Internet QoS Routing using the Bellman-Ford Algorithm,* Proceedings of IFIP Conference on High Performance Networking, Vienna, Austria, September 1998.

[Cha00]. S. Chaudhuri, G. Hjálmtýsson and J. Yates, Internet Draft, *Control of Lightpaths in an Optical Network,* draft-chaudhuri-ip-olxc-control-00.txt, March 2000.

[Che00]. T. Chen, J. Kennington and T. Oh, *Fault Restoration Techniques for MPLS with QoS Constraints,* GLOBECOM 2000, San Francisco, CA, USA, December 2000.

[Che97]. C. S. Chekuri, A. V. Goldberg, D. R. Karger, M. S. Levine and C. Stein, *Experimental Study of Minimum Cut Algorithms,* Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms, New Orleans, LA USA, pp. 324-333, 1997.

[Che99]. T. M. Chen and T. H. Oh, *Reliable Services in MPLS,* IEEE Communications Magazine, Vol.37, pp. 58-62, December 1999.

[Cho93]. C. E. Chow, S. McCaughey and S. Syed, *RREACT: A Distributed Protocol for Rapid Restoration of Active Communication Trunk,* UCCS Technical Report EAS-CS-92-18, Results published in the Proceedings of Second IEEE Network Management and Control Workshop, Westchester, New York, September, 1993.

[Col87]. C. J. Colbourn, *The Combinatorics of Network Reliability,* New York, Oxford, Oxford University Press, 1987.

[Cor90a]. T. H. Cormen, C. E. Leiserson and R. L. Rivest, *Introduction to Algorithms,* the MIT Press, Chapter 25, Single-Source Shortest Paths, pp. 514-549, 1990.

[Cor90b]. T. H. Cormen, C. E. Leiserson and R. L. Rivest, *Introduction to Algorithms,* the MIT Press, Chapter 13, Binary Search Trees, pp. 244-262, 1990.

[Cra98]. E. Crawley, R. Nair, B. Rajagopalan and H. Sandick, RFC 2386, *A Framework for QoS-based Routing in the Internet,* August 1998.

[Dij59]. E. W. Dijkstra, *A Note on Two Problems in Connexion with Graphs,* Numerische Mathematik, pp. 269-271, 1959.

[Din70]. E. A. Dinic, *Algorithm for Solution of a Problem of Maximum Flow in a Network with Power Estimation,* Soviet Math., Dokl., 11(1970), pp. 248-264.

[Doa96]. M. B. Doar, *A Better Model for Generating Test Networks,* Proceedings of Globecom'96, pp. 86-93, London, UK, November 1996.

[Dov01]. R. Doverspike and J. Yates, *Challenges for MPLS in Optical Network Restoration,* IEEE Communications Magazine, Vol.39, pp. 89-96, February 2001.

[Epp99]. D. Eppstein, *Finding the K shortest Paths,* SIAM J. Computing, 28(2): 652-673, 1999.

[Erd59]. P. Erdos and A. R'enyi, *On Random Graphs,* Publicationes Mathematicae Universitatis Debreceniensis 6, pp. 290-297, 1959

[Far03]. A. Farrel, RFC 3479, *Fault Tolerance for the Label Distribution Protocol (LDP),* February 2003.

[For56]. L. R. Ford and D.R. Fulkerson, *Maximum Flow Through a Network,* Canadian Journal of Mathematics, 8 (1956), pp. 399-404.

[Fre80]. J. E. Freund and R. E. Walpole, *Mathematical Statistics,* 3rd edition, chapter 2, pp. 24-59, 1980.

[Gol88]. A. V. Goldberg and R. E. Tarjan, *A New Approach to the Maximum-Flow Problem,* Journal of the Association for Computing Machinery, Vol. 35, No. 4, pp. 921-940, October 1988.

[Gol98]. A. V. Goldberg, *Recent Developments in Maximum Flow Algorithms,* Technical Report, No.98-045, Nec Research Institute, April 1998.

[Gro02]. W. D. Grover and J. E. Doucette, *Advances in Optical Network Design with p-Cycles: Joint optimization and pre-selection of candidate p-cycles,* Proceedings of the IEEE-LEOS Summer Topical Meeting on All Optical Networking, Mont Tremblant, Quebec, July 2002.

[Gue97]. R. Guerin, A. Orda, and D. Williams, *QoS Routing Mechanisms and OSPF Extensions,* in IEEE GLOBECOM'97, vol.3, pp. 1903-1908, Phoenix, Arizona, November 1997.

[Has00]. D. Haskin and R. Krishnan, *A Method for Setting an Alternative Label Switched Paths to Handle Fast Reroute,* draft-haskin-mpls-fast-reroute-04.txt, Internet Draft, May 2000.

[Has85]. R. Hassin and D. B. Johnson, *An O(nlog2n) Algorithm for Maximum Flow in Undirected Planar Networks,* SIAM J. Comput., Vol. 14, No. 3, pp. 612-624, August 1985.

[Hed88]. C. Hedrick, RFC 1058, *Routing Information Protocol,* June 1988.

[Hil03]. M. A. Hiltunen, R. D. Schlichting and C. A. Ugarte, *Building Survivable Services Using Redundancy and Adaptation,* IEEE Transactions on Computers, Vol. 52, No. 2, pp. 181-194, February, 2003.

[Hui95]. C. Huitema, *Routing in the Internet,* Prentice-Hall, 1995.

[ISO02]. ISO 10589, *Information technology -- Telecommunications and information exchange between systems -- Intermediate System to Intermediate System intra-domain routing information exchange protocol for use in conjunction with the protocol for providing the connectionless-mode network service (ISO 8473)*, 2002.

[Iwa00]. A. Iwata, N. Fujita, *A hierarchical multilayer QoS routing system with dynamic SLA management,* IEEE Journal on Selected Areas in Communications, 18 (12), pp. 2603-2616, December 2000.

[Jam02]. B. Jamoussi and et. al., RFC 3212, *Constraint-Based LSP Setup using LDP,* January 2002.

[Jim99]. V.M. Jimenez and A. Marzal, *Computing the K Shortest Paths: a New Algorithm and an Experimental Comparison,* Springer-Verlag, Lecture Notes in Computer Science Series, Vol. 1688, pp. 15-29, 1999.

[Kar00]. K. Kar, M. Kodialam and T. V. Lakshman, *MPLS Traffic Engineering using Enhanced Minimum Interference Routing: An Approach Based on Lexicographic Max-Flow,* IEEE Journal on Selected Areas in Communications, Vol.18, No.12, pp. 2566-2576, December 2000.

