Improving Network Quality-of-Service with Unreserved Backup Paths

by

Ing-Wher Chen

A thesis submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

in

The Faculty of Graduate Studies

(Electrical and Computer Engineering)

The University of British Columbia
(Vancouver)

August 2008

Abstract

To be effective, applications such as streaming multimedia require both a more stable and more reliable service than the default best effort service from the underlying computer network. To guarantee steady data transmission despite the unpredictability of the network, a single reserved path for each traffic flow is used. However, a single dedicated path suffers from single link failures. To allow for continuous service inexpensively, unreserved backup paths are used in this thesis. While there are no wasted resources using unreserved backup paths, recovery from a failure may not be perfect. Thus, a goal for this approach is to design algorithms that compute backup paths to mask the failure for all traffic, and failing that, to maximize the number of flows that can be unaffected by the failure. Although algorithms are carefully designed with the goal to provide perfect recovery, when using only unreserved backup paths, re-routing of all affected flows, at the same service quality as before the failure, may not be possible under some conditions, particularly when the network was already fully loaded prior to the failure. Alternate strategies that trade off service quality for continuous traffic flow to minimize the effects of the failure on traffic should be considered. In addition, the actual backup path calculation can be problematic because finding backup paths that can provide good service often requires a large amount of information regarding the traffic present in the network, so much that the overhead can be prohibitive. Thus, algorithms are developed with trade-offs between good performance and communication overhead. In this thesis, a family of algorithms is designed such that as a whole, inexpensive, scalable, and effective performance can be obtained after a failure. Simulations are done to study the trade-offs between performance and scalability and between soft and hard service guarantees. Simulation results show that some algorithms in this thesis yield competitive or better performance even at lower overhead. The more reliable service provided by unreserved backup paths allows for better performance by current applications inexpensively, and provides the groundwork to expand the computer network for future services and applications.

Table of Contents

List of Tables

List of Figures

Acknowledgements

All my life, I have been blessed with great *teachers* who only wish for me to be the best that I can be, and I am grateful for all of them. For this project, I would most like to thank Dr. Mabo Robert Ito for his patience and guidance. I also give many thanks to Dr. Alan Wagner for his critical input, Dr. Norman Hutchinson for his feedback and for always lending an ear, and Dr. Panos Nasiopoulos for caring. I also thank Dr. Victor Leung and Dr. Kevin Qixiang Pang for their help with the OPNET simulator.

Words fail me when it comes to the three most important people in my life. So, I simply give a heartfelt, and tearful, thank-you to my greatest supporters, my parents, and to my grandmother, lovingly nicknamed Sushi by her grandchildren, for her immense pride in me. They are the wind beneath my wings and unequivocally the stars and heroes of my life. This thesis is a tribute to as well as the culmination of their unyielding support of me.

While chasing this dream, I have encountered many people who helped me stay sane and positive, from Dr. Pindy Badyal at the Thunderbird front desk to Jason Cameron at the Vancouver Rowing Club. I enjoyed the serendipitous meetings and am grateful for their help along the way. I especially thank Hanni Bagnordi, Yan Bai, Angela Foudray, Donna Ledbetter, Meera Srinivasan, Serene Wong, and my sister in particular, for their

pep talks, cajoling, encouragements, advice, and friendships. I could not have survived without them all.

I cannot possibly individually name the countless number of people who helped me just to reach the beginning of this journey, and so I am collectively thanking my friends and family from Taiwan. I am proud to have come from these people and this place, and am grateful for the lessons they taught me and the strength and value they instilled in me. Last but not least, I also thank the following people: Lesley Blimkie, Lechang Cheng, Jessica Chiang, Michael Connor, Hai-Yun Du, Jocelyn Gabua, Mulong Gao, James Herr, David Jordan, Salman Khan, Michelle Kittrell, Shin-Rong Li, Hsiu-Chih Lin, Lily Liu, Keith Marzullo, Barbara Murdoch, Nguyet Nguyen, Chhabilal Rizal, Sumiati Selamat, Chia-Yen Shih, Yi-Chin Tsai, David Vester, Marie Vester, and Ming-Hua Yang. Some of them are my playmates, some have greatly influenced my life, and some simply believed in me.

I thank all of the above, and many more, for standing by me until the bitter end.

To my mother, my hero.
She is the quintessential mother bear who gives the very best to her cubs.

To my father, our family's stoic supporter.

1 Introduction

1.1 Problem overview

The Internet was originally designed to interconnect different kinds of networks to allow machines on various networks to share information and resources [21]. A protocol to provide service for the lowest common denominator was designed. The end result is what is in place for the past decade: popularity of the Internet soared because of its accessibility and computer networks completely changed our ways of life. However, because of the decision to design a common layer to connect different computer networks rather than redesigning a new unified system, irregular performances from the network are expected, and this service characteristic is too unpredictable for newer applications.

The growth of the computer network has created many new applications requiring higher quality-of-service (QoS). Not only do data need to arrive at a destination, but they also need to arrive in a timely fashion. For critical streaming multimedia applications such as tele-medicine or sending continuous critical messages, it is imperative that a more reliable and more stable service is provided. The solution to guarantee steady data transmission rate despite the unpredictability of traffic conditions in the network is often provided by using dedicated resources [13] [14] [91]. A pre-determined amount of resources that can sustain a certain level of service quality are reserved on the nodes and links of a specific path, and data are forwarded only along this path. However, when service can only be provided through a specific path with reserved resources, it is natural that a single point of failure in this particular path will be hard to recover from without planning [9] [11] [64]. When a link failure occurs, the simplest solution that randomly chooses a disjoint path fails to consider that a link failure can cause several flows to fail at the same time. Consequently, without some coordination among these flows, there may be contention for the same resources which can drastically decrease service qualities. Even if the remaining network has enough resources to re-route all the affected traffic, it is still undesirable if it takes several tries to fit all the flows onto acceptable paths when delivery guarantees are needed in the first place. Thus, to improve QoS in the event of a single link failure, alternate backup paths need to be chosen such that they are coordinated and do not compete for the same resources.

1.2 Solutions

There are three major categories of solutions that are designed to specifically provide more reliable QoS. The first one involves the naïve solution that uses redundancy, which consumes a large amount of bandwidth. The second category involves choosing very specific backup paths to reserve so that they can be used in the event of a failure. These reserved backup paths are planned at the same time the primary paths are planned, to provide 100% guaranteed re-routing. Alternatively, an unreserved but well-chosen backup path can be planned in advance to improve the re-routing success rate. Sections 1.2.1, 1.2.2, and 1.2.3 briefly describe these three categories, and Table 1 summarizes the main differences. Further discussion of the research related to QoS is provided in Chapter 2.

1.2.1 Redundancy

A simple way to ensure that a message arrives at the destination despite a link failure is to send duplicates of the message out on different paths. This is extremely costly because it actively consumes network resources at all times. The challenge with this approach is to reduce the amount of redundancy, thereby reducing the bandwidth it consumes. Special message encoding analogous to RAID [67] can be used to reduce redundancy while retaining fault tolerance. The approach of dispersity routing [57], which is further discussed in Chapter 2, uses the redundancy idea but further breaks each message into several smaller messages to provide more reliable QoS while limiting the resource overhead [10] [34].

1.2.2 Two reserved paths

One alternative to the redundancy approach is to prepare a second reserved path disjoint from the first path (primary path) to provide backup in the event that the primary path fails. Unlike Section 1.2.1, before a failure occurs, messages are sent out only on the primary path. Although the resources on the backup path are reserved, regular best-effort traffic can still use these resources before the failure. However, the resources reserved on the backup paths cannot be used by traffic that requires high QoS, i.e., traffic that requires reserved primary paths, and thus the network's capacity to host high QoS traffic is reduced. Because backup paths are idle in most cases, not only because failures are chance happenings, but also because a failure does not affect all flows, the goal in this case is to minimize resources reserved purely for backup paths [32] [33] [51]. While the details of the designs and the techniques are provided in Chapter 2, an overview is provided here for comparison. The key to minimizing reserved backup path resources is analogous to overloading backup paths. One unit of reserved backup path resource is used by different backup paths at different times. One difficulty in minimizing resources reserved for backup paths is the amount of knowledge of the entire network that is required. The more information a router has regarding how the network and its resources are being used, the more the router is able to overload backup paths so that a smaller amount of backup path resources are reserved. Because communicating routing information among routers also consumes resources in the network, minimizing reserved backup resources should be carefully balanced with limiting communication overhead to a more reasonable size. In addition to wasting resources, another issue with the use of two reserved paths is that the optimal solutions often couple primary and backup paths tightly together during computation, resulting in a very specific set of paths that are effective only for a particular combination of flows [2] [46]. An additional flow in the network may require complete re-computation of paths to minimize the resources reserved for backups.

1.2.3 One reserved primary path and one pre-planned, unreserved backup path

Another approach to provide reliable QoS uses pre-planned but unreserved backup paths [48] [49]. Because the backup paths are unreserved, there is no wasted resources or decreased network utilization. The resources available on the planned but unreserved backup paths can be used to route best-effort traffic or reserved for high QoS traffic. However, also because the backup paths are unreserved, recovery from failure may not be perfect. Thus, the goal in this category is to maximize the number of flows that can be unaffected by the failure and to minimize the effects of the failure on the actual traffic. A perfect algorithm would compute backup paths for flows such that, even without reserving resources in advance, all the flows with a failed primary path can be re-routed over the pre-planned backup paths and experience the same service quality as before. One problem when unreserved backup paths are used is that this perfect recovery for all flows may not be possible under some conditions, particularly when the network was already fully loaded prior to the failure. In contrast to Section 1.2.2, an alternative to limit the degradation of service experienced by the flows after a failure is needed. In addition, the same challenge of balancing communication overhead with performance that is described in Section 1.2.2 also applies to this approach. Finding backup paths that can provide good service after a failure requires a lot of information regarding the flows present in the network, but the overhead involved with communicating such information should not be so large that it consumes a significant amount of network resources. Thus, similar to the approach of reserving two paths, trade-offs also need to be made between good performance and communication overhead. Table 1 provides a summary of the three major categories discussed.

Table 1. A summary of the three categories of solutions, their weaknesses and the issues different algorithms within each category try to address.

Solution category	**Main weakness**	**Issues that different algorithms try to address**
Redundancy (Section 1.2.1)	Consumes available resources at all times.	• Reduction of redundancy.
Two reserved paths (Section 1.2.2)	Consumes resources that can be used by other QoS traffic.	• Reduction of the amount of reserved resources on backup paths. • Reduction of communication overhead. • Improvement of computation complexity.
One reserved primary path and one pre-planned, unreserved backup path (Section 1.2.3)	Less than 100% guaranteed recovery from failure.	• Improvement of recovery performance. • Reduction of communication overhead. • Provision of different techniques for different network conditions.

1.3 Objectives and approach

To minimize the cost of providing reliable, high service qualities, the approach taken in this thesis is that of one reserved primary path paired with a carefully chosen pre-planned but unreserved backup path, as described in Section 1.2.3. Because the backup paths are intended to be unreserved prior to the failure, they are not expected to provide complete recovery from all kinds of failures under all conditions. A family of techniques are developed so that as a whole, the use of unreserved backup paths can provide competitive and acceptable performance with low overhead and cost to the users and the service providers. Various algorithms are developed to calculate these unreserved backup paths. The first algorithm, AvoidPBO, is designed with the aim of yielding good performance after the failure regardless of the amount of overhead incurred. It also sets up a basic structure that has the potential to be improved to reduce overhead. The next algorithms, TP and TPmax, are designed to improve on the overhead required by AvoidPBO so that only constant overhead in a fixed network is incurred. Different techniques are also developed to make the best use of these unreserved backup paths after a failure. The traditional strict recovery sacrifices connections for service quality after a failure, whereas the relaxed recovery trades off service quality to maintain connection for all traffic even after failure. Due to the complexity of the system, the evaluation of these methods is done using simulation modeling. Because the variance and behavior of the different methods compared in this thesis are of great interest, simulations are used to obtain data at the desired granularity. Experiments are performed on generalized graphs to gain understanding of the capability of unreserved backup paths and their general behaviors and performance limitations.

1.4 Thesis outline

Chapter 2 discusses in more detail the major categories of solutions to provide reliable QoS and how such a service can be provided in practice. Research studies on QoS in the past and in the current trend are also discussed. Chapter 3 describes the basic framework of the solution to provide QoS with added reliability. Discussions regarding the network environment for which this work is intended and the design and principles of the backup path computation algorithms are provided. The first backup path computation algorithm, AvoidPBO, is discussed, along with the rationale and design decisions that continue to be used in later algorithms. Based on the same principles employed by AvoidPBO, Chapter 4 discusses techniques to reduce the overhead requirement and presents the designs of two other algorithms that incur considerably less overhead while retaining the same high level of performance. Chapter 5 discusses additional methods that can be paired with unreserved backup paths to improve the performance of the unreserved backup paths. Chapter 6 provides simulation results of the methods designed in Chapters 3, 4, and 5, and comparisons to existing methods are also provided. Finally, Chapter 7 provides the concluding thoughts and suggestions for future work.

2 Related Work

2.1 Chapter outline

There are four parts to this chapter. An important part of choosing a path is in the techniques and algorithms used to decide to choose a certain link over other links. Thus, the first part of this chapter, Sections 2.2 and 2.3, describes the fundamental techniques used to grade and represent the desirability of a link. The second part examines traditional QoS routing (Section 2.4), and Sections 2.5, 2.6, 2.7, and 2.8 describe the different types of solutions that can be used to tolerate single component failures in addition to the traditional QoS routing. Section 2.9 provides a comparison of different solutions and an outline of the approach taken in this thesis. The third part of the chapter, Section 2.10, describes real-life network protocols that can support or be modified to support a reliable QoS system. Finally, Section 2.11 discusses some research trends in QoS routing. Section 2.12 provides a summary of this chapter.

2.2 The importance of link weights

Open shortest path first (OSPF) [59] is a fundamental and popular link state routing protocol [58] used within an autonomous system (AS). With OSPF, network topology (link states) is known to all routers in the network. In its basic version, each link is statically assigned a weight, which indicates the cost of using the link. By adding up the weight of links in a path, shortest paths to all destinations are computed at each router separately. A routing table is constructed by storing pairs of destination and corresponding next hop information extracted from the shortest paths. When a packet arrives at a router, the router looks up its routing table for the next hop based on the destination of the packet; the packet is then placed on the interface to the next hop for forwarding. Because the lengths of the shortest paths correlate to the link weights, the next hop or the path over which a data packet traverses is dependent not only on how the links are connected, but also on the link weights assigned to those links. Thus, link weights can be seen as a representation of the desirability of the links in the network as well as be used to control routing. A change in the link weights could cause a change of forwarding path and a shift of the dynamics of the traffic being routed over a network. This observation that link weight can be optimizated to adjust network performance is discussed in Section 2.3.

2.3 Link weight optimization

In OSPF, the default link weights are pre-selected, often set to unit weight or to weight proportional to the inverse of link capacity. If the link weights stay fixed, some cheaper links may be heavily used, causing congestion on those links, while links on other feasible paths are sitting idle, resulting in lower overall network performance. To adapt to different traffic loads and dynamic traffic conditions in the network, link weights should be changed based on link loads to move some of the traffic off heavily loaded links [47]. Link weights can also be specifically designed for known traffic demands to minimize delay [17] and to minimize the maximum utilization on links [27] [28], which in turn minimizes delay. Link weights can further be manipulated to improve the throughput, delay, and load-balancing efforts of the network [31].

While more sophisticated link weights can provide good results, the destination-based hop-by-hop routing approach taken by OSPF still prevents traffic from being forwarded on all available paths, using all available resources, because traffic to a destination is routed only along the shortest paths. Because a small change in link weight can shift more traffic than is intended, congestion at one location may be resolved at the expense of new congestion occurring at a new spot. Thus, destination-based hop-by-hop routing like OSPF does not take full advantage of link weight optimization. In addition, while optimizing link weights can improve average network performance, it is still not enough for newer applications, such as real-time traffic, that require stricter QoS requirements.

2.4 Quality-of-Service (QoS)

Newer applications, such as real-time applications, generally require strict service guarantees from the network for specific traffic flows. Using destination-based hop-by-hop routing, a general solution to provide QoS is to find, for each destination, a path with the highest performance, such as highest bandwidth and shortest delay, in the hope that it can satisfy most, if not all, requests [86]. This can be done with only a few minor changes during route computation in OSPF. However, to provide more specific guarantees while using destination-based hop-by-hop routing, the routing table stored in each router needs to expand to include a next hop for each QoS level, for each destination [5]. Another modification that can be made in conjunction with link weight optimization is to service higher priority packets first [53], instead of servicing packets on a first come first served basis. However, because each hop in the path from source to destination has its own traffic load and conditions, simply regulating packets at each node may not result in stable transmission at the end hosts. Thus, while destination-based hop-by-hop routing can be extended to provide general QoS or improve average service quality without too many modifications, it is still not ideal for more stringent or finer-grained QoS requirements.

To achieve the desired end-to-end QoS, every node in the path from the source to destination needs to cooperate and offer a certain level of service [13] [14] [20] [41] [60] [61] [90] [91] [92]. When a request for connection is made, before the start of a traffic flow, the path from source to destination is determined, and the resources needed to achieve an end-to-end guarantee are reserved on each component in the path. If such a path cannot be found, the request for connection at this particular service level is rejected. Depending on the configuration, either another connection request with reduced service level is submitted or the traffic flow does not get sent over the network.

Using reserved paths to route high QoS traffic as described above is a large research area [8]. Some variations of this framework include the routers' ability to dynamically manage the level of resource reservation to increase the call acceptance rate and network utilization [66] [77]. Because reserving resources is a big task and commitment by the network, one of the major concerns involves resource management to improve network utilization. As network conditions change, switching to a different path during the connection may improve the call admission rate [29] [43] or may lower delay [24] because resources elsewhere have become available and more suitable for certain traffic demands over time. Switching between multiple paths based on network conditions can also achieve better load balancing [25] [54] [81].

More recently, due to the movement of applications from more traditional networks, such as the telephone network, to the computer network, concern for reliability has become of interest. In addition to becoming less desirable over time due to changes in the network, resources on a dedicated path may simply fail, making the original path useless and forcing a switch of paths [40] [79]. In a centralized network like the telephone network, the system can be designed for a specific QoS with certain call acceptance rates and fault tolerance [2] [46] [52] [94]. This is not the case with computer networks. Without advance planning for fault tolerance, recovery at the time of path failure may be unsuccessful or may take too much time [9]. Below, Sections 2.5, 2.6, 2.7, and 2.8 describe different solutions that deliver reliability to improve QoS routing. Table 2 summarizes the different types of solutions and Section 2.9 provides the justification for the approach taken in this thesis.

2.5 Using two reserved paths

In addition to reserving a primary path for packet forwarding prior to a failure, a second, backup path can also be reserved prior to a failure to provide more reliable QoS. Sections 2.5.1 and 2.5.2 describe two types of algorithms to compute these backup paths that are intended to be reserved.

2.5.1 End-to-end protection

One of the simplest solutions to protect a path against a single component failure is to have a backup path (Figure 1). When a request for connection is made, two link-and-node disjoint paths are computed, and the necessary resources are reserved on both paths. Traffic is initially sent over the primary path. If a failure occurs in the primary path, the source switches over to use the backup path [40] [79]. A problem with this approach is the inability to find two node-and-link disjoint feasible paths. In this case, solutions with maximally disjoint paths are proposed [23] [54]. Another problem concerns the trade-off between algorithm efficiency and solution optimality. For instance, it is difficult to efficiently calculate the absolute shortest pairs of paths, which are expected to be the least costly to reserve [63]. In general, the biggest weakness of this approach is that using two reserved paths requires large amounts of resources to be reserved for one single connection. Thus, a large portion of the research is on how to reduce the amount of resources reserved, particularly how to reduce the amount of resources reserved on the backup paths.

To reduce the resources reserved for backup paths, backup resource multiplexing is used. If a link $\ell(i, j)$ is used to back up both primary paths P_1 and P_2 for connections C_1 and C_2 respectively, and if P_1 and P_2 share some of their links or nodes, then resources for both C_1 and C_2 need to be reserved on $\ell(i, j)$. On the other hand, if P_1 and P_2 do not share any components, then the two connections can share resources on their backup links when only one link is expected to fail [50] [55] [49]. Further refinement can be made so that users can choose the level of multiplexing they want, and hence the actual level of fault tolerance [36] [38] [39] [80]. Another approach to reduce resources reserved for backup paths is to consider an area of the network. It is unlikely that a component of every primary path in that area will fail. Thus, although a link $\ell(i, j)$ may be used to backup n connections, the resources reserved on $\ell(i, j)$ can be less than the resources needed for all *n* connections, because realistically, from the perspective of the network as a whole, or even just a portion of the network, not all of those primary paths are expected to fail [69].

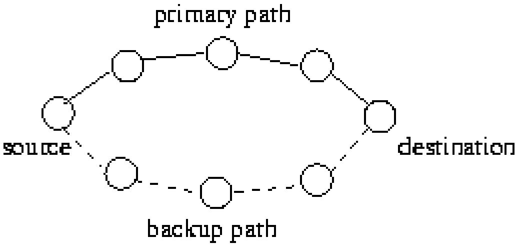


Figure 1. End-to-end primary and backup paths.

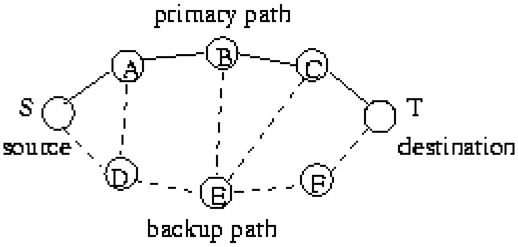


Figure 2. Local protection.

2.5.2 Local protection

Instead of using a disjoint path to protect the primary path, another approach is to protect each component or each segment of the primary path [48] [51] [69] [93]. Figure 2 depicts a primary path from source *S* to destination *T* in solid lines, and links used to protect the primary path in dashed lines. If link *ℓ(C, T)* fails and node *C* detects it, then node *C* can immediately switch to the backup links used to protect *ℓ(C, T)*, links *ℓ(C, E)*, *ℓ(E, F)*, and *ℓ(F, T)*. Compared to the approach in Section 2.5.2, the computation of this approach is much more complex, because the primary path and the “bridges” are not independent of each other. Because it may not always be possible to find bridges to protect each element in the primary path, different algorithms propose that the bridges be computed in segments [12] [32] [33] [74]. Instead of having one backup path, or bridge, to backup each link/node, each bridge is used to backup a segment of the path. Figure 3 shows an example of how backup paths, represented by the dashed lines, are set up to protect different segments in the primary path, which is depicted in solid lines.

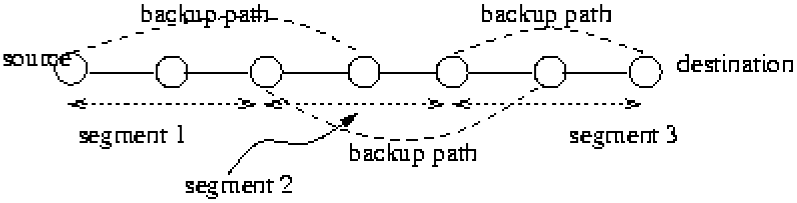


Figure 3. Local protection in segments.