[Kha02]. S. Khan, K. F. Li, E. G. Manning, M. M. Akbar, *Solving the Knapsack Problem for Adaptive Multimedia System,* Studia Informatica, 2002.

[Kha97]. S. Khan, K. F. Li, E. G. Manning, *The Utility Model for Adaptive Multimedia Systems,* presented at the International Conference on Multimedia Modeling, pp. 111-126, Singapore, November 1997.

[Kha98]. S. Khan. *Quality Adaptation in a Multi-Session Adaptive Multimedia System: Model and Architecture,* Ph.D. Dissertation, Department of Electrical and Computer Engineering, University of Victoria, 1998.

[Kin01]. S. Kini, M. Kodialam, S. Sengupta and C. Villamizar, *Shared backup Label Switched Path restoration,* draft-kini-restoration-shared-backup-01.txt, Internet Draft, May 2001.

[Koc98]. R. Koch, *The 80/20 Principle: The Secret of Achieving More with Less,* Bantam Doubleday Dell Pub (Trd); ISBN: 0385491700, 1998.

[Kod00]. M. Kodialam and T. V. Lakshman, *Minimum Interference Routing with Applications to MPLS Traffic Engineering,* IEEE INFOCOM'00, Tel-Aviv, Israel, pp. 884-893, March 2000.

[Kom02]. K. Kompella and Y. Rekhter, Internet Draft, *OSPF Extensions in Support of Generalized MPLS,* draft-ietf-ccamp-ospf-gmpls-extensions-09.txt, December 2002.

[Kub97]. M. Kub, X. Zhongyi and X. Xiaodong, *A Minimax Method for Finding the K Best "Differentiated" Paths,* Geographical Analysis, Vol.29, No.4, pp. 298-313, October 1997.

[Lee01]. S. S. Lee and M. Gerla, *Fault Tolerance and Load Balancing in QoS Provisioning with Multiple MPLS Paths,* Proceedings of 9th International Workshop on Quality of Service (IWQoS), Karlsruhe, Germany, June 2001.

[Lee99]. S. W. Leet and C. S. Wu, *A K-best Paths Algorithm for Highly Reliable Communication Networks,* IEICE Transactions on Communications, E28B(4), pp. 586-590, April 1999.

[Li02]. G. Li, D. Wang, C. Kalmanek, and R. Doverspike, *Efficient Distributed Path Selection for Shared Restoration Connections,* Proceedings of INFOCOM 2002, New York, USA, June 2002.

[Liu01]. Z. Liu, M. S. Squillante and J. L. Wolf, *On maximizing service-level-agreement profits,* Proceedings of the 3rd ACM conference on Electronic Commerce, pp. 213-223, Tampa, Florida, USA, October 2001.

[Lot03]. M. K. Lottor, *Internet Domain Survey,* Network Wizards, URL: http://www.nw.com/, January 2003.

[Mal98].G. Malkin, RFC 2453, *RIP Version 2,* November 1998.

[Man03]. E. Mannie, Internet Draft, *Generalized Multi-Protocol Label Switching Architecture,* draft-ietf-ccamp-gmpls-architecture-05.txt, March 2003.

[Mar90]. S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations (Wiley Interscience Series in Discrete Mathematics and Optimization),* published by John Wiley & Sons, November 1990.

[Mar99]. E.Q.V. Martins, M.M.B. Pascoal, and J.L.E. Santos, *Deviation Algorithms for Ranking Shortest Paths,* International Journal of Foundations of Computer Science, Vol. 10, No. 3 (1999), pp247-261.

[Mcq80]. J. M. Mcquillan and E. C. Rosen, *The New Routing Algorithm for the ARPANET,* IEEE Transactions on Communications, Vol. 28, No. 5, pp. 711-719, May 1980.

[Moy98a]. J. Moy, *OSPF: Anatomy of an Internet Routing Protocol,* Addison-Wesley, February 1998.

[Moy98b]. J. Moy, RFC 2328, *OSPF Version 2,* April 1998.

[Mul99]. N. J. Muller, *Managing service level agreements,* International Journal of Network Management, Vol. 9 , Issue 3, pp. 155-166 , May–June 1999.

[Nag01]. N. Nagy and S. G. Akl, *The Maximum Flow Problem: A Real-Time Approach,* Proceedings of 13th Conference on Parallel and Distributed Computing and Systems, Anaheim, California, USA, August 2001.

[Nar00a]. P. Narvaze, K. Y. Siu, and H. Y. Tzeng, *Fault-tolerant Routing in the Internet without Flooding,* in Dependable Network Computing, Aversky (ed.), Kluwer Academic Publisher, pp. 139-206, 2000.

[Nar00b]. P. Narvaze, K. Y. Siu, and H. Y. Tzeng, *New Dynamic Algorithms for Shortest Path Tree Computation,* IEEE/ACM Transactions on Networking, Vol.8, No.6, December 2000.

[Nar99a]. P. Narvaez, K. Y. Siu, H. Y. Tzeng, *New Dynimic SPT Algorithm based on a Ball-and-String Model,* Proceedings of the Conference on Computer Communications (IEEE Infocom), New York, March 1999.

[Nar99b]. P. Narvaez, K. Y. Siu, H. Y. Tzeng, *New Dynimic SPT Algorithm based on a Ball-and-String Model,* Technical report. December 1999.

[Nex01]. NextHop Technologies, *GateD™ Suite of routing protocols,* Version 10.0, URL: http://www.gated.org/, 2002.

[Nik97]. S. Nikolopoulos, A. Pitsillides and D. Tipper, *Addressing Network Survivability Issues by Finding the K-best Paths through a Trellis Graph,* IEEE INFOCOM'97, Kobe, Japan, April 1997.

[Nor01]. S. Norden, M. M. Buddhikot, M. Waldvogel and S. Suri, *Routing Bandwidth Guaranteed Paths with Restoration in Label Switched Networks,* the 9th International Conference on Network Protocols (ICNP 2001), Riverside, pp. 71-79, CA, USA, November 2001.

[Nua02]. Scope Communications Group, *Nua Internet Surveys*, URL: http://www.nua.ie/surveys/how_many_online/index.html, September 2002.

[Pap01]. D. Papadimitriou and et. al., Internet Draft, *Inference of Shared Risk Link Groups,* draft-many-inference-srlg-02.txt, November 2001.

[Pu01a]. J. Pu, E. Manning, G. C. Shoja, *Routing Reliability Analysis of Partially Disjoint Paths,* 2001 IEEE Pacific Rim Conference on Communications, Computers and Signal processing (PACRIM' 01), Vol. 1, pp. 79-82, Victoria, BC, Canada, August 2001.

[Pu01b]. J. Pu, E. Manning, G. C. Shoja, A. Srinivasan, *A New Algorithm to Compute Alternate Paths in Reliable OSPF (ROSPF),* 2001 International Conference on

Parallel and Distributed Processing Techniques and Applications (PDPTA'2001), Vol. 1, pp. 299-304, Las Vegas, Nevada, USA, June 2001.

[Pu02]. J. Pu, E. Manning, G. C. Shoja, *Reliable Routing in MPLS Networks,* IASTED International Conference on Communications and Computer Networks (CCN 2002), pp. 317-322, Cambridge, Massachusetts, USA, October 2002.

[Pu03a]. J. Pu, M. Akbar, E. Gowland, E. Manning, G. C. Shoja, *A SLA Admission Controller for Reliable MPLS Networks,* IASTED International Conference on Parallel and Distributed Computing and Networks (PDCN'2003), Innsbruck, Austria, February 2003.

[Pu03b]. J. Pu, M. Akbar, E. Manning, G. C. Shoja, K. F. Li and E. Gowland, *A Reliable SLA-based Admission Controller for MPLS Networks,* Submitted to the Journal of Systems and Software (North-Holland), the Special Issue of Adaptive Multimedia Computing, February 2003.

[Rek95]. Y. Rekhterm and T. Li, RFC 1771, *A Border Gateway Protocol 4 (BGP-4),* March 1995.

[Ros01]. E. Rosen, A. Viswanathan and R. Callon, RFC 3031, *Multiprotocol Label Switching Architecture,* January 2001.

[Rou95]. N. M. Rouphail, S. Ranjithan, W. El Dessouki, T. Smith, and E. D. Brill Jr., *A Decision Support System for Dynamic Pre-trip Route Planning,* Proceedings of the 4th International Conference on Applications of Advanced Technologies in Transportation Engineering, Capri, Italy, June 1995.

[Sha01]. O. Sharon, *Dissemination of Routing information in Broadcast Networks: OSPF versus IS-IS,* IEEE Network, pp. 56-65, January/February, 2001.

[Sha03]. V. Sharma and F. Hellstrand, RFC 3469, *Framework for Multi-Protocol Label Switching (MPLS)-based Recovery,* February 2003.

[Seg98]. A. Segall, P. Bhagwat and A. Krishna, *QoS Routing Using Alternate Paths,* Journal of High Speed Networks, 7(2):141-158, 1998.

[Sta00a]. D. Stamatelakis and W. D. Grover, *Theoretical Underpinnings for the Efficiency of Restorable Networks Using Pre-configured Cycles ("p-cycles"),* IEEE Transactions on Communications, vol.48, no.8, pp. 1262-1265, August 2000.

[Sta00b]. D. Stamatelakis and W. D. Grover, *IP Layer Restoration and Network Planning Based on Virtual Protection Cycles,* IEEE JSAC Special Issue on Protocols and Architectures for Next Generation Optical WDM Networks, Vol.18, No.10, pp. 1938-1949, October 2000.

[Sta99]. D. Stamatelakis and W. D. Grover, *Rapid Restoration of Internet Protocol Networks using Pre-configured Protection Cycles,* Proceedings of 3rd Canadian Conference On Broadband Research (CCBR'99), Ottawa, pp. 33-44, November 1999.

[Ste93a]. M. Steenstrup, RFC 1478, *An Architecture for Inter-Domain Policy Routing,* June, 1993.

[Ste93b]. M. Steenstrup, RFC 1479, *Inter-Domain Policy Routing Protocol Specification, Version 1,* July, 1993.

[Sun77]. C. A. Sunshine, *Source Routing in Computer Networks,* Computer Communication Review, vol. 7, pp. 29-33, January 1977.

[Suu74]. J. W. Suurballe, *Disjoint Paths in a Network,* Networks, 4 (1974), pp. 125-145.

[Suu84]. J. W. Suurballe and R.E. Tarjan, *A Quick Method for Finding Shortest Pairs of Disjoint Paths,* Networks, 14 (1984), pp. 325-336.

[Tan88]. A. S. Tanenbaum, *Computer Networks,* Chapter 5, Section 5.2.2, *Multipath Routing,* pp. 291-294, 2nd edition, Prentice-Hall, 1988.

[Tan96]. A. S. Tanenbaum, *Computer Networks,* Chapter 5, Section 5.4.5, *Internetworking Routing,* pp. 405-406, 3rd edition, Prentice-Hall, 1996.

[Vee94]. J. Veerasamy and S. Venkatesan, *Effect of Traffic Splitting on Link and Path Restoration Planning,* IEEE GLOBECOM'94, Vol.3, pp. 1867-1871, San Francisco, CA, USA, November 1994.

[Vee99]. J. Veerasamy and S. Venkatesan, *Spare Capacity Assignment in Telecom Networks using Path Restoration,* the Journal of System and Software, Vol.47, No.1, pp. 27-33, 1999.

[Wal97]. K. V. D. Wal, M. Mandjes and H. Bastiaansen, *Delay Performance Analysis of the New Internet Services with Guaranteed QoS,* Proceedings of the IEEE, Vol.85, No.12, pp. 1947-1957, December 1997.

[Wan90]. Z. Wang and J. Crowcroft, *Shortest Path First with Emergency Exists,* Proceedings of the ACM Symposium on Communications Architectures & Protocols, Sep. 1990, Philadelphia, USA, pp. 166-176.

[Wan92]. Z. Wang and J. Crowcroft, *Analysis of shortest-path routing algorithms in a dynamic network environment,* ACM Computer Communication Review, vol. 22, pp. 63-71, April 1992.

[Wat01]. R. K. Watson, *Applying the Utility Model to IP Networks: Optimal Admission & Upgrade of Service Level Agreements,* Master thesis, Dept of Electrical and Computer Engineering, University of Victoria, April 2001.

[Wax88]. B. M. Waxman. *Routing of multipoint connections,* IEEE Journal on Selected Areas in Communications, 6(9), pp. 1617-1622, December 1988.