2.5.3 Advantages and disadvantages

Regardless of the type of protection, the main advantage of reserving resources on the backup paths is that perfect recovery is guaranteed. A traffic flow with a failed primary path experiences the same service quality before and after the failure. However, there are two major disadvantages. The obvious one is the amount of resources that could be wasted when there is no failure. While the resources reserved on the backup paths can still be used to route best-effort traffic, they can not be used to support QoS traffic. The other somewhat hidden disadvantage is the amount of routing information required to compute the backup paths. Algorithms that compute backup paths with the goal of minimizing the amount of resources reserved on backup paths often require detailed information regarding how traffic in the network is routed, which actively consumes network bandwidth. Thus, using two reserved paths, perfect recovery can be achieved, but it comes at the expense of substantial decrease in network utilization.

2.6 One reserved primary path and one pre-planned unreserved backup path

Similar to Section 2.5, this approach also computes backup paths to overcome the failure of the corresponding primary paths. Unlike Section 2.5, the backup paths are not reserved prior to the failure when their corresponding primary paths are alive, and resource reservation to provide strict QoS is attempted on the backup paths only when they are needed. To improve success rate, backup paths are chosen so that they have the highest chance of successfully reserving enough resources to provide strict QoS when they are needed [48] [49]. Because backup paths are unreserved when their primary paths are active and alive, there are no wasted resources. However, also because the backup paths are unreserved initially, there is a chance that resources may not be available for reservation at the time of failure. Thus, the goal in this category is to maximize the number backup paths that can successfully reserve the required resources to provide strict QoS at the time of failure. Similar to the previous approach, improving the success rate of backup path reservation requires a lot of information on the flows present in the network. Thus, algorithms in this approach also need to carefully manage the balance between high backup path success rate and communication overhead.

2.7 On-demand path computation

Another approach similar to Section 2.6 is to calculate backup paths, for the first time, at the time a failure is detected [9] [11] [64] [85]. Re-routing of traffic is attempted on those backup paths [56] [65]. Like the approach in Section 2.6, this also maintains high network utilization because only the primary paths are reserved and no extra resources are reserved. However, because there is no advance planning to coordinate all the traffic flows like the approach described in Section 2.6, the coordination of traffic flows is done after the failure. One way to prevent traffic with failed primary paths from contending for the same resources is by sequentially re-routing traffic, which may take too much time for the re-routed traffic to be useful. Additionally, some traffic may require several re-routing attempts before a successful path is found, and this is expected to consume network bandwidth. Thus, with this approach, it has similar disadvantages as the approach in Section 2.6, but is not expected to perform as well. While this approach could be the default solution when the approach in Section 2.6 fails, it should not be the primary solution to improve QoS.

2.8 Dispersity routing

The idea of dispersity routing [57] is to break a message into N sub-messages, and to send each sub-message along a different path. Thus, only $\frac{1}{N}$ of the total resource required by the connection needs to be reserved on each path, and one failed path does not impact the other paths. If occasional loss of packets can be tolerated, as is the case in most streaming multimedia, this approach offers fault tolerance without the need to reserve backup resources. There are many variations of dispersity routing, from different ways to break up a message [34] to incorporating redundant information into the multiple segments of each message [10]. To incorporate redundant information, the same number of N paths is computed. In this case, a message is broken up into K sub-messages, where

$K<N$. Among the N paths, K of them are used to send actual data messages, and the remaining N-K paths are used for error correction code. Lost data can then be recovered using the redundant information [35]. The price to pay for this added fault tolerance is the complexity at end hosts for coding and decoding data, and $\frac{N}{K}$ times more resource needs to be reserved. To avoid coding and decoding, a straightforward solution is to send multiple copies of each complete message over disjoint paths [73]. However, this means that if N copies were sent, then N times the resources would be needed. By combining dispersity routing with backup paths [71], more failure models, such as double link failures, can be tolerated. The disadvantages of such combinations are the complexity, the feasibility of deployment, and the lower network utilization. Multiple disjoint paths may not always be found and extra resources are needed to route extra packets.

2.9 Comparison and the chosen approach

Table 2 summarizes the different levels of services as well as the advantages and disadvantages of the different approaches. As mentioned in Chapter 1, the approach taken in this thesis is to pre-plan unreserved backup paths. This approach is chosen because it is relatively inexpensive, compared to the use of two reserved paths or dispersity routing. Compared to on-demand routing, which does not pre-reserve or pre-plan backup paths, simply pre-planning backup paths eliminates the weakness of long disconnection. In the following chapters, several algorithms and techniques are developed to overcome the other weaknesses involved with the use of pre-planned but unreserved backup paths. Different techniques are designed and employed to improve the service experienced by traffic flows after a failure as well as to reduce the amount of routing information communicated over the network.

2.10 Network protocols

The algorithms in Sections 2.4, 2.5, 2.6, 2.7, and 2.8 require some support in the underlying system to provide additional network metrics that allow better paths to be computed. Many algorithms also require an extra step to establish a connection to use a specific, dedicated path. Sections 2.10.1 and 2.10.2 describe two protocols, OSPF [59] and Multiprotocol Label Switching (MPSL) [76], and their extensions, that are widely available and can be structured and modified to support some of the algorithms above. Section 2.10.3 describes how the algorithms designed in this thesis can be deployed using existing network protocols.

Table 2. Comparison of various levels of QoS and different approaches.

Level of QoS	Approach	Pros	Cons
Best-effort	Current default Internet service (Best-effort service)	Reliable (Recovers from component failures.)	Unpredictable delivery.
Timely delivery	One reserved path	Stable and predictable delivery when designated components are alive.	1. Potential waste of resources. 2. Suffers from single component failures. 3. Suffers from difficult deployment because it requires upgrades on all (or most) nodes in the network. 4. Suffers from scalability problems because of the computational complexity at core nodes.
Timely and reliable delivery	Two reserved paths (Section 2.5)	Not only stable and predictable service, but also reliable delivery.	5. Wasted resources on the backup paths are always expected. 6. Some algorithms suffer from high computational complexity. 7. Requires large amounts of routing and network information. • Suffers from 3 and 4 above.
	One reserved primary path and one pre-planned unreserved backup path (Section 2.6)	No extra wasted resources on backup path.	• Suffers from 3 and 4 above. • Some traffic suffers from 2 above. • Potentially suffers from 7 above.
	On-demand backup path computation (Section 2.7)	No extra wasted resources on backup path.	• Suffers from 3 and 4 above. • Some traffic suffers from 2 above. • Potentially suffers from 7 above. • Suffers from potentially long disconnection even for successfully re-routed traffic.
	Dispersity routing (Section 2.8)	• No extra computation at core nodes. • Can recover from losses and delays, and also potentially recover from transmission errors.	• Suffers from 3 above. • Extra computation at end nodes. • Some waste of resources. • Need disjoint paths, which is not always possible.

2.10.1 OSPF

In OSPF, when a router is initiated, it discovers its neighbors and their capabilities regarding extensions [22]. At the end of this stage, bidirectional communication is established between neighbors, and neighbors will know when to declare a failed link and whether particular routing information is to be exchanged. This stage is followed by synchronization of their views of the network, the link state databases. At the end of this procedure, all routers construct one common view of the network topology. Using Dijkstra's shortest path algorithm, this results in identical routing tables in all routers, even though they are constructed by different routers.

While OSPF is traditionally used in best-effort routing, it has many extensions that can be used to provide QoS. In the original OSPF specification, each link advertised in a link state advertisement (LSA) has a link cost statically assigned by network operators. As mentioned in Section 2.3, by monitoring link usage, routers can convey link conditions by dynamically assigning link costs based on available bandwidth, and encoding these costs in LSAs without changing the LSA format. If routers agree that link cost is a function of unallocated bandwidth, the amount of available bandwidth can be re-constructed at destination routers, and paths satisfying different bandwidth requirements can be easily computed [5]. Given a set of known demands, link weights can also be calculated so that simple Dijkstra's shortest path algorithm used in OSPF can construct routing tables to balance loads and accept more connections, compared to some commonly suggested link weight schemes such as unit weight or inverse capacity as weight [27]. Furthermore, instead of exchanging LSAs at a fixed interval, LSA updates can be configured to be triggered only when there are significant changes to the network, where significant change is a pre-determined threshold or a certain percentage of difference [4] [93].

Alternatively, opaque-LSA [22] provides a generalized mechanism to communicate extra information to routers. It has a regular LSA header and is flooded reliably, similar to other LSAs. The body of an opaque-LSA can contain any information to be used by routers, or even applications. For example, opaque-LSAs can be used to distribute extra link information, such as maximum bandwidth and unreserved bandwidth on a link, to facilitate QoS routing [45].

2.10.2 Multiprotocol Label Switching (MPLS)

Traffic engineering and QoS requirements are instrumental in the creation of MPLS to forward packets along very specific paths. MPLS places an extra mapping layer, the label, between destination IPs and the next hops. A label is a short, fixed length identifier that is associated with a forwarding equivalence class (FEC) between a pair of downstream/upstream routers. All packets in an FEC travel on the same path once they traverse a common router. In destination-based hop-by-hop routing, a destination prefix could be seen as being a unique FEC. Assuming that destination IP prefix x is the FEC f. The downstream router, R_d, binds label ℓ to FEC f, which is associated with a destination IP prefix, x, that has a next hop IP address of nh. This binding is reliably communicated, requiring acknowledgement, from the downstream router R_d to the upstream router R_u, either periodically or by request. When R_u wants a packet forwarded to the destination IP prefix x, R_u assigns the packet label ℓ, as is told by R_d. R_u then sends it to the next hop router, R_d. When R_d receives this packet, it looks up the FEC based on the label ℓ in the packet header. It finds that label ℓ identifies FEC *f*, which translates to destination IP prefix *x*, which in turn has the next hop *nh*. This lookup also returns the label that the next router, the one that is downstream from R_d, has communicated to R_d for forwarding packets in the FEC. R_d replaces the old label with this new one and places the packet onto the outgoing interface for *nh*.

Using a label to map to next hops provides the flexibility required for explicit path routing, which is used extensively in QoS routing. With MPLS, next hops are no longer exclusively associated with destination IP prefixes. A particular path can be set up by assigning that path to its own FEC and relaying this information to all the hops along the path using a signaling protocol, such as RSVP with additional objects to express explicit routes [7] [91] or the label distribution protocol (LDP) designed specifically for MPLS [3] [42]. Each router along the path associates a label with the next hop in the path and stores this entry in its routing table. Once the label switched path is set up, packets can be sent along this path simply by labeling them with the specific label chosen earlier. There is no need to list the entire path in every packet header, as is done in IP source routing.

2.10.3 Combining OSPF and MPLS

MPLS is often used as the underlying switching technology in QoS routing because of its flexibility and control in setting up explicit paths and different service levels used to satisfy certain constraints [8] [25] [30] [83] [88]. Combining the ability to forward packets on explicit paths using MPLS and the ability to reliably and efficiently collect link traffic conditions as well as network topology using OSPF, servers have been designed to provide QoS guarantees [6] [70].

2.11 Trends

More recently, the research trend is in pushing QoS service over multiple domains, for both traditional QoS constrained services and reliable QoS constrained services. The current network architecture has difficulties providing such services because topology and routing information is limited across domains for scalability and security issues. However, topology and routing information is vital for QoS routing. Consequently, there is renewed interest in the topic of topology aggregation [84], to provide useful and scalable topological information for QoS routing across multiple domains. Additional network components [26] are also proposed to handle inter-domain QoS routing [75] [89]. Finally, a few efficient algorithms are proposed to provide reliable QoS routing across multiple domains [82]. These algorithms currently compute the traditional two reserved paths as described in Section 2.5, where the backup paths are intended to be reserved for the duration of the connections and perfect recovery is provided in the event of a link failure.

2.12 Chapter summary

There are many techniques that are used to improve QoS in computer networks, which were first designed for best-effort service. These techniques range from link weight management for average improvement to reserving resources for strict service agreements. Existing protocols have been used to construct systems for QoS routing based on these techniques. The following chapters specifically look at modifications that can be made to improve QoS, such that recovery from a single link failure is both possible and satisfactory. As mentioned in Section 2.9, the central idea is to pre-plan backup paths prior to the failure so that the backup paths do not contend for the same resources when they need to be activated. Chapters 3, 4, and 5 discuss the algorithms used to plan these backup paths as well as different ways to use these backup paths to provide good service to the end users.

3 Basic System and Design

3.1 Chapter outline

This chapter describes the basic framework and principles used in this thesis to provide added reliability to QoS. Sections 3.2 and 3.3 describe the network environment and assumptions in which the proposed algorithms are deployed. Section 3.4 provides the rationale and design decisions of the first backup path computation algorithm, AvoidPBO (Avoid Conflicting Primary and Backup Paths with Order Consensus). It shows the strategy employed by AvoidPBO to compute effective backup paths, which has rules to avoid conflicts during calculation and ultimately results in backup paths that do not contend for the same resources if possible. Sections 3.5, 3.6, 3.7, 3.8, and 3.9 provide the details of AvoidPBO, including the information required to compute effective backup paths and steps taken to compute the backup paths. A simple test in Section 3.10 is provided to show that, compared to disjoint paths, AvoidPBO spreads out the use of links so that its backup paths can potentially provide better service than disjoint paths in the event of a single link failure. Finally, Section 3.11 provides a summary of this chapter.

3.2 Assumptions

In this thesis, it is assumed that a solution to provide more reliability requires at least one reserved primary path. Because a second reserved path is too costly, the proposed solution uses an unreserved path as backup. Thus, each flow has a reserved primary path which it uses if there is no failure in the network, and it also has an unreserved backup path for the duration of its connection. The goal is to find a backup path to tolerate a single link failure as much as possible, either by fully recovering from a single link failure so that users do not detect any difference, or if such perfect recovery is not possible, by downgrading the service quality as little as possible.

Traffic in the network is assumed to change over time. It is assumed that more accurate information regarding the forwarding paths taken by such traffic will produce better pre-planned backup paths. The only way to capture the state of the traffic in a volatile network is to perform periodic information updates. Because traffic in the network changes constantly, information regarding the location or placement of traffic also changes constantly. Since information regarding these changes takes time to propagate through the network, previously selected backup paths may not remain the best option over time and should be updated periodically. Thus, even though there is always an unreserved backup path associated with a flow, the backup path may change due to updated information.

Overall, similar to OSPF and Internet best-effort routing, routing information is circulated throughout the network periodically. This additional routing information changes as the traffic conditions in the network change. The algorithms proposed below compute the backup paths based on this routing information. This entire process is repeated periodically.

3.3 Network environment

While the emphasis in the following sections and chapters is on the backup path computation methods and how they differ from each other, these algorithms can be deployed in a network setup similar to those described in Section 2.10.3. The algorithms should be deployed within an autonomous system (AS) that does not maliciously use the advertised path information. The AS likely uses a link-state routing algorithm [58] such as OSPF [59] that can perform the necessary authentication as well as extensions [22] to broadcast extra path information. Because data is now forwarded on very specific paths, the paths will have to be set up beforehand using a signaling protocol such as RSVP [91]. The actual forwarding of data packets along a specific path can be supported by technology such as MPLS [76]. OSPF is widely supported, and MPLS is gradually being accepted and built into routers, particularly for routers providing high QoS.

3.4 Avoid conflicting primary and backup paths with order consensus (AvoidPBO)

AvoidPBO [16] [17] is a heuristic to calculate a second path that remains unreserved but can still provide good service in the event that the corresponding reserved primary path has failed due to a single link failure. In this setup, the primary path of a flow is assumed to be reserved and does not change for the duration of the connection. In contrast, the backup path is unreserved to conserve resources in the network. Because the backup path is unreserved, it needs to be carefully chosen so that it can be useful and still offer good service when it is activated. Without careful planning, a single link failure that causes multiple flows to fail might result in the failed flows all trying to use the same links to re-route their traffic. For example, in Figure 4, it is likely that when *link(a, b)* fails, the flows that used *link(a, b)* to forward traffic previously all choose to use *link(a, c)* and *link(c, b)* to re-route traffic. This is acceptable if there are unlimited resources on those links. However, when free resources are limited or even scarce before the link failure, some planning and coordination among the re-routed flows is required to allow all of those flows to receive acceptable service after the failure.

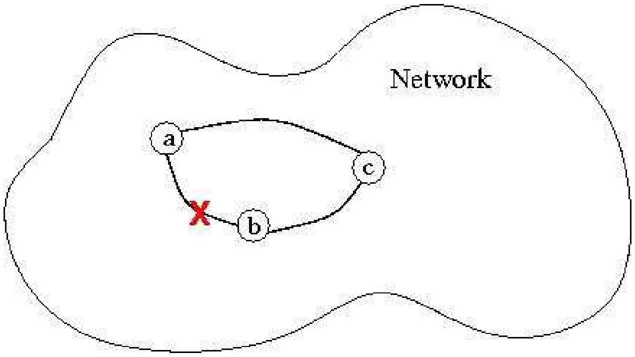


Figure 4. An example of a single link failure.

The basic technique is to first collect various routing information regarding the traffic in the network. Using the additional routing information, an effective backup path can be computed by first determining the links that are undesirable and then adding extra weight to those links to discourage the use of those links. AvoidPBO separates undesirable links into two categories, the links used by conflicting primary paths and those used by conflicting backup paths. In the end, when appropriate link weights are assigned, AvoidPBO invokes Dijkstra's single source shortest path algorithm (Dijkstra's algorithm) to compute the actual backup path.

In Section 3.5, the additional routing information collected and required by AvoidPBO is described. Section 3.6 describes the two types of conflicts in AvoidPBO and the steps taken to compute a backup path using those conflicts. Section 3.7 further discusses the difference between avoiding conflicting primary paths and avoiding conflicting backup paths that is briefly mentioned in Section 3.6. Section 3.8 describes the actual formulae used to calculate extra link weights that are employed in the experiments in Section 3.10 and Chapter 6. Section 3.9 discusses further adjustments that can be made to the basic AvoidPBO design to accommodate different bandwidth requirements. While more comprehensive evaluation is provided in Chapter 6, Section 3.10 provides a simple test to compare the effectiveness of AvoidPBO in terms of choosing different links as backup paths.

3.5 Additional information

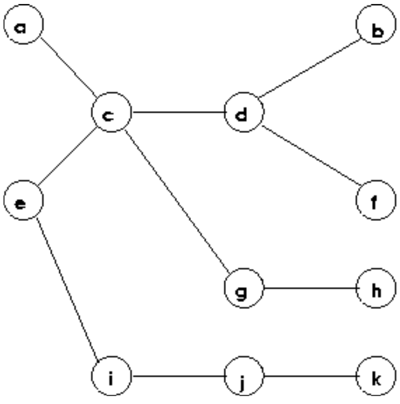
Because the goal is to use unreserved backups to tolerate a failure in the network, backup paths should be carefully chosen. Logically, a backup that is most likely to be successfully activated when the time comes should be maximally disjoint from its corresponding primary path. To fully utilize all the available resources, the backup should share as few links as possible with other paths that might be active at the same time. With AvoidPBO, to determine the paths that might be active at the same time, each source node periodically sends out the primary and backup path information of the flows it initiates and maintains. This primary-backup path information is sent to all other source nodes in the network. Section 3.6 describes how a source node uses such information to help compute backup paths for its flows.

Like other algorithms that also require per-flow information to be communicated over the network, AvoidPBO is also extremely expensive. The overhead required by AvoidPBO is ultimately bounded by the number of flows or the amount of traffic in the network, which can increase indefinitely. However, by pushing computation to source nodes, AvoidPBO sets up a structure that is conducive to other algorithm designs that do not require the distribution of per-flow routing information. A method to condense the routing information and thereby reduce this communication overhead is provided in Chapter 4.

3.6 Avoiding conflicts

As mentioned in Section 3.2, the desired solution to improve QoS is to have a reserved primary path working in cooperation with an unreserved backup path for each flow. However, without reserving resources on the backup paths, a backup path may not be adequate to provide reliable and high quality backup when the primary path fails. Even though a backup path that is disjoint from the primary path does not fail when a single link failure causes the primary to fail, this failure can cause several primary paths to fail. This in turn causes their corresponding backup paths to be activated simultaneously, which can result in contention of resources among the activated backups. Figure 5 illustrates an example in which two flows can fail simultaneously when *link(e, c)* fails. When *link(e, c)* fails, the primary paths of flow 2 and flow 3 will fail, causing the backup paths of flow 2 and flow 3 to be activated simultaneously. Because flow 2 and flow 3's backup paths may be activated simultaneously, these backup paths may conflict with each other and take resources away from each other. Thus, the two backups should be coordinated in such a way that increases both their chances of successful resource reservation.

In addition to avoiding conflicting backup paths, conflicting primary paths should also be avoided. For example, for flow 4, in the event of a single link failure at *link(i, j)*, causing flow 4 to fail but leaving flow 2 and flow 3 to continue to use their respective primary paths, the backup path of flow 4 may conflict with the primary paths of flow 2 and flow 3. In fact, there is no point in re-routing flow 4 on *link(e, c)* if all the resources on *link(e, c)* are reserved for flow 2 and flow 3's primary paths. So, avoiding conflicting primary paths is particularly important when the network is heavily loaded, because even after a single link failure in the network, the majority of the traffic is likely to remain on their original, reserved primary paths. If links are full with primary path traffic, then there will be no free resources left for the backup paths.



flow	primary path
flow 1	a, c, d, b
flow 2	e, c, d, f
flow 3	e, c, g, h
flow 4	e, i, j, k

Figure 5. Examples to illustrate reasons to avoid links.

Altogether, the steps to calculate a backup path are as follows. For a flow f_x with primary path P_x, to compute the corresponding backup path, B_x, the source node goes through each primary-backup path pair it has stored. For each flow f_i with primary-backup pair P_i-B_i, the source node first calculates the number of shared links between P_x and P_i. The source node then uses the steps in Sections 3.6.1, 3.6.2, and 3.6.3 to determine the amount of link weight to add. In general, undesirable links end up with heavier link weights, and vice versa. The actual backup path is computed using Dijkstra's algorithm. The pseudo code is provided in Figure 6.

3.6.1 Step 1

If there are no shared links between P_x and P_i, extra weight can be added to links on P_i to prevent the eventual B_x from conflicting with P_i. This is needed because if P_x and P_i do not share any links, P_x may fail while P_i does not, when there is a single link failure. Consequently, B_x and P_i may be active at the same time. Thus, for B_x to be successful, it should avoid links used by P_i, especially if all those links are already full. To avoid links on P_i, these links should be weighted and their new link weights can be calculated by adding extra weight proportional to the path length of P_i.

3.6.2 Step 2

If P_x and P_i share at least one link, then B_x may potentially conflict with B_i when one of the shared links fails. Thus, either B_x should avoid B_i or vice versa. One solution is to give the flow with the earlier starting time priority, and let the lower priority flow avoid the links used by the higher priority flow. Thus, if the ordering of flows is based on starting time and if the starting time of flow f_x is later than that of flow f_i, then the link weight for links used by B_i should be updated to reflect their undesirable status. The new link weights can be calculated by adding an extra weight proportional to the number of shared links.