[Wu01]. J. Wu, X. Lin, J. Cao, and W. Jia, *An extended Fault-Tolerant Link-State Routing Protocol in the Internet,* International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2001), Las Vegas, Nevada, USA, June 2001.

[Yen71]. J. Y. Yen, *Finding the K Shortest Loopless Paths in a Network,* Management Science, Vol.17, No.11, pp. 712-716, July 1971.

[Zeg96]. E. W. Zegura, K. L. Calvert and S. Bhattacharjee, *How to Model an Internetwork,* Proceedings of IEEE Infocom, pp. 594-602, March 1996.

[Zha93]. L. Zhang, S. Deering, D. Estrin, S. Shenker and D. Zappala, *RSVP: A New Resource ReSerVation Protocol,* IEEE Network, Vol.7, pp. 8-18, September 1993.

10. Appendices

10.1 Appendix A, Calculating Connection Reliability

Assume now we have three paths for a given node pair *(s, t)*. Shared edges make the paths interdependent. If a shared edge fails, multiple paths will be affected. Figures 4.3 and 4.4 illustrate the types of shared edge which may be present. We use the notions described in Section 4.5.1, and apply the 3-D Venn Diagram model to calculate connection reliability *R*, which is the probability that the connection from *s* to *t* still exists.

10.1.1 3-D Venn Diagram

A Venn Diagram model is very useful in calculating the probability of dependent events. A 3-D Venn Diagram [Fre80] is shown in Figure 10.1. Events 1, 2 and 3 represent three dependent events. Let $P_r(1)$, $P_r(2)$ and $P_r(3)$ represent the occurrence probability of events 1, 2 and 3, respectively. The overall event occurrence probability of the system can be calculated using the 3-D Venn Diagram model.

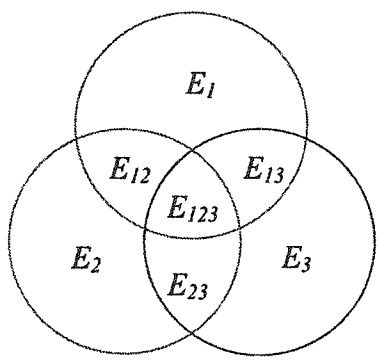


Figure 10.1. 3-D Venn Diagram. Events 1, 2 and 3 represent three dependent events.

We introduce the following definitions in the 3-D Venn Diagram:

S_0 = *the probability of occurrence of events 1, 2 or 3 independently*

$= P_r(E_1) + P_r(E_2) + P_r(E_3)$ *(A.1)*

S_1 = *the probability of occurrence of events 1 and 2, or 1 and 3, or 2 and 3 simultaneously*

$= P_r(E_{12}) + P_r(E_{13}) + P_r(E_{23})$ *(A.2)*

S_2 = *the probability of occurrence of events 1 and 2 and 3 simultaneously*

$= P_r(E_{123})$ *(A.3)*

Using the Venn Diagram model, *S*, the probability of occurrence of any non-null subset of events *1*, *2* and *3*, can be obtained by the following formula:

$$S = S_0 - S_1 + S_2 \quad (A.4)$$

10.1.2 Deriving Mathematical Formulae

We use *F* and *R* to represent the overall probabilities of connection failure and success from *s* to *t*. *R* is the connection reliability and *F = (1–R)*. The 3-D Venn Diagram model is used to calculate *R*. First of all, we have to figure out the three dependent events that may lead to overall connection success from *s* to *t*:

- ***Event 1:*** the connection success caused by success of path P_1

  $P_r(1)$ = *probability of path* P_1 *success* $= p^{m1}$ *(A.5)*

- ***Event 2:*** the connection success caused by success of path P_2

  $P_r(2)$ = *probability of path* P_2 *success* $= p^{m2}$ *(A.6)*

- ***Event 3***: the connection success caused by success of path P_3

$$P_r(3) = probability\ of\ path\ P_3\ success = p^{m3} \tag{A.7}$$

Referring to our notation and Figure 4.4 in Section 4.5, we can calculate the probabilities: $P_r(12)$, $P_r(13)$, $P_r(23)$ and $P_r(123)$

$$P_r(12) = probability\ of\ connection\ success\ using\ both\ P_1\ and\ P_2\ simultaneously$$
$$= p^{(L1+L2+L123+L12+L13+L23)}$$
$$= p^{(m1+m2)-(L123+L12)} \tag{A.8}$$

$$P_r(13) = probability\ of\ connection\ success\ using\ both\ P_1\ and\ P_3\ simultaneously$$
$$= p^{(L1+L3+L123+L12+L13+L23)}$$
$$= p^{(m1+m3)-(L123+L13)} \tag{A.9}$$

$$P_r(23) = probability\ of\ connection\ success\ using\ both\ P_2\ and\ P_3\ simultaneously$$
$$= p^{(L2+L3+L123+L12+L13+L23)}$$
$$= p^{(m2+m3)-(L123+L23)} \tag{A.10}$$

$$P_r(123) = probability\ of\ connection\ success\ using\ P_1,\ P_2\ and\ P_3\ simultaneously$$
$$= p^{(L1+L2+L3+L123+L12+L13+L23)}$$
$$= p^{(m1+m2+m3)-(2*L123+L12+L13+L23)} \tag{A.11}$$

Therefore, according to the 3-D Venn Diagram model,

$$S_0 = P_r(1) + P_r(2) + P_r(3) = p^{m1} + p^{m2} + p^{m3} \tag{A.12}$$

$$S_1 = P_r(12) + P_r(13) + P_r(23)$$
$$= p^{(m1+m2)-(L123+L12)} + p^{(m1+m3)-(L123+L13)} + p^{(m2+m3)-(L123+L23)} \tag{A.13}$$

$$S_2 = P_r(123) = p^{(m1+m2+m3)-(2*L123+L12+L13+L23)} \tag{A.14}$$

So, the connection probability R and the connection failed probability F can be calculated by the following formulae:

$$R = S = S_0 - S_1 + S_2 \qquad (A.15)$$

$$F = (1 - R) \qquad (A.16)$$

In some special cases, simplified formulae may be derived from Formula *A.15* to calculate the reliability of the overall connection from source s to destination t, as shown below.