3.6.3 Step 3

Finally, enough extra weight also needs to be added to the links on P_X to prevent B_X from using the same links as P_X. The amount of extra weight added in this case should be more than the total amount added in the first two cases because having a backup disjoint from the primary path is still the most important element in recovering from a failed primary path.

Finally, based on the adjusted link weights, Dijkstra's shortest path algorithm is used to compute the backup path P_X.

```
computeBackupAvoidPBO (flow $f_x$) {
      W: weight matrix.
      $W_k$: weight on link $\ell_k$.
      $f_i$: flow i.
      $P_i$: primary path of $f_i$.
      $B_i$: corresponding backup path of $P_i$.
      $P_x$: primary path of $f_x$.
      src: source of $f_x$.
      dest: destination of $f_x$.

      for i = 1 .. number of primary-backup path pairs
            stored
            if $P_x$ and $P_i$ do not share any link then
                  add extra weight to links on $P_i$ in W
            if $P_x$ and $P_i$ share at least a link and $f_i$ has higher
                  priority over $f_x$ then
                  add extra weight to links on $B_i$ in W
      end
      add weight $w_{primary}$, where $w_{primary}$ >> $W_k$, to each link $\ell_m$ in $P_x$
      use Dijkstra's algorithm and the new weight matrix W
            to compute a shortest path between src and dest.
}
```

Figure 6. The pseudo code for AvoidPBO.

3.7 *Ordering of flows*

3.7.1 Ordering backup paths

As mentioned in Section 3.6.2, there is an ordering issue between backup paths that should avoid each other. At the start of designing for AvoidPBO, two side effects of adding weight to links before invoking Dijkstra's algorithm were observed. One side effect is that adding too much weight to links often results in avoiding and discounting too many links and options, which has a detrimental effect on the overall result. The second side effect is that when two backup paths actively try to avoid each other, a common third alternative is often chosen for both backup paths. To avoid this problem and to limit the weight added to links, AvoidPBO requires only one backup path to actively avoid the other backup path. The backup path that should actively avoid another backup adds extra weight to links used by the other backup path. To achieve consistency, there should be a consensus on which backup path should avoid the other one. As mentioned in Section 3.6.2, a simple way to establish this consensus is to use the starting time of each flow and assume that a flow with an earlier starting time has priority over a flow with a later starting time. Assume flow f_u and flow f_v have starting times of t_u and t_v, where $t_u < t_v$. When the backup path of flow f_u, B_u, is computed, if it is determined that B_u and B_v should avoid each other according to Section 3.6.2, then because $t_u < t_v$, B_u does not have to actively avoid B_v, and no extra weight is added to the links used by B_v when computing B_u. When computing B_v, based on the ordering rule, B_v needs to actively avoid B_u, and so extra weight is added to the links used by B_u. This is the scheme used in Section 3.10 and Chapter 6.

3.7.2 Not ordering primary paths

When it comes to avoiding conflicting primary paths, AvoidPBO does not use the ordering of flows to determine the path to avoid, because conflicting primary paths should always be avoided. The assumption is that resources reserved and occupied by primary paths will always belong to the primary paths unless they fail. Since a conflicting primary path means the primary path has not failed at the time a backup path is activated, those resources reserved by that particular primary path are assumed to be unavailable to the backup path.

3.8 Weights in experiments

This section describes the weight formulae implemented in the experiments in Section 3.10 and Chapter 6. The notations used in Sections 3.8.1, 3.8.2, and 3.8.3 are defined and described in Table 3. A summary of the conditions under which a link is considered a conflicting link is shown in Table 4. The actual extra weights calculated by Sections 3.6.1, 3.6.2, and 3.6.3 are determined by the formulae described in Sections 3.8.1, 3.8.2, and 3.8.3 respectively. These formulae, which are implemented for the experiments in Section 3.10 and Chapter 6, are summarized in Section 3.8.4 and Table 5.

3.8.1 Extra weight on a conflicting primary path

Following the example described in Section 3.6.1, when computing the backup path B_x, AvoidPBO first determines whether the primary path, P_i, of another flow can fail at the same time as P_x. This is determined by calculating the number of shared links, $numberOfSharedLinks(P_i, P_x)$ in Table 3, between P_i and P_x. If $numberOfSharedLinks(P_i, P_x) = 0$, then P_i and P_x will not fail simultaneously.

Consequently, when P_x fails and B_x is activated, P_i is still active. In other words, B_x conflicts with P_i, and extra weight should be added to links used by P_i, as shown in Table 4. In the tests in Section 3.10 and Chapter 6, the actual weight added to each link in P_i is proportional to the path length of P_i, $pathLen(P_i)$. This weight is the result of $pathLen(P_i)$ factored by the value, *factor*, which is pre-defined for the duration of each simulation run. For example, assume that path P_i consists of links ℓ_1, ℓ_2, and ℓ_3. Then, the path length of P_i is 3, and the weight added to each of ℓ_1, ℓ_2, and ℓ_3 is $3 * factor$. When $factor = 1$, the weight added to each of ℓ_1, ℓ_2, and ℓ_3 is 3, for this particular case of conflict. By including the path length of the conflicting primary path (P_i) when computing weight, alternate paths, even though they are longer than P_i are encouraged. To sum up, when computing B_x, if it is decided that extra weight needs to be added to the links on P_i, then the added weight is $numberOfConflicts(f_i, f_x) * factor$, where $numberOfConflicts(f_i, f_x) = pathLen(P_i)$. During debugging, there was generally no noticeable difference in the performance and quality of backup paths using different values for *factor*. Thus, $factor = 5$ is used for the experiments in Sections 3.10 and 4.8.2 and Chapter 6. Further sensitivity tests are left for future work (Section 7.3).

3.8.2 Extra weight on a conflicting higher priority backup path

In the case described in Section 3.6.2, the number of shared links between P_i and P_x is not zero: $numberOfSharedLinks(P_i, P_x) > 0$ in Table 4. If B_i has higher priority than B_x, then extra weight is added to each link of B_i, and the extra weight added is $numberOfConflicts(f_i, f_x) * factor$, where

$numberOfConflicts(f_i, f_x) = numberOfSharedLinks(P_i, P_x)$. The rationale for this is that the more shared links P_i and P_x have, the more likely it is for P_i and P_x to fail simultaneously due to a single link failure. In other words, the larger $numberOfSharedLinks(P_i, P_x)$ is, the more likely that B_i and B_x will be activated simultaneously. So, it is reasonable that backup paths and their corresponding links that are more likely to be conflicted should have heavier extra weights added.

In general, the extra weight added to the links on conflicting backup paths is smaller than the extra weight added to the links on conflicting primary paths because $numberOfSharedLinks(P_i, P_x) \leq \min\{pathLen(P_i), pathLen(P_x)\} \leq pathLen(P_i)$. Because primary path resources are never available unless the primary path has failed, conflicts with primary paths is more severe, and thus the need to avoid them is higher.

3.8.3 Extra weight on the corresponding primary path P_x

Finally, in the following simulations, the weight added for each link described in Section 3.6.3 is at least the total weight added to a link in Sections 3.6.1 and 3.6.2. Any extra weight added to a link described in Sections 3.8.1 and 3.8.2 is also added to the links used by the primary path, P_x. This is because a link used by P_x should be the least desirable link, and thus should carry the maximum weight that can possibly be added to other links in Sections 3.8.1 and 3.8.2. Additionally, $|N| * factor$ is also added to each link used by P_x, where N is the set of nodes in the network, as defined in Table 3. The additional weight, $|N| * factor$, is added to encourage the use of another path, or group of links, in place of any link in P_x, even if that path consists of all the nodes in the network, as shown in Figure 7.

3.8.4 Summary of AvoidPBO implemented in experiments

To compute the backup path B_x of the flow f_x, which has an existing, reserved primary path of P_x, the source node compares P_x to the routing information of all known flows f_i. Link weights are modified using the formulae described in Sections 3.8.1, 3.8.2, and 3.8.3, which are summarized in Table 5. Dijkstra's algorithm is then invoked on the graph with the modified link weights. The shortest path computed by Dijkstra's algorithm is the designated backup path B_x.

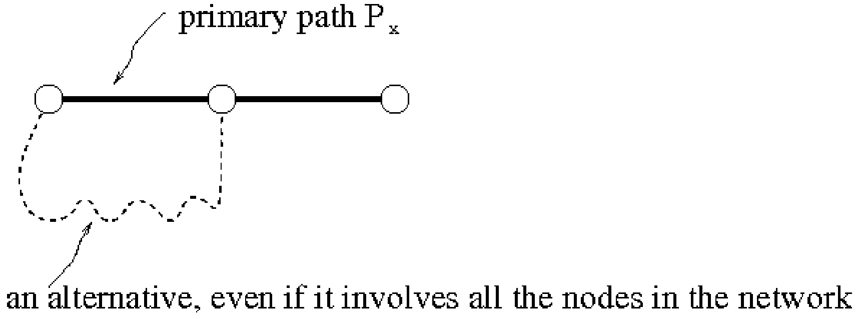

primary path P_x
an alternative, even if it involves all the nodes in the network

Figure 7. Extra weight added to links used by the primary path.

Table 3. Definitions of variables.

Variable	Definition
$\mathcal{N}$	The set of nodes in the network.
$\mathcal{L}$	The set of links in the network.
P_v	The primary path of the flow f_v, which is defined as a set of links $\{\ell_a, \ell_b, \ldots, \ell_n\}$, where each link $\ell_i \in \mathcal{L}$.
B_v	The backup path of the flow f_v, which is defined as a set of links $\{\ell_a, \ell_b, \ldots, \ell_n\}$, where each link $\ell_i \in \mathcal{L}$.
factor	A pre-defined value selected by the source node.
$numberOfSharedLinks(P_i, P_j)$	The number of common links between two paths, P_i and P_j.
$numberOfConflicts(f_i, f_j)$	$\begin{cases} numberOfSharedLinks(P_i, P_j), & \text{if } numberOfSharedLinks(P_i, P_j) > 0. \\ pathLen(P_i), & \text{if } numberOfSharedLinks(P_i, P_j) = 0. \end{cases}$
$pew_{\ell_k}^{f_i}$	The extra weight added to the link $\ell_k \in P_i$ based on comparison between f_i and f_x, which is equal to $\begin{cases} numberOfConflicts(f_i, f_x) * factor, & \text{if } numberOfSharedLinks(P_i, P_x) = 0. \\ 0, & \text{if } numberOfSharedLinks(P_i, P_x) > 0. \end{cases}$
$bew_{\ell_k}^{f_i}$	The extra weight added to the link $\ell_k \in B_i$ based on comparison between f_i and f_x, which is equal to $\begin{cases} 0, & \text{if } numberOfSharedLinks(P_i, P_x) = 0. \\ numberOfConflicts(f_i, f_x) * factor, & \text{if } numberOfSharedLinks(P_i, P_x) > 0. \end{cases}$
w_{ℓ_k}	Weight on link ℓ_k.

Table 4. Conflicting links based on comparison between P_x and P_i of f_x and f_i respectively.

Conflicting link	Condition
Conflicting primary path link ℓ_k	If $numberOfSharedLinks(P_i, P_x) = 0$ and $\ell_k \in P_i$.
Conflicting backup path link ℓ_k	If $numberOfSharedLinks(P_i, P_x) > 0$, $\ell_k \in B_i$, and priority of f_i is higher than f_x.

Table 5. Extra weight added in the steps described in Sections 3.8.1, 3.8.2, and 3.8.3.

Step	**Action taken**	**Link weight modification**
Formula 1 (Section 3.8.1)	After f_x is compared to f_i, if $pew_{\ell_k}^{f_i} > 0$	$w_{\ell_k} = w_{\ell_k} + pew_{\ell_k}^{f_i}, \forall \ell_k \in \mathrm{P_i}$
Formula 2 (Section 3.8.2)	After f_x is compared to f_i, if $bew_{\ell_k}^{f_i} > 0$	$w_{\ell_k} = w_{\ell_k} + bew_{\ell_k}^{f_i}, \forall \ell_k \in \mathrm{B_i}$
Formula 3 (Section 3.8.3)	After completion of f_x vs. f_i for all known flows f_i	$w_{\ell_k} = w_{\ell_k} + \sum_{f_i \neq f_x} pew_{\ell_k}^{f_i} + \sum_{\substack{f_i \neq f_x \\ f_i \text{ has higher priority over } f_x}} bew_{\ell_k}^{f_i} + \lvert \mathbf{N} \rvert * factor, \forall \ell_k \in \mathrm{P_x}.$

3.9 Bandwidth variation of flows

In Section 3.6, the emphasis is on showing the different types of links that should be avoided when choosing a backup path. While AvoidPBO assumes that each flow requires the same bandwidth to satisfy their QoS requirements, it can also be extended for the case where each flow in a network has a different bandwidth requirement. In this case, the additional information communicated out into the network, as described in Section 3.5, will have to include the information regarding bandwidth requirements. Furthermore, the extra weight added to a conflicting link also has to be weighted based on the bandwidth requirement. For example, assume that an extra weight of w is added to link ℓ to avoid this link, which is used by the flow f_1 with b units of bandwidth requirement. If link ℓ is also used by another flow f_2 with $k*b$ units of bandwidth requirement, then an extra weight of $k*w$ should be added to link ℓ to avoid this particular link. This extension is left for future work (Section 7.3).

3.10 Test for ability to not reuse links

As mentioned in Section 3.4, without coordinating backup path computation, it is possible that all the flows affected by a single link failure will choose the same links to use to re-route their traffic (Figure 4). This tends to reduce the service quality, especially when the network was already full of traffic prior to the failure. Thus, when computing a backup path using AvoidPBO, links that might be in use after a failure are avoided if possible, in the hope that the end result is a backup path with free resources available at the time it is needed. The test in this section calculates the average number of backup paths supported by a link when there is a link failure in the network. In this experiment, the ideal case is that, when a backup path is needed, each link in the backup path only re-routes traffic for this single backup path and no other backup path. While this is not always possible in many network and traffic configurations, carefully choosing backup paths, as is done with AvoidPBO, is expected to reduce the number of flows that use the same link. For this test, a randomly generated graph of 20 nodes, each with a connectivity of four, as represented by the adjacency matrix in Appendix A, is used. Each link is assumed to have 20 units of resources, with the ability to carry 20 flows. A total of 600 flows are tested in this network. Each flow is assumed to require or consume one unit of resource on each link of its forwarding path. These flows and the primary paths of these flows are not randomly chosen. Instead, 100 distinct source-destination pairs are first generated: $\{<src_1, dest_1>, <src_2, dest_2>, \ldots, <src_{100}, dest_{100}>\}$. The primary paths are mapped out for these flows and they are computed using Dijkstra's algorithm with uniform link weight of one. Thus, a group of flows, $\{f_1, f_2, \ldots, f_{100}\}$, with $\{<src_1, dest_1>, <src_2, dest_2>, \ldots, <src_{100}, dest_{100}>\}$ are created, and each flow has a corresponding primary path, $\{P_1, P_2, \ldots, P_{100}\}$. Then, the rest of the 500 flows are randomly chosen from this pool of sources and destinations, and they reuse the same pool of primary paths. For example, the 101st flow, f_{101}, could be a flow from src_i to $dest_i$, where $< src_i , dest_i > \in \{< src_1 , dest_1 > , < src_2 , dest_2 > , \cdots , < src_{100} , dest_{100} >\}$, and its primary path is P_i, where $P_i \in \{P_1 , P_2 , \cdots , P_{100}\}$. The only rule is that each link in the network carries at most 20 flows, due to the resource limit set for this network. This traffic setup is designed to create and increase conflicts among flows to force AvoidPBO to actually work and circumvent congestion and other traffic. If none of the flows has a primary path that conflicts with other primary paths, as depicted in Figure 8, then it is likely very simple to choose backup paths that do not conflict with each other and still result in a low average backup path per link. By forcing some flows to conflict with each other, the selection and conflict avoiding ability of AvoidPBO can be examined. While all flows are assumed to start at the beginning of the simulation, the actual starting time of each flow, $t_1, t_2, \ldots, t_{600}$, is assumed to be such that $t_1 < t_2 < \ldots < t_{600}$.

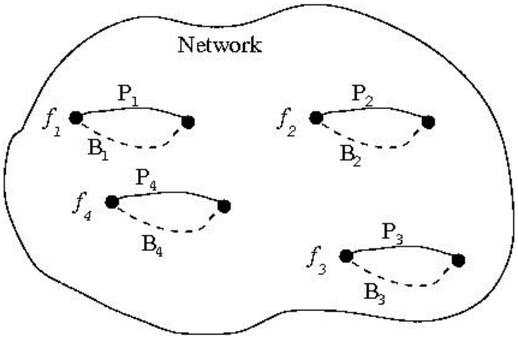


Figure 8. An example with no conflicting primary paths.

Using the setup above, in the beginning, each link services at most 20 flows. The backup paths are then computed in two rounds using the three steps described in Section 3.6. In the first round, only the primary and backup path information of flows originating from the same source node is used. This is to simulate the fact that it takes time to propagate routing information to all the nodes in the network. In the second round of backup path computation, all the routing information of all flows is available to all nodes. The backup paths computed in this step take advantage of the routing information of all the traffic in the network.

Once the primary and backup path of each flow are mapped out, each case of single link failure in the network is tested. A total of 80 failure cases are tested. For each single link failure case that causes at least one backup path to be activated due to at least one failed primary path, the following metrics are calculated. The average number of backup paths on each link, $ABPL_{failCase_i}$, is computed as follows:

$$ABPL_{failCase_i} = \frac{\sum_{B_j} pathLength(B_j)}{NDL}$$, where B_j is a backup path that is activated in $failCase_i$ and NDL is the total number of the distinct links used by the backup paths activated in $failCase_i$.

Then, the value reported in Figure 9, *ratio*, is computed as follows:

$$ratio = \frac{\sum_i ABPL_{failCase_i}}{NFC}$$, where NFC is the number of failure cases considered.

A *ratio* of one indicates that on average, when a link is used after a link failure caused one or more primary paths to fail, that link is used by one re-routed flow. Thus, a high *ratio* indicates high contention for a link. Figure 9 shows the *ratio* for six cases, each with 100, 200, 300, 400, 500, and 600 flows in the network, as described above.

For comparison, Figure 9 includes the *ratio* calculated for disjoint backup paths. Disjoint backup paths are computed by first taking out the links used by their respective primary paths, and then by invoking Dijkstra’s algorithm on a graph with unit link weights on each remaining link. As mentioned, the minimum *ratio* is one, which is also included in Figure 9. Figure 9 shows that there is clearly more contention at each link when disjoint paths are used, whereas AvoidPBO is able to spread out the links used by backup paths. While further evaluation of AvoidPBO is provided in Chapter 6, these preliminary test results show that the links that AvoidPBO chooses to avoid are reasonable and yield better backup paths than simply computing disjoint paths.

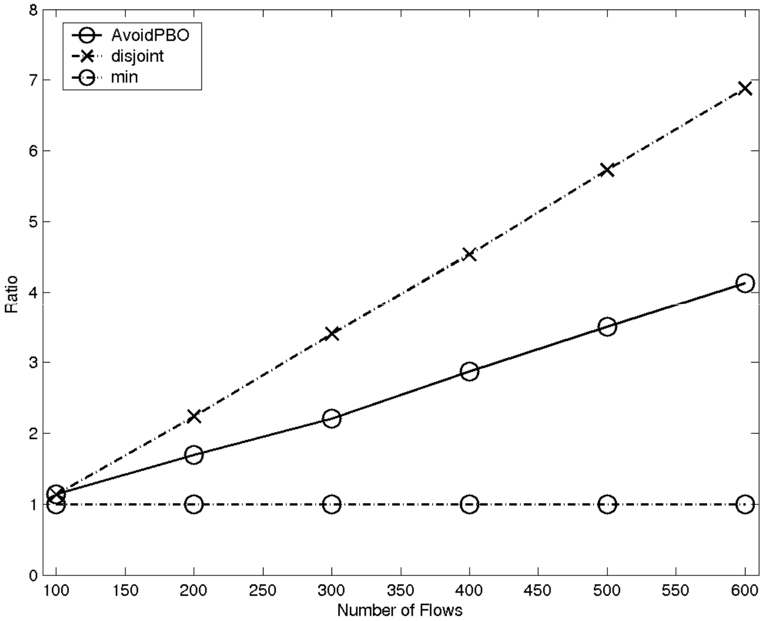


Figure 9. Comparison of the ratio of total number of links used to the number of distinct links used.

3.11 Chapter summary

This chapter describes the foundation of a system that can provide more reliable QoS service specifically in the event of a single link failure. Such a system has been shown to be built by combining existing popular network protocols such as OSPF and MPLS. Using AvoidPBO, backup paths can be chosen to complement the fixed and reserved primary paths. When backup paths are activated due to a single link failure, AvoidPBO backup paths are more spread out than disjoint backup paths, which bode well for the flows routed on these backup paths in terms of QoS. One glaring drawback with AvoidPBO is its overhead, which is examined and improved in Chapter 4.

4 Overhead Reduction

4.1 Chapter outline

While the method (AvoidPBO) in Chapter 3 showed evidence of reducing resource contention (Figure 9), the need to broadcast the primary and backup path information of every traffic flow (Section 3.5) is a big drawback because it is cost prohibitive. With AvoidPBO, in an effort to reduce congestion, potential congestion is also introduced. In addition, although the computation of backup paths can be performed in the background on a secondary processor, the need to look at routing information of every single flow in the network to compute the backup paths can be time consuming. Section 4.2 describes how to condense the raw routing information used by AvoidPBO to a level that is fixed regardless of the amount of traffic in the network. Although the routing information is condensed, Section 4.3 shows that it is condensed in such a way that it still contains usable and helpful information to compute effective backup paths. Two methods, TP [15] and TPmax [18] [19], which use the condensed routing information to compute backup paths in a manner similar to AvoidPBO are provided in Sections 4.4 and 4.5, and Section 4.6 discusses some potential future extensions. Sections 4.7 and 4.8 discuss the differences among AvoidPBO, TP, and TPmax, and Section 4.9 provides a summary of this chapter.

4.2 Condensed routing information

As alluded to in Chapter 3, because AvoidPBO broadcasts path information of all flows, the communication cost incurred by AvoidPBO can be prohibitive. The main goal of this chapter is to reduce this communication overhead while retaining competitive performance. To retain competitive performance, the condensed routing information needs to contain similar critical information used by AvoidPBO, mainly the information that allows an algorithm to determine the resources to avoid in each specific situation.

4.2.1 *PTP* and *BTP*

Assume that L is the number of links in the network. The primary and backup paths stored at each source node are first condensed into two arrays of size $O(L^2)$ bytes: the primary traffic placement array (*PTP*) and the backup traffic placement array (*BTP*). A source node N_i is required to store the primary and backup paths of its own flows, because it is responsible for computing the primary and backup paths and setting up the data forwarding paths for the flows it originates. Based on the primary and backup path information of its flows, N_i constructs PTP^{N_i} and BTP^{N_i}.