(1). The three paths (P_1, P_2 and P_3) are disjoint (case 1 in Figure 4.3)

$$R = 1 - (1 - p^{m1}) \cdot (1 - p^{m2}) \cdot (1 - p^{m3}) \qquad (A.17)$$

(2). Only triple-shared edges exist among P_1, P_2 and P_3 (case 2 in Figure 4.3)

$$R = p^{L123} \cdot [1 - (1 - p^{m1\text{-}L123}) \cdot (1 - p^{m2\text{-}L123}) \cdot (1 - p^{m3\text{-}L123})] \qquad (A.18)$$

(3). Only double-shared edges between P_1 and P_2 exist (case 3 in Figure 4.3)

$$R = 1 - (1 - p^{m3}) \cdot \{1 - p^{L12} \cdot [1 - (1 - p^{L1}) \cdot (1 - p^{L2})]\} \qquad (A.19)$$

10.2 Appendix B, Random Graph Generator

The random graph generator is a part of our alternate path routing simulation platform. It generates random network topologies for the simulations. This appendix describes in detail how we implement a random graph generator based on existing network topology models.

10.2.1 Random Network Topology Models

Recently, several hierarchical network topology models have been developed [Doa96, Zeg96, Cal97]. These topology generation models can fit the simulation needs of hierarchical routing protocols such as OSPF. However, for an enterprise network containing less than *100* nodes, edges and nodes may be placed in a rather free manner, instead of a hierarchical topology. A reasonable requirement for such network topology is that there should be much fewer long edges than short edges. Based on this observation, we can choose a simple random graph model for topology generation.

The most common random graph model is one proposed by Waxman [Wax88], with the probability of an edge from node u to node v given by:

$$P(u, v) = \alpha \times exp(-d / (\beta \times L)) \qquad (B.1)$$

where $0<\alpha, \beta<1$ are parameters of the model, d is the Euclidean distance from u to v, and L is the maximum distance between any two nodes in the graph, i.e., L is the maximum separation in the graph.

An increase in α will increase the number of edges in the graph, while an increase in β will increase the ratio of long edges relative to short edges. In Waxman's model, given the differences in edge probability functions, one might have expected more differences in properties of the graphs generated; moreover, different L values do not cause noticeable changes in the graphs generated [Zeg96]. These are the reasons why we choose this model.

We also use another simple model, the pure random model [Erd59], to give the probability of an edge *(u, v)* occurring as:

$$P(u, v) = p, \qquad \text{where } 0 < p < 1, p \text{ is a constant.} \qquad (B.2)$$

Our graph generator is based on these two models. Let $l = (\beta \times L)$, then we have:

$$P(u, v) = \alpha \qquad \text{when } d \leq l \qquad (B.3)$$

$$P(u, v) = \alpha \times exp(-d / l) \qquad \text{when } d > l \qquad (B.4)$$

The Euclidean distance d of an edge is used as the edge length. Any two nodes within distance l of each other have a constant probability of being joined by an edge. Likewise, nodes with a separation greater than l have a probability of being connected which decays exponentially with their separation. The distance l plays a role as a region threshold. An edge whose length is shorter than l is called a short edge. It will be picked randomly according to a constant probability value α. An edge whose length is longer than l is called a long edge. It will be placed in the graph according to an exponentially decreasing probability based on the edge length. Heuristically, for graphs with the same maximum separation L, a longer threshold distance l should be chosen when a graph has fewer nodes or has denser edges, while a shorter l should be applied for a graph that contains more nodes or sparser edges. The parameters α and l are adjustable in our implementation. If an edge is too short, it should be discarded. This constraint has been used in the graph generator to limit the number of edges that are too short.

10.2.2 Implementing Waxman's Model

In our implementation of the graph generator, there are two input parameters: N (the number of node in the graph) and d (the edge density in the graph, a percentage value). Formulae B.5-B.7 describe the relationships among M (the number of edges in the graph), N, d and $avgD$ (average degree of all nodes in the graph).

$$M = d \times N \times (N - 1) / 2 \qquad (B.5)$$

$$avgD = 2M / N = d \times (N - 1) \qquad (B.6)$$

$$M = (N \times avgD) / 2 \qquad (B.7)$$

The generated random graph will be put into a $K \times K$ grid. The threshold distance l can be calculated as follows [Doa96, Zeg96] *:

$$l = S \times h_0 = S \times (K / sqrt(N)) \qquad (B.8)$$

We can set $S_1 = S = 3$ as the initial threshold for the long edge $l_1 = (S_1 \times h_0)$, i.e., an edge will be counted as a long edge if its length is bigger than l_1, and the probability that this edge will be placed in the graph follows from Formula B.4. In the implementation, S_1 can be changed according to the generated edges. If the obtained graph contains too few edges, we slightly increase the value of S_1; otherwise we decrease S_1 a little. Furthermore, we set $S_2 = S = 0.2$ as the threshold for edges to be too short, i.e., an edge will be discarded if its length is less than $l_2 = (S_2 \times h_0)$. The size of the grid is set as $K = 300$ to be convenient for drawing the pictures of generated graphs with enough separations between nodes.

At first, the graph generator randomly places the desired number of nodes N in the grid. Then the distance between every two nodes is checked and an edge is placed between them according to the probability described in Formulae B.3 and B.4. Because the number of edges M' in the generated graph may not be equal to the desired number of edges M, the software will adjust the value of α and S_1 such that $/ M' - M / / M \leq T_e$ is true. Then the software will stop and output the generated graph. T_e is a predefined threshold to determine whether the generated number of edges M' in the graph is acceptable. We set it to $T_e = 2\%$ in the implementation.

---

* In the $K \times K$ grid, suppose the N nodes are evenly distributed on the K^2 rectangles (i.e., cells) of the grid, then we will have $sqrt(N)$ nodes in the cells in the horizontal (or vertical) direction, each with a point in a cell center. The distance from point to point (a neighbor cell) is $h_0 = K / sqrt(N)$. Based on the variable h_0, the threshold distance l can be preset in a reasonable manner.

10.2.3 Validation of the Implementation

Figure 10.2 shows a sample generated random graph with the number of nodes $N = 100$ and edge density $d = 5\%$ (i.e., the number of edges $M = 248$). Figure 10.3 shows the distribution of long edges (i.e., edge length exceeds the threshold l_1). From this figure, we can observe that such distribution is exponential.