4.2.2 Construction and rationale

The principle employed by AvoidPBO is to first determine the links that may be used at the same time by higher priority traffic, either the primary path traffic or higher-ranked backup path traffic, and then manipulate the link weights so that lower-ranked traffic tries to avoid those links. In other words, it is important to know which links are used simultaneously under what circumstances. Thus, instead of sending out complete path information, the source node can condense the path information such that it still contains information about the links that are used simultaneously and the circumstances under which this occurs. For example, among the flows depicted in Figure 10 and summarized in Table 6, for flow f_1 originating at node N_1 with a pair of paths, primary path P_1 using links $\{\ell_1, \ell_2, \ell_3\}$ and backup path B_1 using links $\{\ell_4, \ell_5, \ell_6, \ell_7\}$, this can be aggregated and presented in the form of the usage of links when one of the links, ℓ_1, ℓ_2, ℓ_3, ℓ_4, ℓ_5, ℓ_6, or ℓ_7, fails. In this case, when ℓ_1 fails, ℓ_4, ℓ_5, ℓ_6, and ℓ_7 will all carry flow f_1. The same goes if the link failure occurs at ℓ_2 or ℓ_3. On the other hand, when any other link fails, for example when ℓ_4 fails, ℓ_1, ℓ_2, and ℓ_3 will carry the flow. Thus, information regarding f_1, P_1, and B_1 can be condensed into "which link carries the traffic when a particular link fails":

when ℓ_1 fails, ℓ_4, ℓ_5, ℓ_6, ℓ_7 carry the traffic;

when ℓ_2 fails, ℓ_4, ℓ_5, ℓ_6, ℓ_7 carry the traffic;

when ℓ_3 fails, ℓ_4, ℓ_5, ℓ_6, ℓ_7 carry the traffic;

when ℓ_4 fails, ℓ_1, ℓ_2, ℓ_3 carry the traffic;

when ℓ_5 fails, ℓ_1, ℓ_2, ℓ_3 carry the traffic;

when ℓ_6 fails, ℓ_1, ℓ_2, ℓ_3 carry the traffic;

when ℓ_7 fails, ℓ_1, ℓ_2, ℓ_3 carry the traffic.

Instead of sending out raw path information of all its flows, a node can condense the path information into a traffic placement array, *TP*, of size L^2. In this case, $TP_{i,j}$ indicates the number of flows using link ℓ_j when link ℓ_i fails. *TP* can further be divided into two parts, *PTP* and *BTP*, where *PTP* is the traffic placement of primary paths and *BTP* is the traffic placement of backup paths. Using the example above, PTP^{N_1} stores the following information:

when ℓ_4 fails, ℓ_1, ℓ_2, ℓ_3 carry primary path traffic;

when ℓ_5 fails, ℓ_1, ℓ_2, ℓ_3 carry primary path traffic;

when ℓ_6 fails, ℓ_1, ℓ_2, ℓ_3 carry primary path traffic;

when ℓ_7 fails, ℓ_1, ℓ_2, ℓ_3 carry primary path traffic.

BTP^{N_1} stores the following information:

when ℓ_1 fails, ℓ_4, ℓ_5, ℓ_6, ℓ_7 carry backup path traffic;

when ℓ_2 fails, ℓ_4, ℓ_5, ℓ_6, ℓ_7 carry backup path traffic;

when ℓ_3 fails, ℓ_4, ℓ_5, ℓ_6, ℓ_7 carry backup path traffic.

To sum up, at each node N_i, primary and backup path information is condensed into PTP^{N_i} and BTP^{N_i}. Assume that the flows originating at N_i are categorized as the following:

$\mathcal{P}_k$ = the set of flows whose primary paths use link ℓ_k,

$\mathcal{B}_j$ = the set of flows whose backup paths use link ℓ_j,

$\mathcal{AP}_{j,k}$, defined as the set of flows that continue to use their primary paths when ℓ_j fails and these primary paths all use ℓ_k,

= $\{f \mid f \notin \mathcal{P}_j \wedge f \in \mathcal{P}_k\}$, and

$\mathcal{AB}_{j,k}$, defined as the set of flows that have a failed primary path when ℓ_j fails and their corresponding backup paths all use ℓ_k,

$= \{f \mid f \in \mathcal{P}_j \wedge f \in \mathcal{B}_k\}$.

Then, the values condensed by N_i are

$PTP^{N_i}_{j,k}$, defined as the number of flows that use their primary paths when ℓ_j fails and these primary paths all use ℓ_k,

$= \left| \mathcal{AP}_{j,k} \right|$, and

$BTP^{N_i}_{j,k}$, defined as the number of flows that use their backup paths when ℓ_j fails and these backup paths all use ℓ_k,

$= \left| \mathcal{AB}_{j,k} \right|$.

The definitions above are also summarized in Table 7 and Table 8.

Using the earlier example where N_1 originates a single flow using P_1 and B_1, $PTP^{N_1}_{j,k}$ should be zero for all links ℓ_k when ℓ_1 fails, because there is no other primary path traffic when ℓ_1 fails. $PTP^{N_1}_{j,1}$, $PTP^{N_1}_{j,2}$, and $PTP^{N_1}_{j,3}$ for all links ℓ_j except ℓ_1, ℓ_2, and ℓ_3, should all store one, because if any link other than ℓ_1, ℓ_2, or ℓ_3 fails, P_1 is used. Conversely, $BTP^{N_1}_{1,4}$, $BTP^{N_1}_{1,5}$, $BTP^{N_1}_{1,6}$, and $BTP^{N_1}_{1,7}$ should all have the value of one, because when ℓ_1 fails, there is backup path traffic on ℓ_4, ℓ_5, ℓ_6, and ℓ_7. Table 9 provides a summary of the elements in PTP^{N_1} and BTP^{N_1} based on the traffic that originates from N_1, which is described in Table 6.

Overall, based on the primary and backup path information stored at N_i, N_i condenses per-flow information into link-based information that stores the usage of network resources in the event that a link ℓ_j fails, as summarized in Table 8. Because the two arrays actually communicated into the network are of fixed size regardless of the amount of traffic in the network, the use of *PTP* and *BTP* reduces the communication cost to a more reasonable constant size in a fixed network.

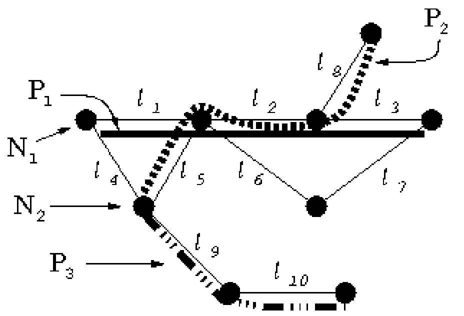


Figure 10. A diagram of the sample flows summarized in Table 6.

Table 6. Sample flows.

Source node order	Source node	Flow order	Flow	Primary path	Backup path
high→low	N_1	high→low	f_1	P_1: $\{\ell_1, \ell_2, \ell_3\}$	B_1: $\{\ell_4, \ell_5, \ell_6, \ell_7\}$
	N_2		f_2	P_2: $\{\ell_5, \ell_2, \ell_8\}$	B_2: Compared to f_1, should avoid ℓ_4, ℓ_5, ℓ_6, and ℓ_7.
	N_2		f_3	P_3: $\{\ell_9, \ell_{10}\}$	B_3: • Compared to f_1, should avoid ℓ_1, ℓ_2, and ℓ_3. • Compared to f_2, should avoid ℓ_5, ℓ_2, and ℓ_8.

Table 7. Groupings of primary and backup paths of flows originated at N_i.

Set	Definition	Value
$\mathcal{P}_j$	The set of flows whose primary paths use link l_j.	
$\mathcal{B}_j$	The set of flows whose backup paths use link l_j.	
$\mathcal{AP}_{j,k}$	The set of flows that continue to use their primary paths when l_j fails and these primary paths all use l_k.	$\{f \mid f \notin \mathcal{P}_j \wedge f \in \mathcal{P}_k\}$
$\mathcal{AB}_{j,k}$	The set of flows that has a failed primary path when l_j fails and their corresponding backup paths all use l_k.	$\{f \mid f \in \mathcal{P}_j \wedge f \in \mathcal{B}_k\}$

Table 8. Definition of each element in PTP^{N_i} and BTP^{N_i}.

Data	Definition	Value
$PTP_{j,k}^{N_i}$	The number of flows originated at N_i that use their primary paths when ℓ_j fails and these primary paths all use ℓ_k.	$\lvert \mathcal{AP}_{j,k} \rvert$
$BTP_{j,k}^{N_i}$	The number of flows originated at N_i that use their backup paths when ℓ_j fails and these backup paths all use ℓ_k.	$\lvert \mathcal{AB}_{j,k} \rvert$

Table 9. PTP^{N_1} **and** BTP^{N_1} **based on Table 6.**

Array	Values in each element of PTP^{N_1} and BTP^{N_1}
PTP^{N_1}	$PTP^{N_1}_{j,k}$ = 1, where $\ell_j \in \{\mathbf{L} - \ell_1 - \ell_2 - \ell_3\}$ and $\ell_k \in \{\ell_1, \ell_2, \ell_3\}$.
	$PTP^{N_1}_{j,k}$ = 0, where $\ell_j \in \{\ell_1, \ell_2, \ell_3\}$ and $\ell_k \in \mathbf{L}$.
BTP^{N_1}	$BTP^{N_1}_{j,k}$ = 1, where $\ell_j \in \{\ell_1, \ell_2, \ell_3\}$ and $\ell_k \in \{\ell_4, \ell_5, \ell_6, \ell_7\}$.
	$BTP^{N_1}_{j,k}$ = 0, where $\ell_j \in \{\mathbf{L} - \ell_1 - \ell_2 - \ell_3\}$ and $\ell_k \in \mathbf{L}$.
Assumption: $\mathbf{L}$ is the set of links in the network.	

4.3 Determining conflicts using *PTP* and *BTP*

The algorithms discussed in this chapter, TP (Section 4.4) and TPmax (Section 4.5), are algorithms designed to compute paths that are intended to remain unreserved to complement the reserved primary paths. Like AvoidPBO in Chapter 3, TP and TPmax require routing information of other traffic in the network, and they use that information to determine the conflicts and undesirable links. When the primary and backup path information of traffic flows is available, the steps taken to determine the links to avoid are the same as those taken by AvoidPBO in Sections 3.6.1 and 3.6.2. For example, a source node N_i naturally knows of the primary and backup paths of the flows it originates for routing purposes. N_i can easily determine how its own flows can plan to avoid each other using the primary and backup paths of these flows, as is done by AvoidPBO in Sections 3.6.1 and 3.6.2. Unlike AvoidPBO, when planning to avoid conflicting traffic originated by other source nodes, N_j, $N_i \neq N_j$, the routing information that TP and TPmax use is in the form of *PTP* and *BTP* described in Section 4.2. This section explains the rationale behind the use of *PTP* and *BTP*. Table 10 summarizes the conditions under which a link is determined to be a conflicting link based on *PTP* and *BTP*. Sections 4.4.1, 4.4.2, and 4.4.3 describe the three steps that TP takes to modify link weights based on the routing information available to a source node, and Sections 4.5.1, 4.5.2, and 4.5.3 describe the three steps that TPmax takes.

As mentioned, with TP and TPmax, each source node constructs both *PTP* and *BTP*, and this information is broadcast to other source nodes periodically. Using the example described in Figure 10 and Table 6, source node N_1 constructs the two arrays PTP^{N_1} and BTP^{N_1} described in Table 9 and periodically exchanges these two arrays with N_2. Source node N_2 accepts and stores these two arrays for future use. As described in Section 3.4, the central idea in computing effective backup paths is to identify undesirable links and add weight to those links. For the purpose of computing unreserved backup paths, the undesirable links are the links that are being used by higher priority traffic at the same time that the prospective backup path needs to be activated. With AvoidPBO, the identification of these conflicting links is made with the help of the primary and backup paths taken or planned by all the other traffic in the network. With TP and TPmax, these undesirable links are identified using the *PTP* and *BTP* collected from other source nodes. In this example, source node N_2 uses PTP^{N_1} and BTP^{N_1} when it calculates backup paths for its flows.

Based on the principles established by AvoidPBO, to compute a backup path N_2 first identifies the links used by conflicting primary paths, and then adds extra weight to those links to avoid them. In the second step, N_2 identifies the links used by conflicting higher priority backup paths, and then adds extra weight to those links to avoid them. Thus, to compute the backup path for flow f_3, N_2 first determines the links used by conflicting primary paths that B_3 should avoid. To avoid the links of such flows originating from N_1, N_2 uses PTP^{N_1}. Because $PTP^{N_1}_{9,1}$, $PTP^{N_1}_{9,2}$, $PTP^{N_1}_{9,3}$, $PTP^{N_1}_{10,1}$,

$PTP^{N_1}_{10,2}$, and $PTP^{N_1}_{10,3}$ all store the value of one, indicating that when ℓ_9 or ℓ_{10} fails and B_3 is activated, there is primary path traffic on ℓ_1, ℓ_2, and ℓ_3. Thus, N_2 decides that B_3 should avoid links ℓ_1, ℓ_2, and ℓ_3. As a backup path, B_3 should proactively avoid primary path traffic that is alive at the same time that it is activated, and so extra weight should be added to these three links accordingly.

The second step involves determining and avoiding links of conflicting higher priority backup paths. To compute B_2 for its flow f_2 whose primary path P_2 uses links {ℓ_5, ℓ_2, ℓ_8} (Table 6), N_2 should decide that links ℓ_4, ℓ_5, ℓ_6, and ℓ_7 should be avoided. This decision is made because $BTP^{N_1}_{2,4}$, $BTP^{N_1}_{2,5}$, $BTP^{N_1}_{2,6}$, and $BTP^{N_1}_{2,7}$ all have the value of one, indicating that when ℓ_2 fails, there is one flow using its backup path on ℓ_4, one on ℓ_5, one on ℓ_6, and one on ℓ_7. In addition, when ℓ_2 fails, B_2 is also needed, indicating that B_2 needs to avoid other higher ranked backup paths that are also activated when ℓ_2 fails. Like AvoidPBO, backup paths need to be ranked so that only one conflicting backup path avoids the other. However, in this case, because the backup paths are grouped by source nodes, the simplest ranking of the backup paths is based on the ranking of the source nodes. Assuming that flows originated from N_1 are ranked higher than those from N_2, then backup paths of flows originating at N_2 should proactively avoid potentially conflicting traffic indicated in $BTP^{N_1}_{j,k}$. Thus, f_2 should avoid traffic indicated in $BTP^{N_1}_{2,4}$, $BTP^{N_1}_{2,5}$, $BTP^{N_1}_{2,6}$, and $BTP^{N_1}_{2,7}$.

Table 10 summarizes the conditions under which a link ℓ_k is considered a conflicting link, of either the conflicting primary path link variety or the conflicting backup path link variety.

Table 10. Conflicting links based on comparison between P_x and all pairs of PTP^{N_j} and BTP^{N_j}.

Conflicting link	Condition
Conflicting primary path link ℓ_k	If $PTP^{N_j}_{m,k} > 0$.
Conflicting backup path link ℓ_k	If $BTP^{N_j}_{m,k} > 0$ and priority of N_j is higher than N_i.
Assumption: link $\ell_m \in P_x$.	

4.4 Traffic placement method (TP)

Like AvoidPBO, TP [15] is an algorithm that computes backup paths that are intended to complement their corresponding primary paths to provide added reliability and are intended to remain unreserved when the corresponding primary paths are alive. Each TP source node N_i stores the primary and backup path information of the flows it originates. Based on this information, N_i constructs PTP^{N_i} and BTP^{N_i}. This condensed information is the only information that is communicated into the network. To compute the backup paths, the same principles as first described in Section 3.4 and then reiterated in Section 4.3 above are followed. The basic idea is to determine conflicting links and then add weight to those links so that Dijkstra's algorithm avoids them. The difference between TP and AvoidPBO is that the information used to make this decision is different in certain circumstances.

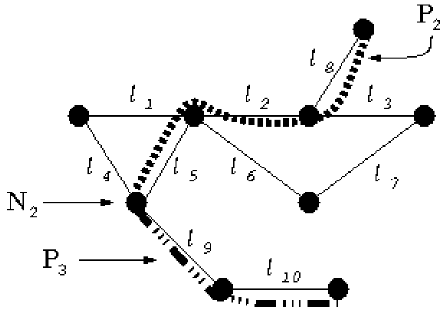
Under TP, to compute the backup path, B_x, of flow f_x, whose reserved primary path is P_x, source node N_i follows the three steps described in Sections 4.4.1, 4.4.2, and 4.4.3 to add extra weight. To begin to compute the backup paths, each TP source node

N_i uses the primary and backup path information of its own flows, which is called the internal information in Section 4.4.1. N_i also uses PTP^{N_j} and BTP^{N_j} received from other nodes N_j, $N_i \neq N_j$, which is called the external information in Section 4.4.2. Like AvoidPBO, TP also adds extra weight on the primary path P_x, which is discussed in Section 4.4.3. Once the extra weight is added, Dijkstra's algorithm is invoked to compute the shortest path, which is designated as B_x. Finally, Section 4.4.4 provides a summary of TP. The pseudo code for TP is provided in Figure 12.

4.4.1 Step 1: extra weight based on internal information

This step involves determining conflicts using the explicit primary and backup path information source node N_i stores regarding its own flows. This explicit routing information is in the same format as the routing information AvoidPBO uses. The difference is that a TP source node only has such information regarding its flows, whereas an AvoidPBO source node eventually has such information regarding all traffic in the network. Thus, in this step, source node N_i uses the same rules established in Sections 3.6.1 and 3.6.2 to modify link weights. The conditions under which a link is determined to be a conflicting link are summarized in Table 4. The implementation of TP is based on the steps described in Sections 3.8.1 and 3.8.2. A summary of the variables used in the calculation of the extra weight is in Table 11 and Table 12, and the actual formulae are in Table 13. Below is an example of the link weight modification process based on the sample scenario in Figure 10 and Table 6.

Based on the sample flows in Figure 10 and Table 6, to compute the backup path, B_3, of flow f_3, N_2 first determines the links to avoid based on the primary and backup path information of f_2. Because N_2 is the source node for both f_2 and f_3, N_2 should have no trouble ranking f_2 and f_3 relative to each other and determining how their respective backup paths should avoid each other. Based on the rule of avoiding conflicting primary path links in Section 3.6.1, N_2 decides that B_3 should avoid ℓ_5, ℓ_2, and ℓ_8. (This is because when B_3 is activated, P_2 is still alive.) Thus, extra is added to links ℓ_5, ℓ_2, and ℓ_8, as depicted in Figure 11.



The prospective B_3 conflicts with P_2.
Thus, extra weight is added to conflicting primary path links, ℓ_5, ℓ_2, and ℓ_8.

Figure 11. Internal information known by the TP source node N_2.

4.4.2 Step 2: extra weight based on external information

In addition to avoiding traffic generated by its own source node in Section 4.4.1, for a flow f_x whose primary path is P_x, the prospective backup path B_x should avoid undesirable links due to the traffic generated by other nodes N_j, $N_i \neq N_j$. This section involves adapting the process described in the previous step such that the external information, PTP^{N_j} and BTP^{N_j}, $\forall N_j \neq N_i$, is used to determine extra link weights. Table 10 summarizes the conditions under which a link is considered a conflicting link, whether it is a conflicting primary path link or a conflicting backup path link. The rest of this section explains the implementation of TP, specifically the actual weight added to such a conflicting link.

Like before, both conflicting primary path links and conflicting backup path links should be avoided, and thus, should have extra weight added. Similar to the basic ideas described in Sections 3.6.1 and 3.6.2, B_X should avoid the primary path traffic that is active when B_X is activated, and B_X should also avoid the higher-ranked backup path traffic that is active when B_X is activated. With TP, PTP^{N_j} is used to determine the conflicting primary path links and the amount of extra weight added to those links. BTP^{N_j} is used to identify the higher priority conflicting backup path links and the extra weight added to those links.

Assume that ℓ_m is a link in P_X ($\ell_m \in P_X$). Because $PTP^{N_j}_{m,k}$ stores the number of primary path flows that ℓ_k carries when ℓ_m fails, $PTP^{N_j}_{m,k}$ is also the number of primary path flows that ℓ_k carries when P_X fails and B_X is activated. Thus, B_X should avoid ℓ_k and extra weight should be added on ℓ_k. In TP, the extra weight added to ℓ_k is simply $PTP^{N_j}_{m,k} * factor$. The total extra weight added to ℓ_k based on PTP^{N_j} is $\text{TP\_}pew^{N_j}_{\ell_k}$ in Table 14, which is equal to $\sum_{\ell_m \in P_X} PTP^{N_j}_{m,k} * factor$.

Similarly, B_X should avoid other backup path traffic that is active the same time that B_X itself is activated. However, because backup paths are not reserved, B_X has as much right to or priority over a particular network resource as any other backup path. To prevent discounting and avoiding too many links, either B_X should avoid other backup path traffic, or vice versa. Similar to Section 3.7 regarding AvoidPBO, a rule should be agreed upon among all nodes to determine which backup path traffic should avoid other conflicting backup path traffic. Assume that backup path traffic originated at source nodes with higher IP addresses (e.g. 216.x.x.x) should avoid the backup path traffic originated from source nodes with lower IP address (e.g. 128.x.x.x). Then, only BTP^{N_g}, where N_g has a lower IP address than N_i, should be used to determine link weights. Assume that $\ell_m \in P_x$. Because $BTP^{N_g}_{m,k}$ stores the number of backup path flows that ℓ_k carries when ℓ_m fails, $BTP^{N_g}_{m,k}$ is also the number of backup path flows that ℓ_k carries when P_x fails and B_x is activated. Like before, a weight of $BTP^{N_g}_{m,k} * factor$ should be added on ℓ_k. The total extra weight added to ℓ_k based on BTP^{N_g} is $TP\_bew^{N_g}_{\ell_k}$ in Table 14, which is equal to

$$\sum_{\substack{\ell_m \in P_x \\ N_g \text{ has higher priority over } N_i}} BTP^{N_g}_{m,k} * factor .$$

A summary of the variables used for the calculation of extra weight is provided in Table 11 and Table 12. Table 14 provides the actual formulae used to calculate the extra weight.

4.4.3 Step 3: extra weight based on the primary path

Like AvoidPBO, the last step is to add enough extra weight onto the primary path, P_x, so that Dijkstra's algorithm can compute a backup path B_x such that it is disjoint from P_x if possible. Weight that is added to a link in the previous two steps, Sections 4.4.1 and 4.4.2, is also added to each link ℓ_m, where $\ell_m \in P_x$. Like AvoidPBO, an additional $|N|$* *factor* is also added to each link ℓ_m, $\ell_m \in P_x$, to discourage the use of any link used by P_x

even if the alternative is to traverse through all the nodes in the network. Based on Table 11 and Table 12, the actual formula to calculate the extra weight is provided in Table 15.

```
computeBackupTP (flow $f_x$) {
    W: weight matrix W.
    $W_k$: weight on link $\ell_k$.
    $PTP^{N_j}$ : condensed primary path information from node $N_j$.
    $BTP^{N_j}$ : condensed backup path information from node $N_j$.
    $P_x$: primary path of $f_x$.
    src: source of $f_x$.
    dest: destination of $f_x$.
    $N_i$: source node of $f_x$.

    for j = 1 .. number of $PTP^{N_j}$-$BTP^{N_j}$ information pairs
        stored
        for each link $\ell_k$ in the network
            add extra weight proportional to $PTP^{N_j}_{m,k}$ to $W_k$,
                where link $\ell_m$ is in $P_x$
        end
        if $N_j$ has higher priority than $N_i$ then
            for each link $\ell_k$ in the network
                add extra weight proportional to $BTP^{N_j}_{m,k}$
                    to $W_k$, where link $\ell_m$ is in $P_x$
            end
    end
    add weight $w_{primary}$, where $w_{primary}$ >> $W_k$, to each link $\ell_m$ in $P_x$
    use Dijkstra's algorithm and the new weight matrix W
        to compute a shortest path between src and dest.
}
```

Figure 12. The pseudo code for TP.