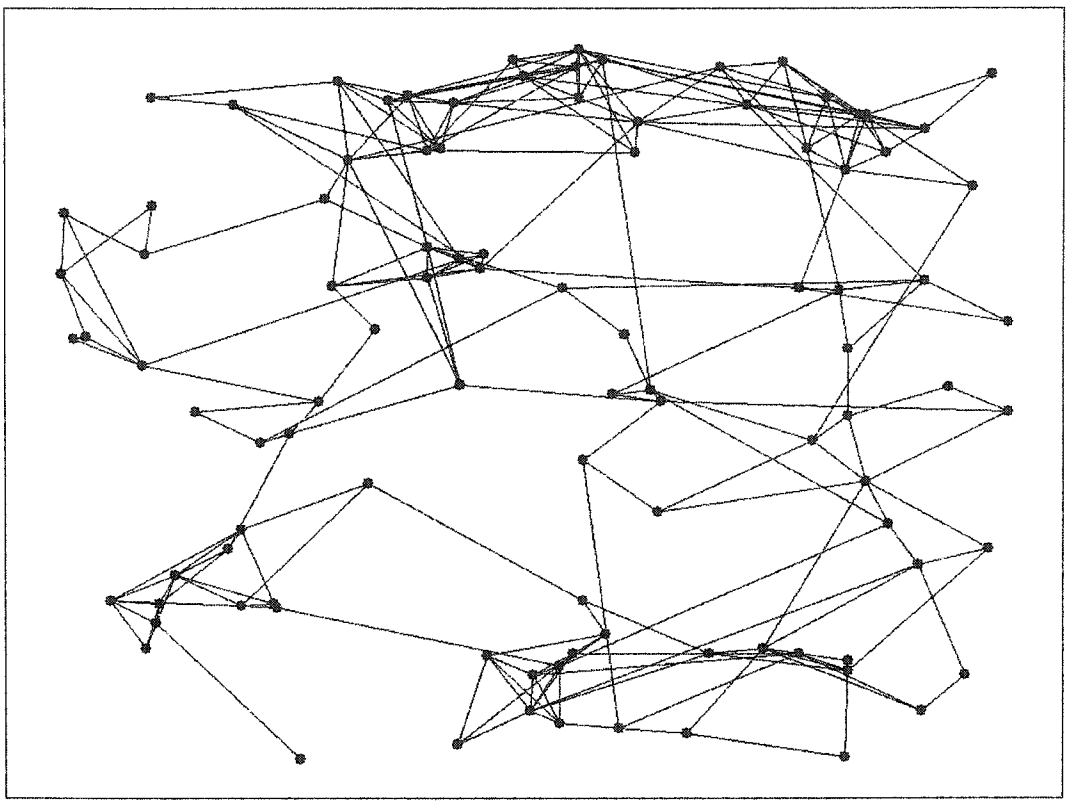


Figure 10.2. A sample generated random graph with 100 nodes and 5% edge density

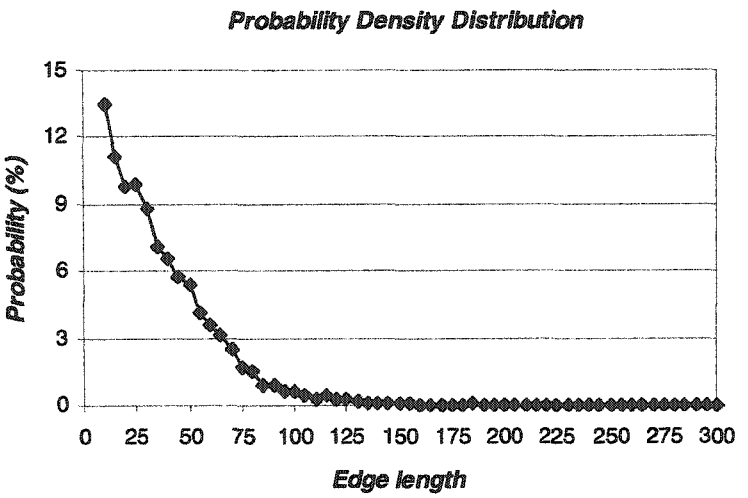


Figure 10.3. Exponential distribution of edge lengths for the graph in Figure 10.2

10.3 Appendix C, More Simulations on R-SLAOpt

This appendix contains more simulation results for R-SLAOpt, the proposed reliable admission control approach. Our GAPA algorithm was used to calculate multiple acceptable candidate path groups subject to predefined constraints on QoS levels and reliability requirements. The simulations to analyze features of GAPA and QoS adaptation in R-SLAOpt were performed. Also, the simulation results illustrated the influence on the algorithm results from different inputs of group number and group size. All simulation conditions were the same as those in Section 7.4.

10.3.1 Features of the Calculated Working Path Groups

The shortest path between a given pair of nodes is usually considered preferable by traditional routing technologies when transmitting data flows. It is therefore meaningful to investigate how often the shortest path was included in the selected working groups when R-SLAOpt finishes the QoS adaptation. By using the same simulation conditions as the previous sections, the results show that the probability of including the shortest path was about *58%* when group size $k = 2$. When $k = 3$, the probability was about *75%*. When a working path group contains the shortest path, that path may be chosen as the primary working path, depending upon the path-selection policy in R-SLAOpt. In the simulation implementation, this was the case. These results indicate that R-SLAOpt takes advantage of shortest path routing in most cases.

Table 10.1 shows the link type distribution amongst the paths selected by R-SLAOpt. There are three possible distinct link types in a path group. *Independent Links* (IndepL) are links utilized by only one path in the group, *Double-Shared Links* (DblSL) are links shared by two paths, and *Triple-Shared Links* (TriSL) are links shared by three paths. From this table, it can be seen that the majority of the links in each path group were of the independent type, which is expected and benefits routing reliability.

Table 10.2 illustrates further analysis of the number of shared links in the selected working groups. For simplicity and clarity, group size was set to $k = 2$. Thus, no *Triple-*

Shared Links would exist. Column 2 shows the probability that a working group only contains *Independent Links*. In other words, only disjoint paths are included in these groups. Columns 3-6 show the probabilities that a group contains only *1, 2, 3,* or *4* double-shared links, respectively. There was about a *50%* probability that the selected working path groups contain only disjoint paths. Groups with more than *3* double-shared links were fairly uncommon.

Table 10.1. The distribution of link types in the selected working groups.

	Group size k = 2		*Group size k = 3*		
	IndepL	*DblSL*	*IndepL*	*DblSL*	*TriSL*
Probability of link type distribution	*87.62%*	*12.38%*	*55.89%*	*37.06%*	*7.05%*

Table 10.2. Double-shared links in the selected working path groups (k = 2).

Working groups contain only ...	*Disjoint paths*	*Number of DblSL = 1*	*Number of DblSL = 2*	*Number of DblSL = 3*	*Number of DblSL = 4*
Occurrence probability	*50.46%*	*33.17%*	*13.11%*	*2.67%*	*0.58%*

10.3.2 Algorithm Execution Time

Table 10.3 compares execution times * of one initial QoS adaptation for SLAOpt and R-SLAOpt. It can be observed that the proposed algorithm GAPA in R-SLAOpt will take up to *72%* more time than that of the MPS *K*-shortest paths algorithm (shown in Section 10.4, Appendix D) in SLAOpt. This is expected as GAPA effectively adds several additional steps to MPS. However, it can also be seen that the total time for QoS adaptation in R-SLAOpt is less than that in SLAOpt. Although this might at first seem

* The simulations of SLAOpt and R-SLAOpt were implemented in Java, and ran on a desktop computer (650MHZ Pentium III CPU, 128MB main memory, and Windows 2000 operating system) and Java™ 2 Platform, Standard Edition (J2SE™) 1.4.1_01.

unusual, it is consistent with the way the two admission controllers work. The explanation is as follows.

In this simulation, SLAOpt always considered up to five shortest paths from the MPS algorithm. R-SLAOpt, on the other hand, considered up to five path groups. From the point of view of the internal admission algorithm (i.e., I-HEU in this case), both these situations are simply a choice between sets of links. However, since each path group contains three paths, there is a greater probability that a path group will violate the end-to-end delay constraint and thus be eliminated as a valid choice by the admission controller. The admission algorithm will require less time to select the optimal path group in R-SLAOpt, as it will likely have fewer choices. Moreover, when each path group contains more paths, the maximum number of SLAs that can be admitted will be reached more quickly, as each SLA requires more resources. Hence, fewer SLAs will be accepted for bigger group sizes. Using SLAOpt as a baseline (i.e., $k = 1$), we observed that about *84%* and *70%* of the SLAs are accepted when $k = 2$ and $k = 3$, respectively. As a result, the execution time will be reduced further because the most optimal solution will consist of fewer SLAs and be reached earlier.

Table 10.3. Comparisons of time consumption between SLAOpt and R-SLAOpt.

m (# of SLA)	*SLAOpt (in seconds)*			*R-SLAOpt (in seconds)*			T_1' / T_1 *(comparisons of time costs for GAPA and MPS)*
	T *Total time costs*	T_1 *Only for MPS*	T_1 / T *(%)*	T' *Total time costs*	T_1' *Only for GAPA*	T_1' / T' *(%)*	
150	*18.4*	*1.25*	*6.79*	*16.1*	*1.40*	*8.69*	*1.60*
300	*31.3*	*2.55*	*8.15*	*23.5*	*2.39*	*10.2*	*1.62*
450	*57.1*	*3.48*	*6.09*	*49.6*	*3.91*	*7.88*	*1.72*
600	*104.2*	*4.11*	*3.94*	*99.2*	*5.30*	*5.34*	*1.29*
750	*172.8*	*7.46*	*4.32*	*144.9*	*9.17*	*6.32*	*1.23*
900	*265.1*	*13.32*	*5.03*	*203.8*	*16.05*	*7.87*	*1.20*
1050	*343.8*	*28.99*	*8.43*	*281.1*	*32.55*	*11.58*	*1.12*
1200	*473.2*	*35.20*	*7.44*	*404.1*	*40.00*	*9.90*	*1.14*
1350	*552.3*	*43.96*	*7.96*	*473.1*	*49.92*	*10.55*	*1.14*
1500	*675.3*	*50.20*	*7.43*	*566.5*	*57.36*	*10.13*	*1.14*
1650	*804.4*	*54.01*	*6.71*	*680.0*	*62.66*	*9.22*	*1.16*
1800	*983.9*	*72.80*	*7.40*	*825.7*	*83.20*	*10.08*	*1.14*

Also from above table, it can be observed that the execution time of the routing algorithms (both MPS and GAPA) comprised only a small part of the total time for the entire QoS adaptation process. For instance, MPS took approximately *8.5%* of the total calculation time for SLAOpt, and GAPA took about *12%* of the time required for R-SLAOpt.

10.3.3 QoS Adaptation with Different Group Number and Group Size

Figure 10.4 illustrates the simulation results for QoS adaptation in R-SLAOpt when the number of considered path groups G and the group size k were varied. Figures 10.4A and 10.4B show results for $k = 3$, while Figures 10.4C and 10.4D show results for $k = 2$.