Table 11. Notations and symbols referred to in Table 12, Table 13, Table 14, and Table 15.

Notation	Definition and value
$\mathcal{N}$	The set of nodes in the network.
$\mathcal{L}$	The set of links in the network.
$\mathrm{N_i}$	The source node that originates flow f_x.
$\mathrm{N_j}$	A node, $\mathrm{N_j} \in \mathcal{N}$, but $\mathrm{N_j} \neq \mathrm{N_i}$.
$\mathrm{N_g}$	A node, $\mathrm{N_g} \in \mathcal{N}$, but $\mathrm{N_g} \neq \mathrm{N_i}$. $\mathrm{N_g}$ generally denotes a node whose traffic has higher priority than $\mathrm{N_i}$ when it is used in Chapter 4.
$\mathcal{F}^{\mathrm{N_i}}$	The set of flows f_i that are originated at $\mathrm{N_i}$.
f_x	The flow whose backup path $\mathrm{N_i}$ is computing, $f_x \in \mathcal{F}^{\mathrm{N_i}}$.
$\mathrm{P_x}$	The primary path of flow f_x, which is considered as a set of links.
$\mathrm{B_x}$	The prospective backup path of f_x, which is considered as a set of links.
f_i	A generic flow.
$\mathrm{P_i}$	The primary path of f_i, which is considered as a set of links.
$\mathrm{B_i}$	The backup path of f_i, which is considered as a set of links.

Table 12. Notations based on regarding AvoidPBO from Table 3 and the extra weight determined, also based on AvoidPBO.

Notation	Definition and value
$pew_{\ell_k}^{f_i}$	The extra weight added to the link $\ell_k \in \mathrm{P_i}$ based on comparison between f_i and f_x, which is equal to $\begin{cases} numberOfConflicts(f_i, f_x) * factor & \text{, if } numberOfSharedLinks(\mathrm{P_i}, \mathrm{P_x}) = 0. \\ 0 & \text{, if } numberOfSharedLinks(\mathrm{P_i}, \mathrm{P_x}) > 0. \end{cases}$ (This value is the same as in Table 5.)
$bew_{\ell_k}^{f_i}$	The extra weight added to the link $\ell_k \in \mathrm{B_i}$ based on comparison between f_i and f_x, which is equal to $\begin{cases} 0 & \text{, if } numberOfSharedLinks(\mathrm{P_i}, \mathrm{P_x}) = 0. \\ numberOfConflicts(f_i, f_x) * factor & \text{, if } numberOfSharedLinks(\mathrm{P_i}, \mathrm{P_x}) > 0. \end{cases}$ (This value is the same as in Table 5.)

Table 13. Formulae used to determine extra weight based on internal information by a TP source node.

Notation	Definition and Value
$internal\_pew_{\ell_k}$	The sum of extra weight added on ℓ_k, $\ell_k \in \mathrm{P_i}$, due to all conflicting primary paths based on internal routing information in Section 4.4.1, which is equal to $\sum_{f_i \neq f_x} pew_{\ell_k}^{f_i}$, $\forall f_i \in \mathcal{F}^{\mathrm{N_i}}$.
$internal\_bew_{\ell_k}$	The sum of extra weight added on ℓ_k, $\ell_k \in \mathrm{B_i}$, due to all higher priority conflicting backup paths based on internal routing information in Section 4.4.1, which is equal to $\sum_{\substack{f_i \neq f_x \\ f_i \text{ has higher priority over } f_x}} bew_{\ell_k}^{f_i}$, $\forall f_i \in \mathcal{F}^{\mathrm{N_i}}$.

Table 14. Formulae used to determined extra weight based on the external information by a TP source node.

Notation	Definition and Value
$\mathrm{TP\_}pew_{\ell_k}^{\mathrm{N_j}}$	According to TP (Section 4.4), the weight added on ℓ_k based on $PTP^{\mathrm{N_j}}$, which is equal to $\sum_{\ell_m \in \mathrm{P_x}} PTP_{m,k}^{\mathrm{N_j}} * factor$.
$\mathrm{TP\_}bew_{\ell_k}^{\mathrm{N_g}}$	According to TP (Section 4.4), the weight added on ℓ_k based on $BTP^{\mathrm{N_g}}$, which is equal to $\sum_{\substack{\ell_m \in \mathrm{P_x} \\ \mathrm{N_g} \text{ has higher priority over } \mathrm{N_i}}} BTP_{m,k}^{\mathrm{N_g}} * factor$.

Table 15. Formula used to determine the extra weight to add to ℓ_m, $\forall \ell_m \in \mathrm{P_x}$, by a TP source node.

Notation	Definition and Value
$\mathrm{TP\_}pp\_ew_{\ell_m}$	The extra weight added on ℓ_m, $\forall \ell_m \in \mathrm{P_x}$, is equal to $\sum_{f_i \neq f_x} pew_{\ell_k}^{f_i} + \sum_{\substack{f_i \neq f_x \\ f_i \text{ has higher priority over } f_x}} (bew_{\ell_k}^{f_i}) + \sum_{\mathrm{N_j}} (\sum_{\ell_k \in \mathcal{L}} \mathrm{TP\_}pew_{\ell_k}^{\mathrm{N_j}}) + \sum_{\mathrm{N_g} \text{ has higher priority over } \mathrm{N_i}} (\sum_{\ell_k \in \mathcal{L}} \mathrm{TP\_}bew_{\ell_k}^{\mathrm{N_g}}) + \lvert \mathcal{N} \rvert * factor$

Table 16. Formulae used to determined extra weight based on the external information by a TPmax source node.

Variable	Definition and Value
$\text{TPmax}\_pew_{\ell_k}^{\mathrm{N_j}}$	According to TP (Section 4.4), the weight added on ℓ_k based on $PTP^{\mathrm{N_j}}$, which is equal to $\max_{\ell_m \in \mathrm{P_x}}\{PTP_{m,k}^{\mathrm{N_j}}\} * factor$.
$\text{TPmax}\_bew_{\ell_k}^{\mathrm{N_g}}$	According to TP (Section 4.4), the weight added on ℓ_k based on $BTP^{\mathrm{N_g}}$, which is equal to $\max_{\substack{\ell_m \in \mathrm{P_x} \\ \mathrm{N_g} \text{ has higher priority over } \mathrm{N_i}}} \{BTP_{m,k}^{\mathrm{N_g}}\} * factor$.

Table 17. Formula used to determine the extra weight to add to ℓ_m, $\forall \ell_m \in \mathrm{P_x}$, by a TPmax source node.

Variable	Definition and Value
$\text{TPmax}\_pp\_ew_{\ell_m}$	The extra weight added on ℓ_m, $\forall \ell_m \in \mathrm{P_x}$, is equal to $\sum_{f_i \neq f_x} pew_{\ell_k}^{f_i} + \sum_{\substack{f_i \neq f_x \\ f_i \text{ has higher priority over } f_x}} bew_{\ell_k}^{f_i} + \sum_{\mathrm{N_j}} \sum_{\ell_k \in L} \text{TPmax}\_pew_{\ell_k}^{\mathrm{N_j}} + \sum_{\mathrm{N_g} \text{ has higher priority over } \mathrm{N_i}} (\sum_{\ell_k \in \boldsymbol{L}} \text{TPmax}\_bew_{\ell_k}^{\mathrm{N_g}}) + \lvert \boldsymbol{N} \rvert * factor$

4.4.4 TP summary

TP is an algorithm that computes backup paths that are intended to stay unreserved prior to a failure but still remain effective after a failure. It is based on AvoidPBO. The difference is that its source nodes construct and communicate their routing information in the *PTP* and *BTP* format (Section 4.2). In this case, *PTP* and *BTP* are used to identify conflicting links and determine the extra weight to be added. The formulae to compute the extra weight are summarized in Table 13, Table 14, and Table 15. Like AvoidPBO, for a flow f_x, once link weights are modified based on all the available routing information, Dijkstra's algorithm is used to compute a shortest path, which is designated as the backup path B_x.

4.5 Traffic placement method using the max{} function (TPmax)

TPmax is similar to TP. The only difference is in the calculation of the extra weight using *PTP* and *BTP*. Once link weight modification is complete, Dijkstra's algorithm is invoked to compute the shortest path, which becomes the eventual backup path, just like AvoidPBO and TP. Sections 4.5.1, 4.5.2, and 4.5.3 present the formulae used to calculate the extra weight, and Section 4.5.4 provides a short summary of the TPmax algorithm. The formulae used to calculate extra weight are summarized in Table 13, Table 16, and Table 17. The pseudo code for TPmax is provided in Figure 13.

```
computeBackupTPmax (flow $f_x$) {
    W: weight matrix W.
    $W_k$: weight on link $\ell_k$.
    $PTP^{N_j}$ : condensed primary path information from node $N_j$.
    $BTP^{N_j}$ : condensed backup path information from node $N_j$.
    $P_x$: primary path of $f_x$.
    src: source of $f_x$.
    dest: destination of $f_x$.
    $N_i$: source node of $f_x$.

    for j = 1 .. number of $PTP^{N_j}$-$BTP^{N_j}$ information pairs
        stored
        for each link $\ell_k$ in the network
            max ← the maximum among $PTP^{N_j}_{m,k}$ where link $\ell_m$ is
                in $P_x$
            add extra weight proportional to max to $W_k$
        end
        if $N_j$ has higher priority than $N_i$ then
            for each link $\ell_k$ in the network
                max ← the maximum among $BTP^{N_j}_{m,k}$ where
                    link $\ell_m$ is in $P_x$
                add extra weight proportional to max to
                    $W_k$
            end
    end
    add weight $w_{primary}$, where $w_{primary}$ >> $W_k$, to each link $\ell_m$ in $P_x$
    use Dijkstra's algorithm and the new weight matrix W
        to compute a shortest path between src and dest.
}
```

Figure 13. The pseudo code for TPmax.

4.5.1 Step 1: extra weight based on internal information

As mentioned in Section 4.4.1, this step refers to the adjustment of link weights based on the explicit primary and backup path routing information known to the source node itself, i.e., the information concerning the flows that the source node originates. Details regarding this step can be found in Section 4.4.1 and in the discussion regarding AvoidPBO in Sections 3.4, 3.6.1, 3.6.2, 3.8.1, and 3.8.2. The formulae for computing the extra weight are provided in Table 13.

4.5.2 Step 2: extra weight based on external information

Like TP, this step involves adapting the process of determining undesirable links and their link weights in Section 4.5.1 to use *PTP* and *BTP*. The underlying principles are the same as TP. The rationale of using *PTP* and *BTP* was already discussed in Section 4.3 and details of steps taken by TP are provided in Section 4.4.2. This section will only discuss the formulae TPmax uses to calculate the actual extra weight.

Assume that $\mathrm{N_i}$ is the source node that originates f_x, whose primary path is $\mathrm{P_x}$ and the prospective backup path is $\mathrm{B_x}$. Assume also that ℓ_m is a link on $\mathrm{P_x}$ ($\ell_m \in \mathrm{P_x}$). Then, the prospective $\mathrm{B_x}$ conflicts with primary path link ℓ_k if $PTP_{m,k}^{\mathrm{N_j}} > 0$ where $\mathrm{N_j} \neq \mathrm{N_i}$. In TPmax, $\max\limits_{\ell_m \in \mathrm{P_x}}\{PTP_{m,k}^{\mathrm{N_j}}\} * factor$ is added to ℓ_k, as shown in Table 16.

Similarly, the prospective $\mathrm{B_x}$ conflicts with backup path link ℓ_k if $BTP_{m,k}^{\mathrm{N_g}} > 0$ where $\mathrm{N_g} \neq \mathrm{N_i}$ and $\mathrm{N_g}$ has higher priority over $\mathrm{N_i}$. In TPmax, $\max\limits_{\substack{\ell_m \in \mathrm{P_x} \\ \mathrm{N_g} \text{ has higher priority over } \mathrm{N_i}}}\{BTP_{m,k}^{\mathrm{N_g}}\} * factor$ is added to ℓ_k as show in Table 16.

4.5.3 Step 3: extra weight based on the primary path

Like AvoidPBO and TP, the last step is to add enough extra weight onto the primary path $\mathrm{P_x}$, so that Dijkstra's algorithm can compute a backup path $\mathrm{B_x}$ such that it is disjoint from $\mathrm{P_x}$ if possible. The extra weight added to links used by $\mathrm{P_x}$ is shown in Table 17.

4.5.4 TPmax summary

TPmax is an algorithm that computes backup paths that are intended to stay unreserved prior to a failure but still be effective after a failure. It is based on AvoidPBO and is similar to TP. It is different from AvoidPBO because its source nodes construct and communicate their routing information in the *PTP* and *BTP* format (Section 4.2). In addition to the explicit routing information used by AvoidPBO, *PTP* and *BTP* are also used to identify conflicting links and determine the extra weight to be added to those links. The difference between TP and TPmax is in the calculation of extra weight. The formulae to compute the extra weight are summarized in Table 13, Table 16, and Table 17. Like AvoidPBO, for a flow f_x, once link weights are modified based on all the available routing information, Dijkstra's algorithm is used to compute a shortest path, which is designated as the backup path B_x.

4.6 Variations

Like AvoidPBO, the algorithms in this chapter can be extended to adapt to traffic with variable bandwidth (resource) requirements. Instead of storing the number of flows in $PTP_{j,k}^{N_i}$, the amount of bandwidth, or the factored amount of resources, can be stored. Additionally, like the priority of traffic flows in Section 3.7, the ordering of nodes can also be done differently, such as through voting. These extensions are left for future work (Section 7.3). In the simulations described in Section 4.8.2 and Chapter 6, all traffic flows are assumed to require the same bandwidth, and thus $PTP_{j,k}^{N_i}$ stores the number of flows. In these simulations, the priority of the traffic that determines the avoiding among conflicting backup path traffic is as described in Section 4.4.2.

4.7 Differences

4.7.1 Complete path information vs. *PTP* and *BTP*

One big difference between complete path information and *PTP* and *BTP* is in their size. In fact, the sheer size of complete path information is the driving reason behind the development of *PTP* and *BTP*. Communicating complete path information of all traffic flows is cost prohibitive in practice. In contrast, as long as the topology remains the same, the size of *PTP* and *BTP* remains constant regardless of the amount of traffic in the network. This aspect is further discussed in Section 4.8 and evaluated in Chapter 6.

The other big difference is the knowledge that the two different formats provides. With complete path information, more information regarding how traffic is routed is available. For example, based on the sample flows described in Table 6 and illustrated in Figure 10, if the complete path of P_1 is known, then one can deduce that ℓ_1, ℓ_2, and ℓ_3 all cause P_1 to fail and B_1 to be activated. However, in the condensed form of PTP^{N_1} and BTP^{N_1}, the source node, such as N_2, that has PTP^{N_1} and BTP^{N_1} only knows that when ℓ_1 fails, ℓ_4 should be avoided because $BTP_{1,4}^{N_1} > 0$ (Table 9), and that when ℓ_2 fails, ℓ_4 should also be avoided because $BTP_{2,4}^{N_1} > 0$. Using PTP^{N_1} and BTP^{N_1}, N_2 cannot tell that either failure causes the same backup path to be activated. Because of this, two algorithms, TP and TPmax, are designed, and they represent two different ways to use *PTP* and *BTP*. The difference between TP and TPmax is further discussed in Section 4.7.2.

4.7.2 TP vs. TPmax

As mentioned in Section 4.7.1, using *PTP* and *BTP*, a source node cannot discern if two different failure scenarios result in the same backup path to being activated. With TP, the decision is to treat each failure scenario independently. In contrast, TPmax treats all failures as connected. For example, based on the sample flows and sample PTP^{N_1} and BTP^{N_1} in Table 6 and Table 8, to compute the backup path B_2, the TP source node N_2 would add $PTP^{N_1}_{5,1} * factor$ and $PTP^{N_1}_{8,1} * factor$ to ℓ_1. This source node assumes that failures at ℓ_5 and ℓ_8 result in different primary paths to remain alive at ℓ_1. With a TPmax source node N_2, the weight added to ℓ_1 is $\max_{\ell_m \in \{\ell_5, \ell_2, \ell_8\}} \{PTP^{N_1}_{m,1}\}$. This is based on the assumption that the primary path traffic that remains at ℓ_1 under the three failure cases is the same, and thus does not require duplicate extra weight added.

Table 18. Upper bounds of the communication overhead for AvoidPBO, TP, TPmax, and disjoint methods.

Method	Communication Overhead	Upper bound
AvoidPBO (Chapter 3)	Primary and backup path information of each flow is communicated to all nodes in the network.	*O (FL)*
TP TPmax	Constant-sized $PTP_{j,k}^{N_i}$ and $BTP_{j,k}^{N_i}$ of each node N_i is communicated to all nodes in the network.	*O (NL)*
Disjoint	No extra information is needed.	*O (1)*

4.8 *Comparisons*

4.8.1 Communication overhead upper bounds

This section compares the upper bounds of the overhead required to communicate extra routing information among AvoidPBO, TP, TPmax, and the disjoint method. Like Section 3.10, the disjoint method uses the shortest path disjoint from the primary path as the backup path. It is considered the least expensive method because it does not require any extra routing information or computation other than invoking Dijkstra's algorithm.

Assume that *F* is the number of flows, *N* is the number of nodes, and *L* is the number of links in the network. Broadcasting a message is assumed to cost *O(L)*, where a message is sent out at most once along all the links in the network. The communication overhead of the different methods compared is summarized in Table 18. Because *F* >> *L* and *F* >> *N*, the communication overhead incurred by AvoidPBO can be seen as bounded by *F*, and its communication cost is not scalable as it can increase indefinitely even in a fixed network. Conversely, the overhead of TP and TPmax is bounded only by *O(NL)*, which is constant in a fixed network.

4.8.2 Test for TP's and TPmax's ability to not reuse links

The same test performed in Section 3.10 is performed on TP and TPmax. This test computes the value *ratio*, where $ratio = \frac{\sum_i ABPL_{failCase_i}}{NFC}$. It approximates the contention for a link by the activated backup paths. A lower *ratio* is better in this case because it means the expected number of activated backup paths using a particular link is low. A *ratio* of one indicates that only one backup path is expected to use any particular link, which is highly likely to be the optimal result in a heavily loaded network. Figure 14 shows that TP and TPmax both produce a lower *ratio* than AvoidPBO, despite incurring much less overhead than AvoidPBO (Section 4.8.1). Although Figure 14 shows encouraging results for TP and TPmax, because the calculation of *ratio* does not take into account the links used by the primary paths that remain alive after a failure, having the lowest *ratio* does not necessarily mean that TP will have the best performance overall. However, this simple test shows that, compared to the disjoint method and even AvoidPBO, the algorithms using *PTP* and *BTP*, especially TP, achieve the particular objective of reducing link contention among backup paths.

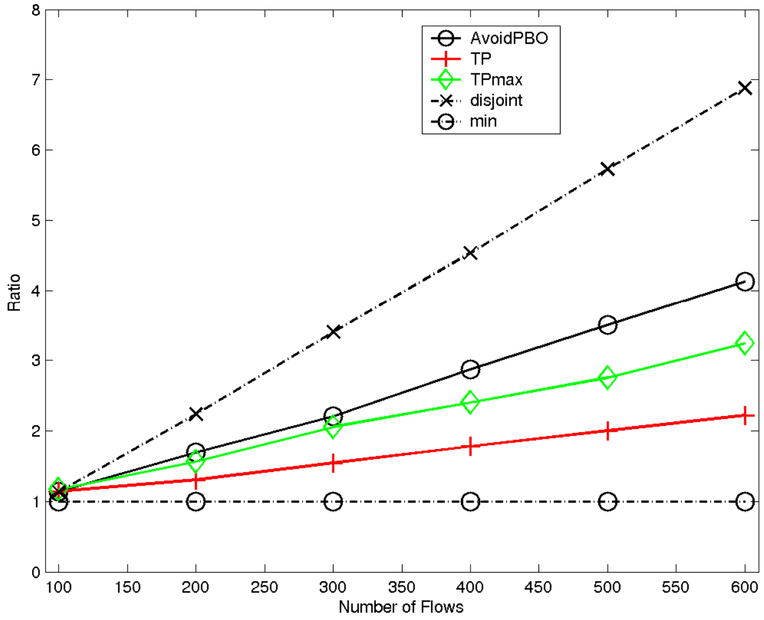


Figure 14. Comparison of the ratio of total number of links used to the number of distinct links used, among AvoidPBO, TP, TPmax, and disjoint backup paths.

4.9 Chapter summary

This chapter describes a way to condense the complete routing information required by AvoidPBO such that the condensed format still contains critical information to identify conflicting links. Two algorithms, TP and TPmax, which compute unreserved backup paths based on the condensed information *PTP* and *BTP*, are developed. Asymptotically, TP and TPmax incur only constant overhead (in a fixed network) whereas the overhead incurred by AvoidPBO can increase indefinitely as the amount of traffic in a network grows. A simple diagnostic test performed on TP, TPmax, and AvoidPBO shows that the ability of both TP and TPmax to reduce contention for the same resources by backup paths is better than AvoidPBO. The following chapters discuss more comprehensive tests to evaluate unreserved backup path computation methods.

5 Recovery Modes

5.1 Chapter outline

Because the backup paths are unreserved prior to the link failure, there are limitations to the performance of these backup paths. Two different ways to use an unreserved backup path once it needs to be activated—strict recovery and relaxed recovery, are proposed in this chapter. Each recovery mode results in different behaviors after a failure. The strict recovery provides hard service guarantee even after a failure by sacrificing some connections. The relaxed recovery maintains the connections of all traffic flows by sacrificing service quality after a failure. The differences and the performance metrics used to evaluate the different algorithms are discussed in Sections 5.2, 5.3, 5.4, and 5.5. These metrics are also summarized in Table 19, Table 20, Table 21, and Table 22. Section 5.6 provides a summary of this chapter. The simulation data provided in Chapter 6 are based on these metrics.

5.2 Strict recovery

The use of unreserved backup paths to overcome a link failure arose from the waste of resources when two reserved paths (Sections 1.2.2 and 2.5) are used to tolerate a failed primary path. When both the primary and backup paths are reserved prior to the failure, traffic flows are always guaranteed to recover from a link failure. The service qualities experienced by these traffic flows are the same before and after the failure. Pairing unreserved backup paths with the strict recovery mode is a direct comparison between the use of reserved backup paths and the use of unreserved backup paths.