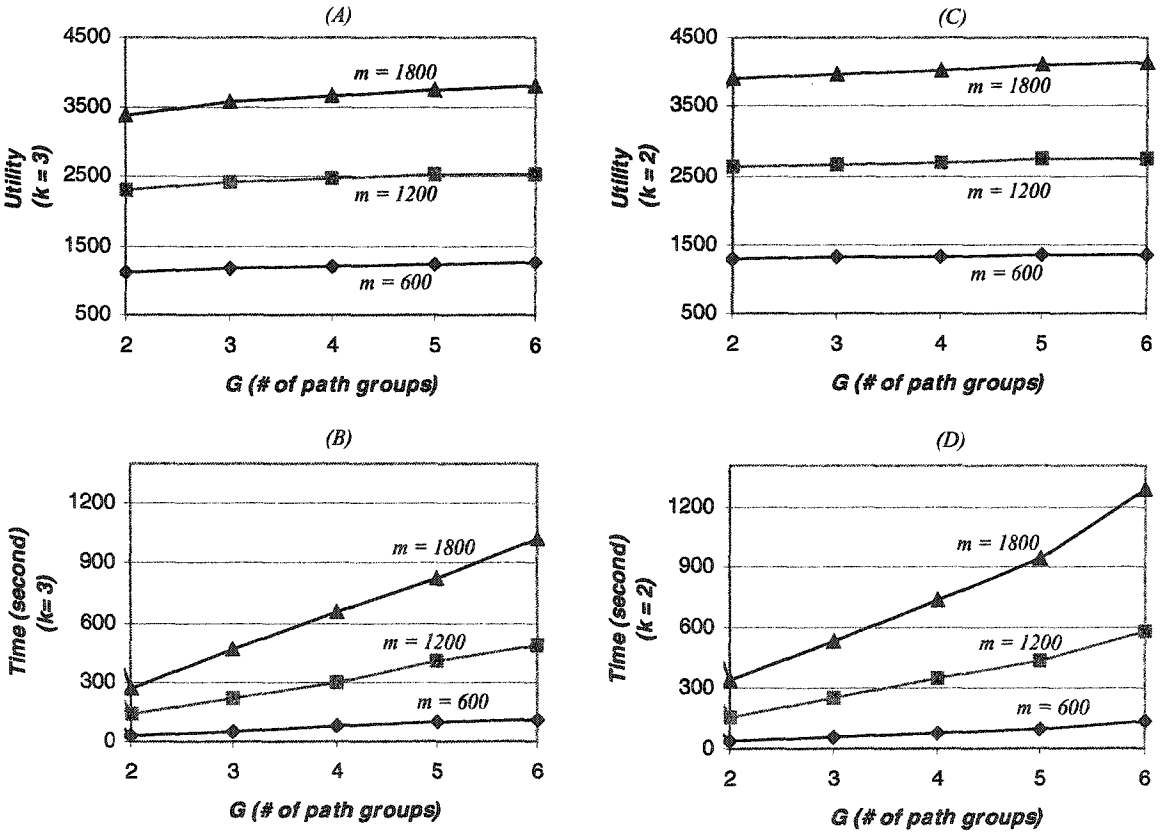


Figure 10.4. The utilities and time costs of R-SLAOpt in varying group number G and group size k.

It can be observed that the execution time and the utility obtained increased as the group number G increased. This is expected, because if there are more path groups as candidates for each SLA, the adaptation algorithm will need more time to choose the optimal group. More choices also increase the probability that higher utility may be obtained after adaptation.

When group size was increased (i.e., from $k = 2$ *to* $k = 3$), the obtained utility decreased up to *15%* while the execution time decreased up to *20%*. This is consistent with expectations, as for $k = 3$ resources must be reserved for an additional path in each selected path group. More resources will be consumed and fewer SLAs can be admitted, hence, fewer SLAs will be admitted and the system utility will decrease. The admission algorithm would require less time to select the optimal group, as it would likely have fewer choices when the group size is bigger.

Further simulations were performed to analyze the relationship between QoS adaptation and the group size k. With a fixed $G = 3$, QoS adaptation was run upon different k values. Figure 10.5 shows the simulation results. The utilities decreased up to *11%* when k increased from *1* to *2*, and decreased a further *13%* when k increased from *2* to *3*. Meanwhile, the execution time of the algorithm dropped *15%* and *20%*, respectively.

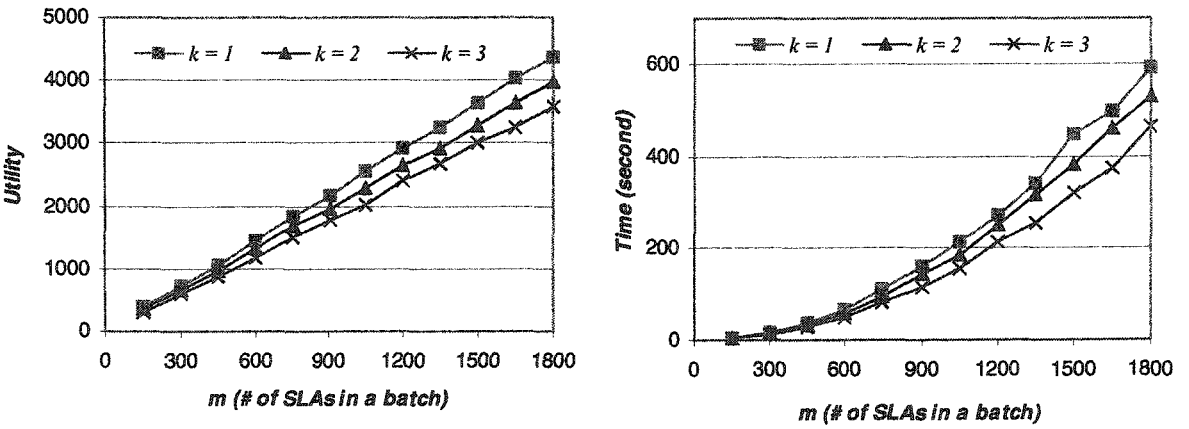


Figure 10.5. Impact on the utilities and time costs for different group sizes.

10.4 Appendix D, The MPS *K*-Shortest Paths Algorithm

Developed by Martins, Pascoal, and Santos, MPS [Mar99] is a well known *K*-shortest paths algorithm. Ease of implementation and absence of complex data structures are the main advantages of MPS. MPS uses a deviation approach to calculate paths in a directed (or undirected) graph. It calculates and ranks the *K* shortest deviation paths in for a given node pair *(s, t)* a graph *G(N, A)* based on the shortest path from *s* to *t*. MPS has time complexity *O(MlogN + KN)*. Pseudo code of MPS is shown in Figure 10.6.

// X – the set of candidates to the calculated shortest paths.
// T_t^* – the shortest path tree (SPT) rooted at *t*. Based on this SPT, let *D(v)* = the distance from *v* to *t*.
// $\overline{c}_{ij}$ – the detour length for arc *(i, j)*, which is $\overline{c}_{ij} = w(i, j) + D(j) - D(i)$.

```
Compute T_t^*
Compute c̄_ij, for any arc (i,j) ∈ A
Rearrange the set of arcs of (N, A) in the sorted forward star form
        /*          For the computed reduced costs          */
p_1 ← shortest path from s to t                  /*   p_1 is a path of T_t^*   */
k ← 1
X ← {p_k}
T_k ← {p_k}
While k < K and X ≠ ∅
do begin
        X ← X − {p_k}
        v_k ← deviation node of p_k
        for each node v ∈ p^k_{v_k t}
        do begin
                if A(v) − A_{T_k}(v) ≠ ∅
                then begin
                        (v, x) ← first arc in the set A(v) − A_{T_k}(v)
                        q ⟵ p^k_{sv} ◇ ⟨v, (v, x), x⟩ ◇ p^*_{xt}
                        X ⟵ X ∪ {q}
                        q_{vt} ⟵ ⟨v, (v, x), x⟩ ◇ p^*_{xt}
                        T_k ⟵ T_k ∪ {q_{vt}}
                           /*  In such a way that p^k_{sv} ◇ q_{vt} is a path of T_k  */
                end
        end
        k ← k + 1
        p_k ← shortest path in X
end
```

Figure 10.6. Pseudo code of the MPS K-shortest paths algorithm.