With unreserved backup paths each flow has a reserved primary path and an unreserved backup path. Although the backup path remains unreserved prior to the failure, it is the goal of a backup path computation method to find a path that is most likely to have the resources available to re-route an additional flow after a failure. With the strict recovery mode, in the event of a link failure that causes the primary path to fail, resources are reserved on the backup path to provide the same QoS as before the failure. Because the backup paths are unreserved initially, there is no guarantee that the resource reservation attempt on the backup path after a failure will be successful. Consequently, some flows will not recover from the failure while others will. Like the use of two reserved paths, the flows that recover from the failure experience the same service qualities as before the failure. In other words, in the strict recovery mode, the service guarantee agreements agreed upon before the failure are strictly followed even after a failure. This hard service guarantee after a failure is attained by sacrificing some flows. In this case, the performance metrics of interest, which are discussed in Section 5.3, generally involve the number of flows that can be successfully re-routed after a failure.

5.3 Strict recovery performance metrics

The performance metrics under the strict recovery mode are summarized in Table 19 and Table 20, and the details of the environment and the expected behaviors under the strict recovery mode are discussed in Sections 5.3.1 and 5.3.2.

5.3.1 Environment

Assume that a network has a limited amount of resources on each link. Assume also that before the failure, the traffic flows in the network use their respective primary paths and that these primary paths are reserved to provide the desired service quality. In other words, all flows have 100% of the QoS they need. When a link failure occurs, some backup paths may need to be activated. Resource reservation on these backup paths is attempted first. If resource reservation is successful, the flow is re-routed on to the backup paths and is considered to have survived the failure. If there are not enough resources on all the links on the backup path, the flow is considered failed and not re-routed on the backup path at all. Overall, if a flow stays connected after the failure, that flow is routed either on the original reserved primary path or the newly reserved backup path, and the service quality provided to the flow remains unchanged.

5.3.2 Performance metrics

Because traffic and network topologies are rarely uniform, the effects of single link failures are not equal. In certain cases, a link failure may not be felt at all because the link in question was not in use at the time of failure. Depending on the distribution of the traffic and the network topology, each case of single link failure may result in a different number of successful flows and a different set of successful flows. Thus, one comparison between the use of two reserved paths and unreserved backup paths can be taken from the overall perspective of how effective the unreserved backup paths are under different failure scenarios. With two reserved paths, recovery is guaranteed for every case of single link failure. With unreserved backup paths, depending on network conditions and the location of the failure, some failure scenarios may result in disconnection for some flows. As summarized in Table 19, assume that $\mathcal{FC}$ is the set of failure cases tested and that $\mathcal{PRC}$ is the set of failure cases where all of the flows that require re-routing after a failure can be successfully re-routed. Then, the *perfect recovery rate*, defined as the percentage of failure cases where all flows are successfully re-routed, is equal to $\frac{|\mathcal{PRC}|}{|\mathcal{FC}|} * 100\%$. Because the approach of two reserved paths always recovers from a single link failure regardless of the location of the failure or the distribution of the traffic, the perfect recovery rate for the approach of two reserved paths can be considered 100%. In contrast, the perfect recovery rate of the unreserved backup path approach is unlikely to be 100%, and the rate observed can be an indication of how effective unreserved backup paths are compared to the use of two reserved paths.

Although unreserved backup paths cannot guarantee perfect recovery under all circumstances, it is likely that many flows do recover from their failed primary paths under many circumstances. To determine how well these unreserved backup paths recover from a link failure, the metrics *recovery rate* and *success rate* are used. As summarized in Table 19, assume that $\mathcal{F}$ is the set of flows started at the beginning of the simulation, $\mathcal{FC}$ is the set of single link failure scenarios that are considered, $\mathcal{FP}_{failCase_i}$ is the set of flows with failed primary paths under $failCase_i$ where $failCase_i \in \mathcal{FC}$, $\mathcal{SRR}_{failCase_i}$ is the set of flows $f \in \mathcal{FP}_{failCase_i}$ that are successfully re-routed on backup paths under $failCase_i$ where $failCase_i \in \mathcal{FC}$, and $\mathcal{SF}_{failCase_i}$ is the set of flows that are successful even after a single link failure, routed either on the primary path or the backup path, under $failCase_i$ where $failCase_i \in \mathcal{FC}$. Then, the recovery rate is defined as the percentage of successfully re-routed flows among the flows that require re-routing, $\frac{1}{|\mathcal{FC}|}\sum_{failCase_i \in \mathcal{FC}} \frac{|\mathcal{SRR}_{failCase_i}|}{|\mathcal{FP}|} * 100\%$. It is aimed to show, on average, the percentage of flows that can recover from their failed primary paths. The success rate is defined as the percentage of all successful flows after a failure, $\frac{1}{|\mathcal{FC}|}\sum_{failCase_i \in \mathcal{FC}} \frac{|\mathcal{SF}_{failCase_i}|}{|\mathcal{F}|} * 100\%$. Because a single link failure only affects a limited number of flows, it is expected that the success rate will be close to 100%. In contrast, because the recovery rate depends solely on the effectiveness of the unreserved backup paths, it is expected that the backup paths become less effective as the traffic load increases, resulting in considerably lower recovery rate under some circumstances.

5.4 Relaxed recovery

In circumstances where the network was already heavily loaded with traffic before the failure, it is expected that few backup paths can be successfully reserved after the failure under the strict recovery mode described in Section 5.2. Because of the strict adherence to the service agreement even after a failure, few flows, if any at all, are expected to overcome their failed primary paths using unreserved backup paths. To reduce the amount of disconnected traffic, an alternative is to relax the service agreement after the failure. This relaxed recovery mode allows all the flows to remain connected by sacrificing the service quality provided to traffic after the failure. Unlike the strict recovery mode, resource reservation is not attempted on the backup paths before or after the failure. If a primary path has failed, the corresponding backup path is always activated for re-routing. Due to the missing/failed link, there are fewer resources available in the network to transport the same amount of traffic. Consequently, the service quality is expected to be lower than before the failure, though all flows stay connected. Thus, the goal in this case is to minimize the amount of service degradation after a failure. Section 5.5 discusses the performance metrics used to compare different backup path computation methods under the relaxed recovery mode.

5.5 Relaxed recovery performance metrics

The performance metrics under the relaxed recovery mode are summarized in Table 21 and Table 22, and the details of the environment and the expected behaviors under the relaxed recovery mode are discussed in Sections 5.5.1 and 5.5.2.

5.5.1 Environment

Similar to the strict recovery mode, each flow begins by using its reserved primary path. Like before, the primary path is chosen and enough resources are reserved so that it can provide the required QoS. A backup path computation algorithm chooses a complementing backup path that remains unreserved but is expected to provide good service after a failure. In the event of a link failure, re-routing for all the flows affected by the failure is allowed.

Assume that each link has *n* units of resources and that each flow requires one unit of resource on every link in its path to satisfy its service requirement. Thus, to satisfy the service guarantee for existing traffic initially, each link carries at most *n* traffic flows. Assume that before the failure each flow has one unit of resource reserved on every link of its forwarding path. Thus, each flow is provided with 100% of its service guarantee. With relaxed recovery, after a failure, traffic is routed onto the pre-planned backup paths when needed, resulting in some links carrying more than *n* flows. A link carrying more than *n* flows is considered to be congested, and the service quality offered to the flows using this link is degraded. If a link carries less than *n* flows, then QoS provided by this link is 100% as promised. If a link carries *k* flows, $k > n$, and each link only has *n* units of resources, then these flows will experience some QoS degradation. If the most congested link used by a flow carries *m* flows, where $m > n$, then the QoS of the flow using this link is at most $\frac{n}{m} * 100\%$ of the original QoS guarantee, resulting in a drop of service quality of $100\% - \frac{n}{m} * 100\%$. Using the relaxed recovery mode, although all flows stay connected, some flows may suffer degraded service. So, for this recovery mode, the resulting degradation of service, in the form of the amount of resources available to a flow, is examined. Section 5.5.2 provides detailed discussion of these performance metrics, which are summarized in Table 21 and Table 22.

5.5.2 Performance metrics

As summarized in Table 21, assume that ℓ_i is a link in the network and that a flow *f* takes the path $\{\ell_1, \ell_2, \ldots, \ell_u\}$. Assume also that $\mathcal{F}$ is the set of flows started at the beginning of simulation, and that *n* is the units of resources on each link ℓ_i. Thus, *n* is the maximum number of flows each link ℓ_i can carry without compromising QoS, and before the failure, *n* is the maximum number of flows a link actually carries. Assume that c_{ℓ_i} is the number of flows that ℓ_i carries, and that $\mathcal{D}_{failCase_i}$, defined as the set of flows using one or more congested links under $failCase_i$ where $failCase_i \in \mathcal{FC}$, is equal to $\{f \mid \exists\ c_{\ell_i} > n\}$, where flow *f* takes the path $\{\ell_1, \ell_2, \ldots, \ell_u\}$. Then, the set of flows in $\mathcal{D}_{failCase_i}$ are considered to be suffering from service degradation after the failure and are the traffic of interest in the relaxed recovery mode. In particular, metrics used for comparison are the *number of congested flows*, the *worst case resource availability,* and the *average case resource availability*.

As mentioned, $\mathcal{D}_{failCase_i}$ is the set of flows that suffer from service degradation after the failure under $failCase_i$. Thus, the number of congested flows is defined as $\frac{1}{|\mathcal{FC}|}\sum_{failCase_i \in \mathcal{FC}} \mathcal{D}_{failCase_i}$. Although it is preferable to have as few congested flows as possible, this value should also be balanced with the worst case resource availability, which measures the level of service degradation experienced by flows that actually suffer from service degradation.

The worst case service degradation considers the level of service downgrade of only the flows that suffer such a condition. The level of service degradation suffered by a flow is measured in terms of the amount of resources that are available to the flow after a failure. As mentioned in Section 5.5.1, it is assumed that before the failure, a flow is provided with 100% of the resources it needs, i.e., one unit of resource on each link in the forwarding path. For flows that suffer service degradation, the resources on their forwarding paths are shared by more traffic, and thus these flows receive less than one unit of resource on each link. Assume that the service degradation of a flow *f*, SD_f, is defined as the minimum amount of resources available to flow *f*, which is calculated with the formula

$$\begin{cases} \dfrac{n}{\max\{c_{\ell_1}, c_{\ell_2}, \ldots, c_{\ell_u}\}} * 100\%, & \text{if } \max\{c_{\ell_1}, c_{\ell_2}, \ldots, c_{\ell_u}\} > n, \\ & \text{where } f \text{ takes the path } (\ell_1, \ell_2, \ldots, \ell_u). \\ 100\% \quad \text{otherwise.} & \end{cases}$$

Then, the worst case resource availability is equal to $\dfrac{1}{|\mathcal{FC}|} \displaystyle\sum_{\mathit{failCase}_i \in \mathcal{FC}} \dfrac{\sum_{f \in \mathcal{D}_{\mathit{failCase}_i}} \mathrm{SD}_f}{\left|\mathcal{D}_{\mathit{failCase}_i}\right|} * 100\%$. It is expected that some flows will continue to have 100% of the resources they desire, while some flows will not after the failure. For the flows that do not have the desired amount of resources, the better path computation methods are expected to make more resources available to these flows, and thus, should yield a higher worst case resource availability.

The last metric of concern is the average case resource availability. This metric is defined to be the average service degradation of all flows, and is equal to

$$\frac{1}{|\mathcal{FC}|} \sum_{\mathit{failCase}_i \in \mathcal{FC}} \frac{\sum_{f \in \mathcal{F}} \mathrm{SD}_f}{|\mathcal{F}|} * 100\% .$$

In a way, this value combines the number of congested flows and the worst case resource availability. It is expected that methods with a very small number of congested flows might yield a good result because these non-congested flows all have 100% of resource availability. From this perspective, high average case resource availability does not necessarily indicate a good method because it is possible for the method to be unbalanced by providing very good service for some flows and very bad service for others. If the goal is to provide the least amount of service degradation, then the worst case resource availability is the more important metric. However, all three metrics should be considered together for a more complete understanding of the performance of different methods. Table 22 summarizes the three metrics to evaluate different backup path computation methods under the relaxed recovery mode.

5.6 Chapter summary

In light of the drawback of wasted resources when two reserved paths, reserving both the primary and the backup paths in advance, are used, unreserved backup paths are proposed. Traditionally, the hope for unreserved backup paths is the same as that for the reserved backup paths: to mask the single link failure. In other words, the traditional way to use unreserved backup paths is under the static recovery mode (Section 5.2), and the goal of routing algorithms is to compute backup paths that can provide the same service quality as before the failure. The side effect of this approach is that some flows will completely fail. As the network load becomes heavier, it is expected that very few flows will actually recover from a failure, rendering the unreserved backup paths and the work done to compute these backup paths useless.

Instead of providing reliable hard service guarantees, a better fit for unreserved backup paths may be to relax the service guarantee after a failure with the goal of minimizing the effect of the failure. This is the approach taken in the relaxed recovery mode (Section 5.4), with the hope that good backup paths can be chosen so that minimal degradation of service can be achieved. The two applications of backup paths are summarized in Table 23, and the performance metrics of concern are provided in Table 20 and Table 22.

Table 19. Notations used in strict recovery performance metrics.

Data	Definition
$\mathcal{FC}$	The set of failure cases tested whose performance data are collected. In a network with *n* directed links, the largest possible $\mathcal{FC}$ includes all *n* links.
$\mathcal{PRC}$	The set of failure cases where *all* of the flows that require re-routing after a failure can be successfully re-routed based on strict recovery.
$\mathcal{F}$	The set of flows started at the beginning of the simulation.
$\mathcal{FP}$	The set of flows with failed primary paths.
$\mathcal{SRR}_{\mathit{failCase}_i}$	The set of flows $f \in \mathcal{FP}$ that are successfully re-routed on backup paths, under $\mathit{failCase}_i$ where $\mathit{failCase}_i \in \mathcal{FC}$.
$\mathcal{SF}_{\mathit{failCase}_i}$	The set of flows that are successful even after a single link failure, routed either on the primary path or backup path, under $\mathit{failCase}_i$ where $\mathit{failCase}_i \in \mathcal{FC}$.

Table 20. Strict recovery performance metrics.

Metric	Definition	Value
Recovery rate	The percentage of flows successfully re-routed (only among flows that require re-routing).	$\frac{1}{\lvert\mathcal{FC}\rvert}\sum_{\mathit{failCase}_i \in \mathcal{FC}} \frac{\lvert\mathcal{SRR}_{\mathit{failCase}_i}\rvert}{\lvert\mathcal{FP}\rvert} * 100\%$
Success rate	The percentage of all successful flows after a failure.	$\frac{1}{\lvert\mathcal{FC}\rvert}\sum_{\mathit{failCase}_i \in \mathcal{FC}} \frac{\lvert\mathcal{SF}_{\mathit{failCase}_i}\rvert}{\lvert\mathcal{F}\rvert} * 100\%$
Perfect recovery rate	The percentage of failure cases where all the flows are successfully re-routed	$\frac{\lvert\mathcal{PRC}\rvert}{\lvert\mathcal{FC}\rvert} * 100\%$

Table 21. Notations used in relaxed recovery performance metrics.

Data	Definition	Formula
$\mathcal{F}$	The set of flows started at the beginning of simulation.	
$\mathcal{FC}$	The set of failure cases tested whose performance data are collected. In a network with *n* directed links, the largest possible $\mathcal{FC}$ includes all *n* links.	
n	The units of resources on each link ℓ_i. This is the maximum number of flows each link ℓ_i can carry without compromising QoS.	
c_{ℓ_i}	The number of flows the link ℓ_i carries.	
$\mathcal{D}_{failCase_i}$	The set of flows using one or more congested links under $failCase_i$.	$\{f \mid \exists\ c_{\ell_i} > n\}$ where flow *f* takes the path $\{\ell_1, \ell_2, \ldots, \ell_u\}$.
SD_f	The service degradation of flow *f*, which is equal to the minimum amount of resources available to flow *f*.	$\begin{cases} \dfrac{n}{\max\{c_{\ell_1}, c_{\ell_2}, \ldots, c_{\ell_u}\}} *100\%, & \text{if } \max\{c_{\ell_1}, c_{\ell_2}, \ldots, c_{\ell_u}\} > n, \\ & \text{where } f \text{ takes the path } (\ell_1, \ell_2, \ldots, \ell_u). \\ 100\% \quad \text{otherwise.} & \end{cases}$

Table 22. Relaxed recovery performance metrics.

Metric	Definition	Formula
Number of congested flows		$\dfrac{1}{\lvert\mathcal{FC}\rvert} \sum_{failCase_i \in \mathcal{FC}} \mathcal{D}_{failCase_i}$
Worst case resource availability	The service degradation of flows that use congested links.	$\dfrac{1}{\lvert\mathcal{FC}\rvert} \sum_{failCase_i \in \mathcal{FC}} \dfrac{\sum_{f \in \mathcal{D}_{failCase_i}} \mathrm{SD}_f}{\lvert\mathcal{D}_{failCase_i}\rvert} *100\%$
Average case resource availability	The average service degradation of all flows.	$\dfrac{1}{\lvert\mathcal{FC}\rvert} \sum_{failCase_i \in \mathcal{FC}} \dfrac{\sum_{f \in \mathcal{F}} \mathrm{SD}_f}{\lvert\mathcal{F}\rvert} *100\%$

Table 23. A comparison of the strict and the relaxed recovery modes.

	Strict recovery	Relaxed recovery
Trade-offs	• Service quality remains the same before and after the failure. • Some flows may disconnect after the failure.	• Service quality is likely worse after the failure. • All flows stay connected after the failure.
Goal	To minimize disconnection.	To minimize service degradation.

6 Evaluation

6.1 Chapter outline

In addition to the tests shown in Sections 3.10 and 4.8.2, this chapter provides additional evaluation of AvoidPBO, TP, and TPmax on different topologies, traffic distributions, and recovery modes (Chapter 5). Sections 6.2 and 6.3 describe the simulation program in OPNET [62] and the resulting simulation environment. Section 6.4 discusses the performance metrics, particularly how the communication cost is calculated. Comparisons to known methods APLV Norm, CV, and BV+APV ([48] [49]) that also compute backup paths that are intended to be unreserved prior to the failure, are also provided in this chapter. Section 6.5 provides an overview of all the methods tested in this chapter as well as the expected overhead in terms of the communication cost. Sections 6.6, 6.7, and 6.8 provide the simulation results of three different scenarios. These scenarios are different yet related so that educated guesses regarding the performances of other conditions can be extended from these results. Finally, Section 6.9 provides a summary of this chapter.

6.2 Simulation program

The simulations discussed in this chapter were performed in OPNET [62]. OPNET was chosen because it allowed the traffic to work itself out when resource reservation is attempted. The simulation program was built similarly to Section 2.10.3, where an OSPF-like network layer is responsible for distributing the routing information. Like the link-state information exchange, routing information that is required to be communicated among all the nodes in the network is done with exchange of information between neighbors. Packet forwarding of data packets is done in the same manner as MPLS, where a path is first established and labels are used to indicate the forwarding path.

At the beginning of the simulation, traffic flows are laid out over the network. Data packet generation, routing information exchange, and backup path re-computation occur at regular intervals based on the input parameters. At a pre-selected time, a link failure is injected into the simulation and re-routing of affected flows is performed. Re-routing is performed based on the recovery modes described in Chapter 5. The simulation completes once a decision is made on whether the re-routed flows are successful.

6.3 Simulation choices

6.3.1 Simulation environment

Because fine-grained, discrete events are simulated, the computation times of each data point in the heaviest traffic loads shown in Sections 6.6, 6.7, and 6.8 range from almost one week (for TP, TPmax, APLV Norm, CV, and BV+APV) to a little over two weeks (for AvoidPBO). Under this circumstance, it was not feasible to consider a multiplicity of graphs. Thus, the approach of the simulations in this chapter is to use moderate environments to compare the performance of the algorithms proposed in this thesis and other previous methods. The aim for these experiments is to develop an understanding of the behavior of unreserved backup paths as a whole. The choices of simulated graphs and traffic are rationalized in Sections 6.3.2 and 6.3.3 respectively.

6.3.2 Choice of graphs

The three setups in Sections 6.6, 6.7, and 6.8 are chosen to show the general behavior and performance limitations of unreserved backup paths. These setups are modeled after existing work [48] [49], though slightly bigger graphs are chosen. The graphs used below all consist of 20 nodes, and depending on the setup, each node has a degree of four or five. While there are many types of graphs to choose from, this type of graph is a moderate choice in terms of environments that can best use unreserved backup paths in a single AS. With a tree or tree-like graph, there will not be many alternatives to choose as backup paths. Thus, the results obtained under such a condition may simply indicate a limitation of the network, rather than limitations of the algorithms. In contrast, if there are too many alternatives because the nodes are too well-connected, then the results under this condition may not be indicative of the limitations of the algorithms.

6.3.3 Choice of traffic model

As mentioned in Section 6.3.1, at this stage, the aim is to explore the possibilities and potential of unreserved backup paths. Instead of simulating special traffic characteristics, such as audio or video traffic, generic traffic is assumed. Any special handling of traffic is left for future work (Section 7.3). Specifically, the actual simulated traffic is designed to provide an understanding of the capabilities of unreserved backup paths as well as the range of overhead. Thus, different traffic loads are simulated and different types of traffic distributions are tested for comparison.

6.4 Performance metrics

The simulation results based on the metrics described in Chapter 5 are provided and discussed in Sections 6.6, 6.7, and 6.8. Because one of the major concerns regarding these backup path computation methods is the amount of extra routing information that is required to be distributed through the network, this communication cost is also measured during the simulations and discussed in Sections 6.6, 6.7, and 6.8. Specifically, the measurement provided is the bandwidth consumed by routing data for the duration of the simulation. For example, with some of the methods tested (Section 6.5), the primary path information is sent along the pre-planned backup paths. Assume that the packet with all the required information has a size of d bytes and that the backup path consists of h hops (links). Then, the communication cost of sending this piece of routing information is $d * h$, because this packet continues to consume network resources until it reaches its destination.

Overall, the performance of the different methods is measured based on Chapter 5, specifically Table 20 and Table 22. The overhead of these methods is based on the amount of additional routing data that is circulating in the network.

6.5 Additional methods tested

In addition to the methods designed in Chapters 3 and 4, known methods—APLV Norm [48], CV [48], and BV+APV [49], are also simulated for comparison. A simple disjoint method, which uses the shortest disjoint paths as the designated backup path, is also tested as a comparison to the more complex methods. All of these methods compute backup paths that are intended to stay unreserved prior to the failure.

As mentioned, in addition to the difference in performance, the other major difference among the methods is the difference in the communication overhead described in Section 6.4. Table 24 provides an overview of the differences in terms of the communication overhead among AvoidPBO, TP, TPmax, APLV Norm, CV, and BV+APV. The expected communication overhead upper bounds are also provided in Table 24. It assumes that N is the number of nodes in the network, L is the number of links in the network, and F is the number of traffic flows in the network. A discussion of the different methods is also provided below.

Table 24. A comparison of the major difference in terms of communication overhead among the backup path computation methods.

Method	Communication overhead	Upper bound
AvoidPBO (Chapter 3)	• Primary and backup path information of each flow is communicated to all nodes in the network.	• *O (FL)*
TP (Section 4.4) TPmax (Section 4.5)	• Each node constructs two fixed-sized arrays, *PTP* and *BTP*, which are communicated to all other nodes in the network.	• *O (NL)*
APLV Norm [48]	• Primary and backup path information of each flow is sent along its backup path. • Condensed path information of each link is communicated to all nodes.	• *O (FN)* • *O (NL)*
CV [48]	• Primary and backup path information of each flow is sent along its backup path. • Condensed path information of each link is communicated to all nodes. (The condensed path information of each link is different from APLV Norm.)	• *O (FN)* • *O (NL)*
BV+APV [49]	• Primary and backup path information of each flow is sent along its primary path. • Information to compute backup path of each flow is sent along its primary path.	• *O (FN)* • *O (FN)*
Disjoint	• No extra information is needed.	• *O (1)*

With the disjoint method, the link weights are modified only based on the corresponding primary path. In this case, the only information needed is the topology and the primary path of the flow concerned. Thus, there is no communication overhead associated with the disjoint method. With the other methods, AvoidPBO, TP, TPmax, APLV Norm, CV, and BV+APV, the link weights are modified based on the extra routing information that the nodes distribute among themselves. With AvoidPBO (Chapter 3), primary and backup path routing information of every flow is distributed throughout the network. For TP and TPmax, Chapter 4 showed that the primary and backup path information is condensed into LxL arrays before being distributed throughout the network. APLV Norm and CV take an approach that is somewhat of a combination of AvoidPBO and TP/TPmax. With APLV Norm and CV, the complete primary and backup path information of a flow is distributed only along the backup path of that flow. The intermediate nodes along these backup paths condense the information they receive based on the design of each method. The condensed information is then distributed throughout the network. BV+APV puts another twist on the approach used by APLV Norm and CV. With BV+APV, primary and backup path routing information of a flow is distributed along the primary path of the flow. Each intermediate node also condenses the information received. The difference with BV+APV is that this condensed information is not broadcast over the network. Instead, when the backup path of a flow needs to be computed, another message is sent out by the source node along the primary path to collect the condensed information. Overall, TP and TPmax are the only methods expected to have constant communication overhead in a fixed network. The difference among AvoidPBO, APLV Norm, CV, and BV+APV is that AvoidPBO broadcasts per-flow information while the other three methods only send per-flow information along specific paths. Asymptotically, the overhead of AvoidPBO, APLV Norm, CV, and BV+APV are all bounded by *F*, the number of flows in the network. The actual difference in the overhead measured during the simulation is provided in Sections 6.6, 6.7, and 6.8.

6.6 Random graph with concentrated traffic

6.6.1 Topology, traffic, and failure cases

The topology used in this scenario is the same as the one first used in Sections 3.10 and 4.8.2, which is illustrated in Appendix A. It is a randomly generated graph of 20 nodes, each with a degree of four. For this simulation case, each simplex link has the ability to carry 370 units of traffic. Similar to Section 3.10, 100 unique sources-destination pairs, roughly 26% of all possible pairings, are randomly chosen to make up the sources and destinations of the first 100 flows. Their primary paths, the shortest paths computed using unit link weights on all links, are mapped out. To create conflicts among traffic, additional traffic is generated by randomly choosing source-destination pairs among these first 100 pairs, and these additional flows use the same primary paths as the first 100 flows. Thus, flow number 101 can be considered a duplicate, with the same source and destination and taking the same primary path, of one of the first 100 flows, and so on.

As described in Chapter 5, all the flows are assumed to require one unit of resource on each link of its primary path to satisfy their service quality, and the flows are mapped out such that the network can satisfy such service quality before the failure. In this particular setup, a total of 12000 traffic flows are generated. For this set of simulations, traffic loads of 2000, 4000, 6000, 8000, 10000, and 12000 flows are tested. Among the 80 simplex links in this setup, 40 simplex links, two out-going links from each node, are tested. Thus, 40 failure scenarios are tested, and all the scenarios cause at least one failed primary path in every traffic load test case. More details regarding this particular setup are provided in Appendix B.

6.6.2 Performance

Figures 12-20 show the simulation results under the setup described in Section 6.6.1. These results include a direct comparison to the use of two reserved paths, a look at the difference between pairing unreserved backup paths with strict recovery and with relaxed recovery, and also the overhead for communicating extra routing information. A discussion of these results is provided below, and Section 6.6.3 provides a summary of this particular simulation scenario.

As mentioned in Sections 5.2 and 5.3, the use of unreserved backup paths was first designed to prevent the waste of resources when two reserved paths are used. Though it is not always possible, unreserved backup paths are intended to provide the same service as two reserved paths. In other words, the hope for the unreserved paths is to achieve perfect recovery under the strict recovery mode, where resource reservation on the backup paths is attempted and successful. Figure 15 shows the percentage of failure cases tested where perfect recovery was achieved. As expected, perfect recovery is generally only achieved during very light traffic loads, the two lightest loads in this case. During medium loads, unreserved backup paths can provide perfect recovery for some link failures but not others. At heavy loads, perfect recovery is generally impossible, save for the odd one or two link failure locations. Thus, in this regard, unreserved backup paths simply cannot compete with the use of two reserved paths in most cases.

Because perfect recovery (Table 20) is an all-or-nothing measurement, it misses some link failure cases where many backup paths are successful. Figure 16 provides the recovery rate (Table 20), which shows the average percentage of backup paths that are successful under the strict recovery mode. For the better methods, a fairly high percentage of the backup paths are successful in the three lighter traffic loads tested. Specifically, in the first three loads, where traffic takes up roughly 50% or less of the available resources in the network, when a backup path is needed, 90% of the ones computed by AvoidPBO, TP, TPmax, and BV+APV are successful. Even the simple disjoint methods can provide good results in some cases, particularly in the first two cases. As expected, however, as the traffic load becomes heavier, unreserved backup paths paired with the strict recovery mode becomes increasingly ineffective. However, because a link failure only affects a limited number of flows, Figure 17 shows that, as a whole, the network continues to function quite well for the majority of the traffic. Thus, in the event of a single link failure, the use of unreserved backup paths with strict recovery is not catastrophic in general and may be a good compromise, particularly with respect to the large amounts of wasted resources incurred by reserving two paths. The drawback of unreserved backup paths under strict recovery is that it is fatal for some traffic.

Pairing unreserved backup paths with relaxed recovery (Sections 5.4 and 5.5) is expected to overcome this particular deficiency. When the primary paths fail after a link failure, the relaxed recovery forgoes resource reservation on the backup paths and so cannot guarantee to preserve service quality. Instead, traffic is always allowed to re-route over the planned backup paths. As a result, some traffic is expected to retain the same service quality as before, while some traffic is expected to be routed over links that are congested, as shown in Figure 18. The traffic routed over congested links[1] is expected to suffer some service degradation. While it is better to have zero flows suffer from congestion, it is also important that these congested flows do not suffer too much service downgrade. Figure 19 shows the amount of service degradation that the congested flows experience (Table 22). Taking Figure 18 and Figure 19 together, although TPmax causes more flows to suffer from congestion, these flows do not suffer too much, even in the heavier traffic loads. While some methods have flows that suffer no congestion and flows that suffer bad congestion, TPmax provides a more standard service for all the traffic concerned. For example, even in the heaviest traffic load, TPmax can provide at least 90% of the original service agreement to a flow, whereas the disjoint method provides less than 80% of the original service quality in the worst case. With TPmax, a flow either experiences the original service guarantee or has its service downgraded by about 10%. With the disjoint method, a flow might experience the original service guarantee, but another flow suffers a downgrade of about 20%. Figure 20 can be considered an average of these disparate types of service qualities experienced by the traffic as a whole. Because TPmax results in a higher number of congested flows, especially in the two heaviest network loads, on average, the average service quality provided by TPmax is second to TP and AvoidPBO in those cases. Overall, with relaxed recovery, all flows can remain connected, and the side effect is the possibility of small and maybe negligible service degradation, particularly for certain methods. Without

---

[1] These traffic flows are often called congested flows.

considering the overhead, if the objective is to minimize the service degradation experienced by a flow, then TPmax is the best option.

Based on Figures 12-17, an emerging trend is that at light traffic loads, almost all methods, even the simple disjoint method, can achieve good results, whereas poor performance is likely expected for all methods in heavy loads. The real difference in performance among the methods generally occurs in medium traffic loads, where there is obvious distinction between better methods and worse methods. This is not unexpected, as it is logical that when there is barely any traffic in the network, any method can work, whereas when there the network is completely loaded with traffic, likely nothing will improve matters. However, when there is room for manipulation, it is more obvious that some methods are better at managing traffic.

As mentioned in Table 1, within the category of unreserved backup paths, there is concern that good performance needs to be balanced with scalable overhead. Specifically, the fear is that good performance can only be achieved by using a large amount of extra routing information that is not practical. As outlined in Table 24, some algorithms, e.g., AvoidPBO, APLV Norm, CV, and BV+APV, that require flow-based information to be communicated over the network are not expected to be scalable or practical, because *F*, the number of flows (the amount of traffic), can increase indefinitely. During the simulation, the amount of network resources consumed by the distribution of extra routing information (Section 6.4) is collected and provided in Figure 21. (Since the disjoint method does not require additional routing information, there is no overhead involved and thus this method is not shown.) As expected, only the overhead of TP and TPmax stays constant, and the overhead incurred by other methods increases as the amount of traffic increases. Compared to the other methods, the overhead incurred by AvoidPBO can be considered off-the-chart, likely because AvoidPBO requires per-flow information to be broadcast. Figure 22 shows the overhead that focuses on TP, TPmax, APLV Norm, CV, and BV+APV to magnify the relationship among these methods. Although the overhead of APLV Norm, CV, and BV+APV is expected to grow as the amount of traffic increases, there is one twist with CV, where the overhead for the case of 12000 flows is smaller than that for the case of 10000 flows. The reason is that the majority of the extra routing information is communicated along the backup paths based on the CV algorithm design. According to the calculation described in Section 6.4, the amount of overhead is also dependent on how long a piece of routing information circulates in the network. If a backup path is extremely long, then not only is the information depicting the paths more lengthy, but the actual packet carrying the routing information also remains in the network for a long period of time, resulting in higher overhead. Thus, the actual amount of overhead incurred by the flow-based, path-based methods may not always match the number of flows in the network. In some cases, the overhead matches the sum of the path lengths of the paths along which routing information is distributed. Figure 23 shows the sum of the path lengths of the paths taken when communicating routing information, which matches the trend in Figure 22. However, because the sum of the path lengths generally matches the number of flows, the overhead is still expected to increase as the amount of traffic increases.

Based on the results of the overhead analysis, AvoidPBO should never be recommended for practical use. Together with the performance analysis, none of the flow-based algorithms should be recommended for practical use because the performance gain is relatively small compared to the overhead. Although AvoidPBO is unsuitable for practical purposes, it is still instrumental in the design of TP and TPmax, which turned out to be both efficient and effective.

The confidence intervals are not given in these results, and this convention is also used in Sections 6.7.2 and 6.8.2. This decision was made because performance varies greatly with different failure locations. As a result, lower performance generally does not occur under the same condition as higher performance. In the interest of providing cleaner plots for interpreting the overall behavior of unreserved backup paths, only the average is provided. Also, to emphasize the overall trends, solid lines are drawn through data points; they do not indicate actual values obtained through simulations.

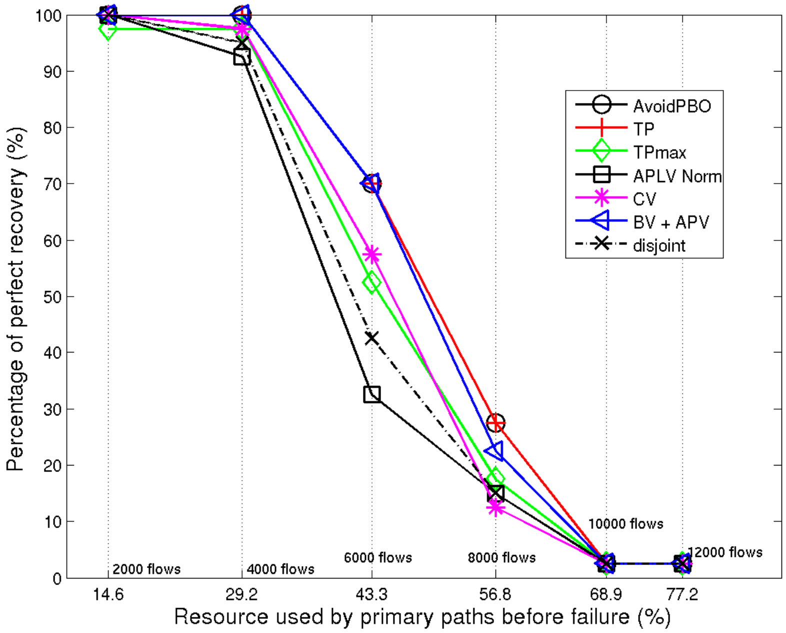


Figure 15. The percentage of failure cases in which unreserved backup paths can provide perfect recovery after a failure.

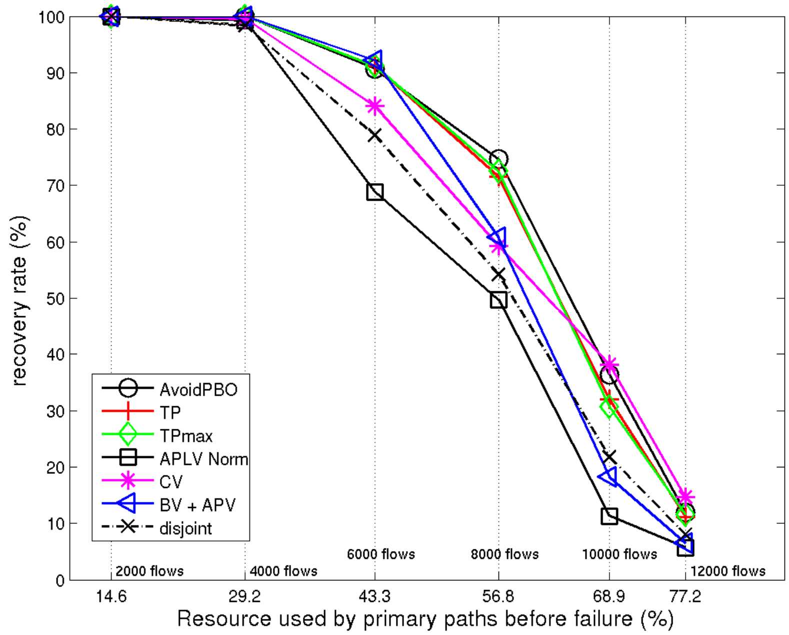


Figure 16. The percentage of flows that are successful after a failure, among those that require re-routing.

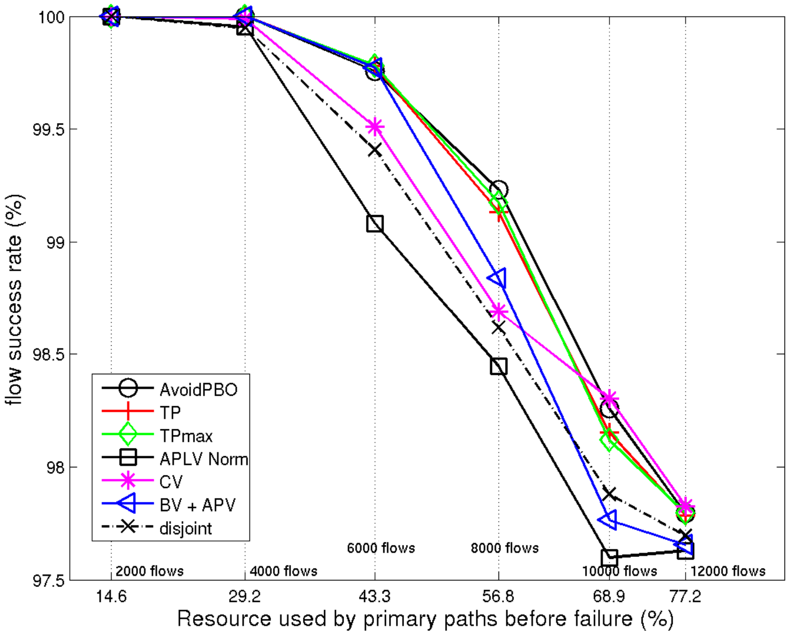


Figure 17. The percentage of flows that are successful after a failure among all the flows.

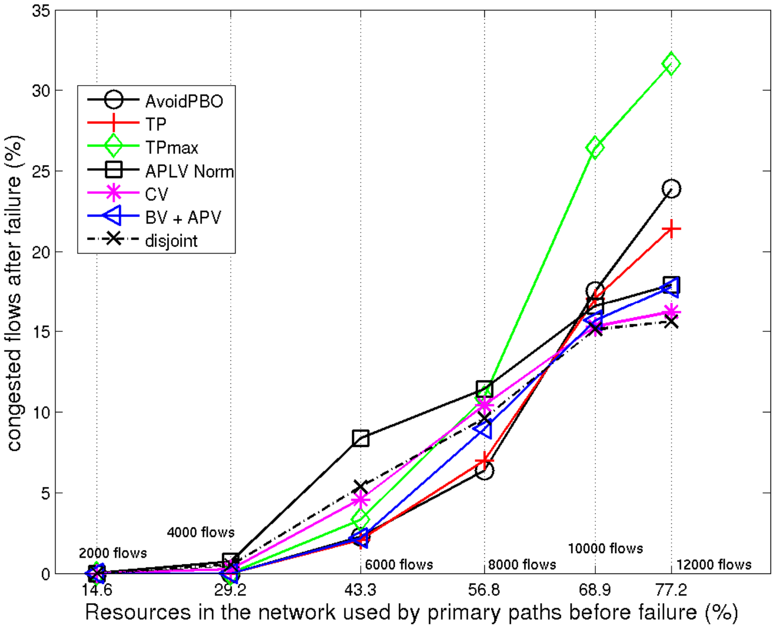


Figure 18. The percentage of flows that have at least one congested link along their forwarding path.

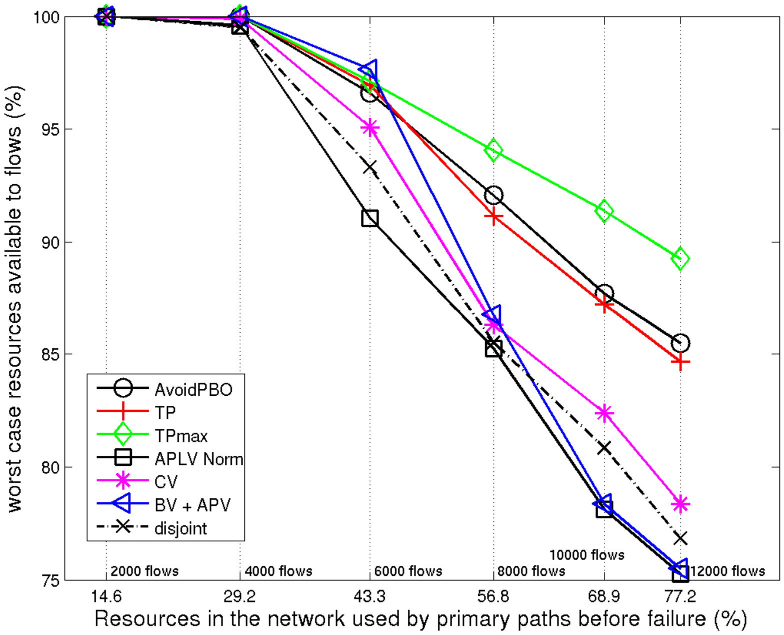


Figure 19. The resources available to flows that have at least one congested link along their forwarding path.

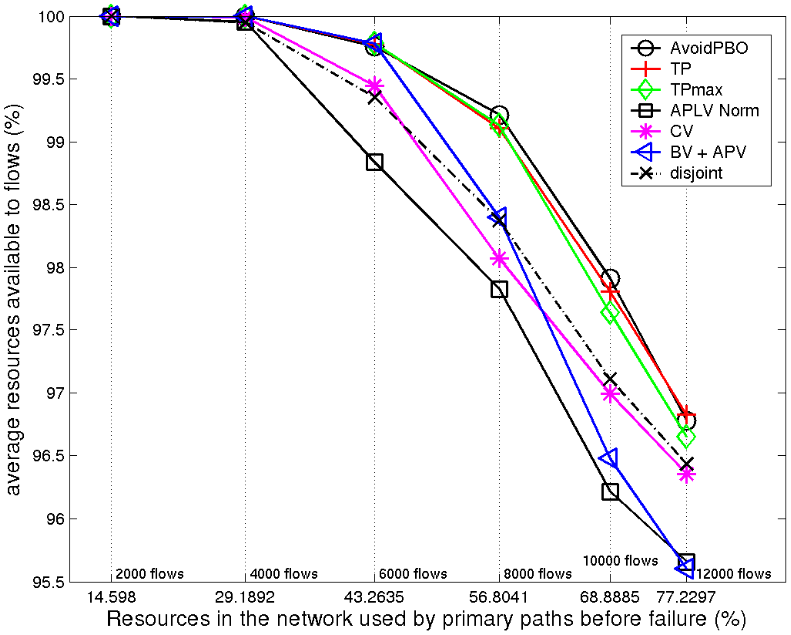


Figure 20. The average resource availability for all flows.

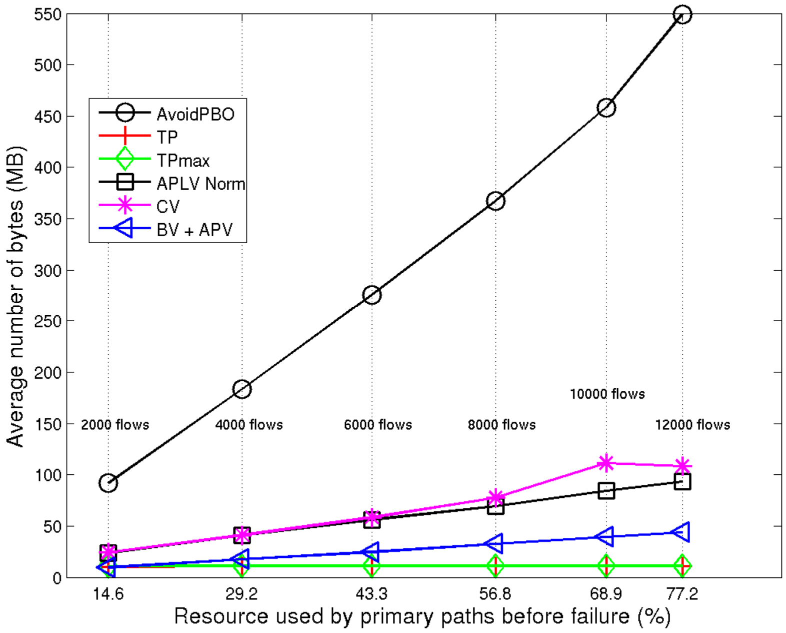


Figure 21. The communication overhead in terms of the amount of extra rouging information that is exchanged, as described in Section 6.4.

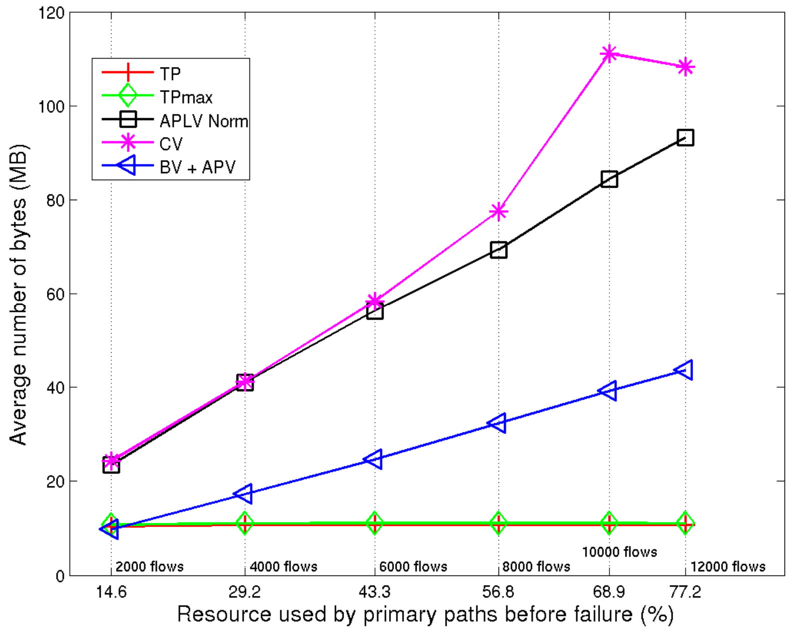


Figure 22. A magnified plot to show the communication overhead comparison among TP, TPmax, APLV Norm, CV, and BV+APV, without AvoidPBO.

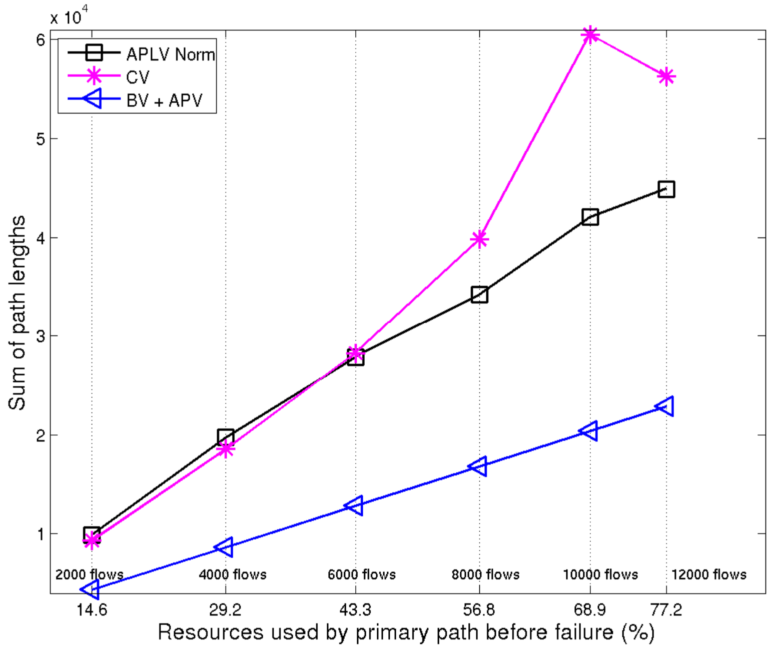


Figure 23. The sum of the path lengths of the paths to distribute flow-based rouging information.

6.6.3 Case summary

The general trend is a drop in performance as the traffic load increases. More specifically, the drop occurs when the network is generally half loaded with traffic prior to the failure. Strict recovery is only suitable in light network loads, before the network is more than half loaded, and only for certain methods, such as AvoidPBO, TP, TPmax and BV+APV, though in extremely light loads, even the disjoint method will suffice. When the network is more than half full, taking the overhead analysis into consideration, TP or TPmax should be recommended, and the unreserved backup paths chosen should be used under the relaxed recovery mode. Overall, this set of simulations shows that unreserved backup paths, especially when paired with relaxed recovery, can provide satisfactory performance even at heavy network loads. Furthermore, using TP or TPmax, good results can be obtained despite using less routing information.

6.7 Random graph with four extra links

6.7.1 Topology, traffic, and failure cases

For this set of simulations, the traffic and the failure cases are the same as those in Section 6.6. The topology is almost the same, except that four simplex links were randomly selected and added into the graph, as depicted in Appendix C. Like all the other links, each of these four links has a capacity of routing 370 traffic flows without compromising service quality. While the general trend in the results is expected to be similar to Section 6.6, the main reason for this setup is to examine how well different methods take advantage of the extra resources.

6.7.2 Performance

Figures 21-28 show the results from this simulation scenario. Figure 24, Figure 25, Figure 26, Figure 28, Figure 30, and Figure 31 are analogous to the graphs provided in Section 6.6.2. Figure 27 and Figure 29 provide comparisons between the scenarios simulated in Section 6.6 and this section. Based on the results in Section 6.6, AvoidPBO is deemed unsuitable, more so than other methods, for practical purposes due to its high overhead. Thus, results from AvoidPBO are not included in this section. A discussion of the simulation results is provided below, and Section 6.7.3 provides a summary of this set of simulations.

As mentioned, because of the minor change in the topology, the general trend of the performance is not expected to be different from Section 6.6. Figure 24, Figure 25, and Figure 26 show the performance of the unreserved backup paths under strict recovery that is similar to the results in Section 6.6.2: the backup paths only work well in light network loads. Because the difference is most evident in terms of recovery rate, Figure 27 provides a comparison of recovery rates between the scenarios in Section 6.6 and this section. Figure 27 shows the number of cases in which better recovery is obtained by adding four more links, among the 40 cases of link failures that are tested. In general, TP, TPmax, APLV Norm, CV, and BV+APV are better at taking advantage of the additional resources than the disjoint method. By considering how traffic is distributed in the network and how the distribution is expected to change in the event of a link failure, TP, TPmax, APLV Norm, CV, and BV+APV are better at managing the traffic to make the most out of the extra resources. In contrast, the disjoint method simply considers the topology and the corresponding primary path when a backup path is computed. Thus, extra resources may not even be considered if they do not happen to be along the shortest disjoint path.

Similarly, Figure 30 shows the resource availability experienced by congested flows, which is the representative performance measurement under the relaxed recovery mode. Similar to Section 6.6.2, many algorithms, particularly TPmax, can maintain good service even at heavy network loads. Comparing the results between the two topologies, Figure 29 shows the number of failure cases in which better service is obtained under the topology with four additional links. Similar to the comparison in Figure 27, the methods that consider how traffic is routed in the network are more consistent at taking advantage of the extra resources compared to the simple disjoint method.

For completeness, the overhead incurred by extra routing information distribution is shown in Figure 28. This is consistent with the upper bound analysis in Table 24, where the overhead for APLV Norm, CV, and BV+APV increases as the amount of traffic increases. As mentioned in Section 6.6.2, comparing Figure 30 and Figure 31, the overhead incurred by APLV Norm, CV, and BV+APV is also closely related to the paths on which the routing information is distributed.

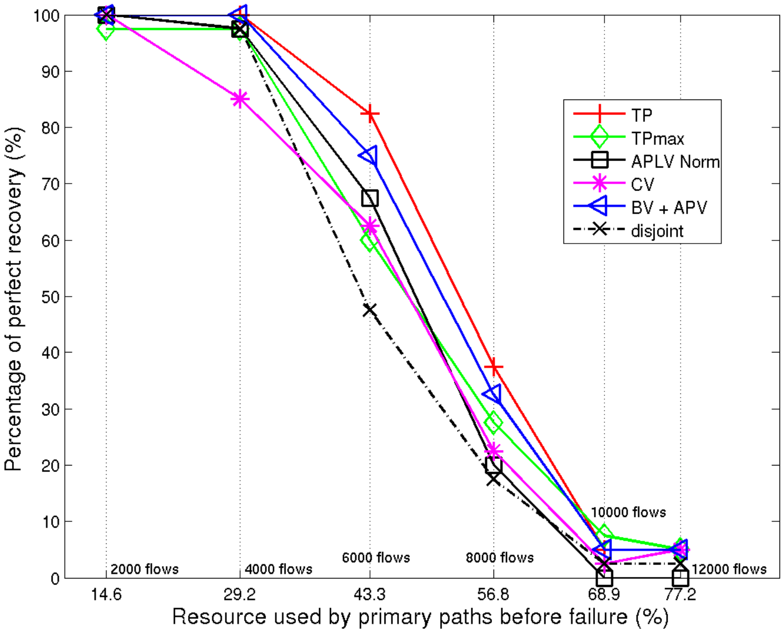


Figure 24. The percentage of failure cases with perfect recovery rate when there are four additional links.

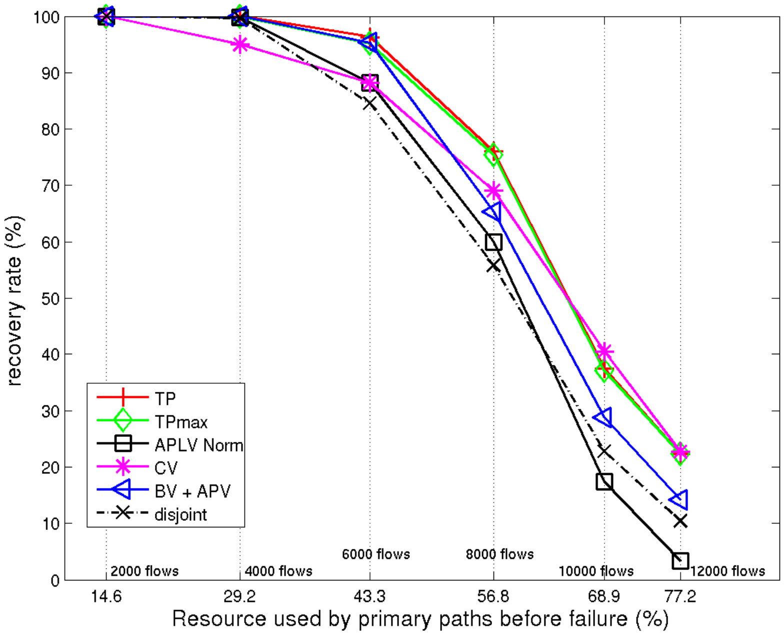


Figure 25. The percentage of successful backup paths when there are four additional links.

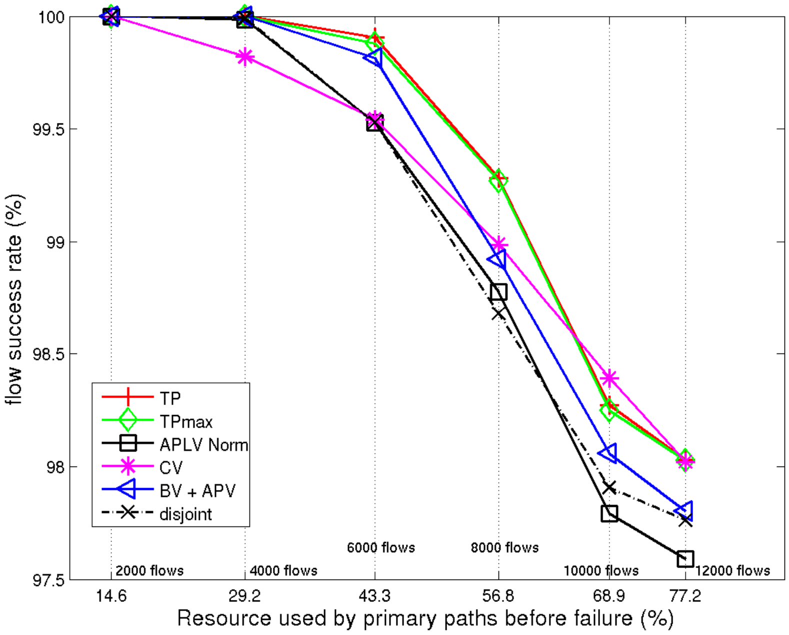


Figure 26. The percentage of successful flows when there are four additional links.

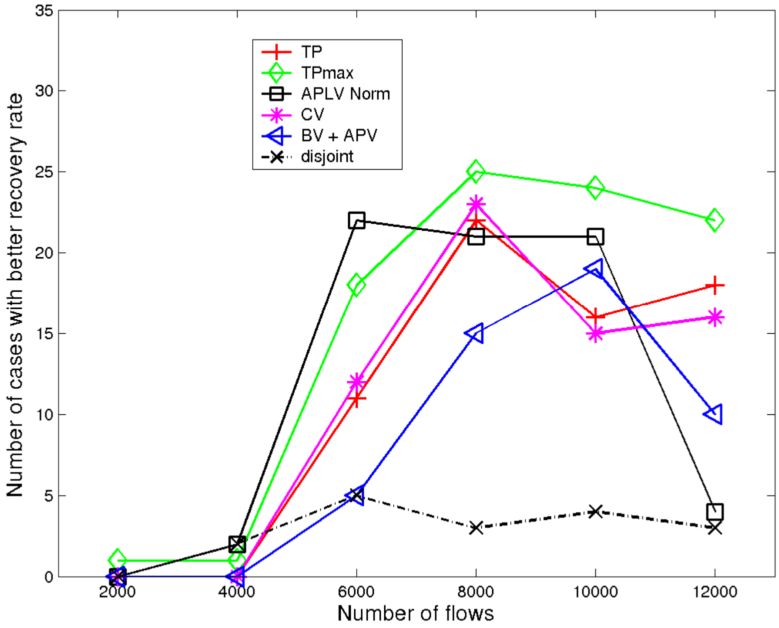


Figure 27. The number of failure cases with better recovery rates when there are four additional links compared to the original topology.

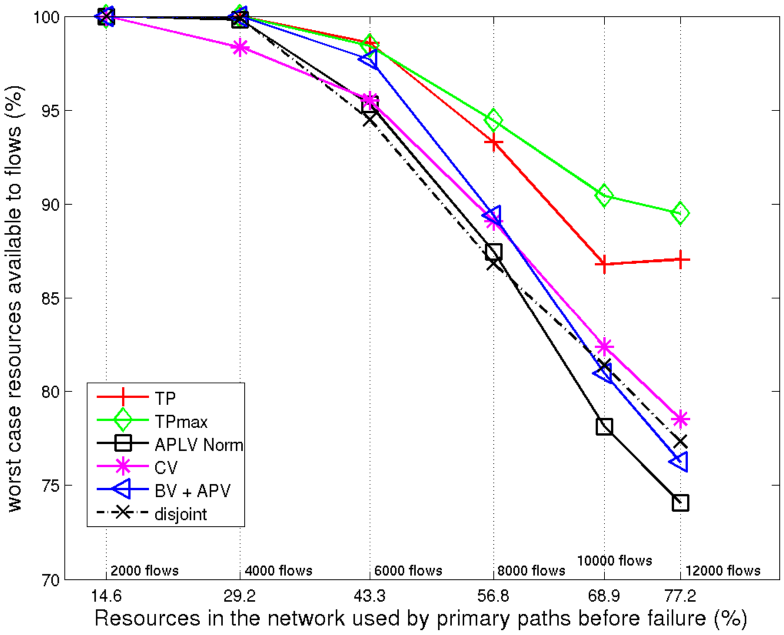


Figure 28. The resources available to flows using at least one congested link along their forwarding paths when there are four additional links.

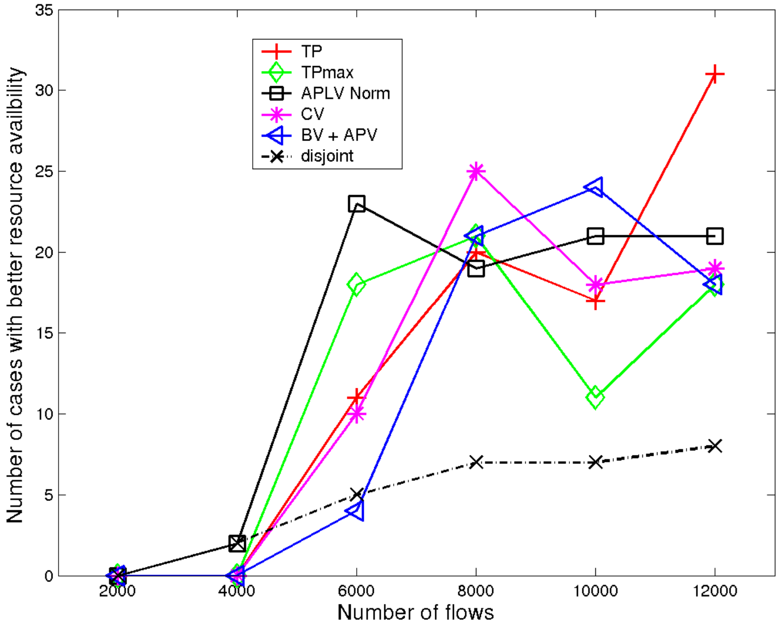


Figure 29. The number of cases with better service quality when there are four additional links compared to the original topology.

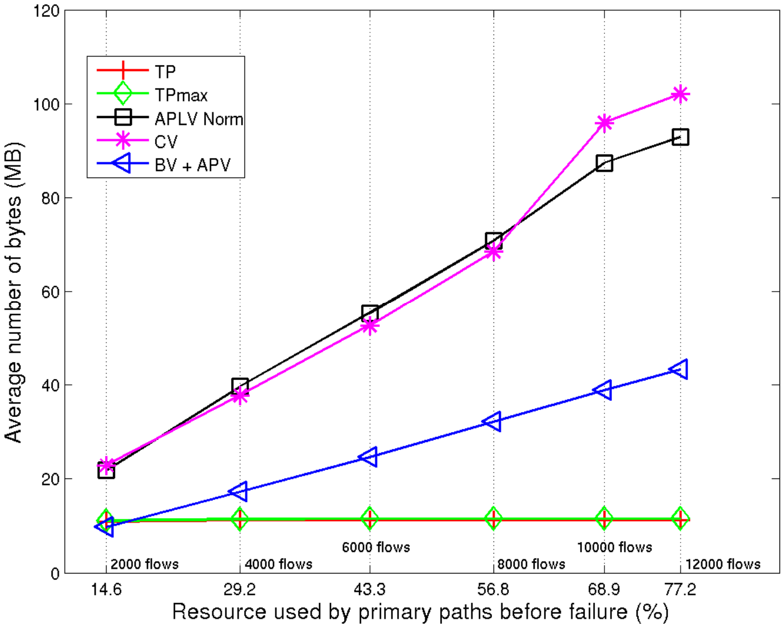


Figure 30. The overhead incurred to communicate extra routing information when there are four additional links.

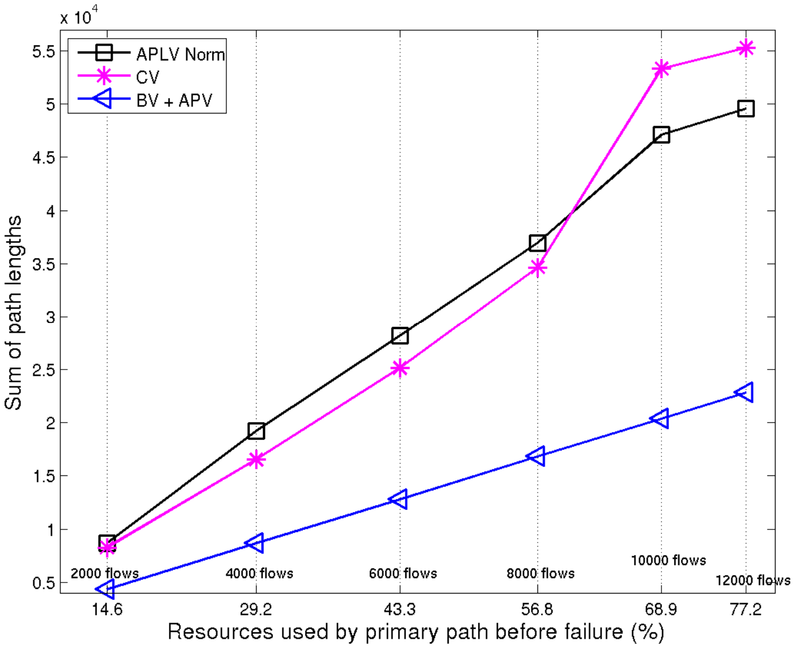


Figure 31. The sum of path lengths of the paths to distribute flow-based information when there are four additional links.

6.7.3 Case summary

As expected, additional resources, in the form of extra links, result in improved performance in general. The general trend is that the improvements obtained by the disjoint method are limited, likely because it only considers the topology. In contrast, methods such as TP and APLV Norm which consider how traffic flows related to each other when planning backup paths can make more adjustments and tailor the solution to existing network conditions, wherever the available resources happen to be. Overall, this test case provides a positive outlook regarding the expected performance on topologies and scenarios not tested in this thesis using methods such as TP and APLV Norm. Such methods are expected to adapt to the current conditions and make good use of the available resources.

6.8 Random graph with random traffic

6.8.1 Topology, traffic, and failure cases

The topology used in this set of simulations is identical to what is used in Section 6.6. The difference in this case is the initial traffic. Unlike Sections 6.6 and 6.7, the initial traffic is not concentrated between 100 pairs of source-destination nodes. Instead, the initial traffic is randomly selected between any two nodes for the entire 12000 traffic flows. Like Sections 6.6 and 6.7, for each flow, a source and destination node pair is first chosen, and then the shortest path between these two nodes are mapped out and reserved as the primary path of the flow. Because the source and destination pairs are not limited to the first 100 pairings like in Sections 6.6 and 6.7, the initial traffic is expected to be more spread out. Consequently, the resources that are not initially reserved are also more evenly spread out over the network.

6.8.2 Performance

Because the overhead incurred by AvoidPBO is much too high compared to other methods, results from AvoidPBO are again not included in this section. Figure 32 provides the recovery rate (Table 20), the success rate of the activated backup paths, in the strict recovery mode (Section 5.2). Like before, the general trend is that the recovery rate decreases dramatically when the initial traffic consumes roughly more than 50% of the network resources. Compared to the traffic condition in Section 6.6, Figure 33 shows that the performance with less concentrated traffic is better for all traffic loads tested. Despite having the same number of traffic flows, which actually consumes a little more resources initially than the concentrated traffic setup in general, as shown in Figure 34, the random traffic setup still results in better performance after a failure. While the dramatic drop in recovery rate begins in the case with 6000 flows when the network is slightly less than half full in Figure 16, Figure 32 shows that the dramatic drop with random traffic begins after the network is half full in the case with 8000 flows. Likewise, Figure 35 provides the performance trend under relaxed recovery (Section 5.4), and Figure 36 also shows that better performance is obtained in the random traffic setup. Looking at how the initial traffic is distributed in the network, Figure 37 shows that the initial traffic in the setup in Section 6.6 is indeed more concentrated, generally leaving more links with very few resources, less than 10% of the total capacity, for re-routing. With so few free resources available, these links are essentially rendered useless, or become the weak link and bottleneck at the very least, because they are unlikely to handle any re-routing successfully. Figure 38 further shows that, with the more concentrated traffic distribution in Section 6.6, there are fewer links with a medium (40-70%) amount of resources available for re-routing. Together, a smaller number of links with adequate resources and a larger number of links with very limited resources likely contribute to the comparatively lower performance in the concentrated traffic setup.

Overall, the results from this section and Sections 6.6.2 and 6.7.2 all show that the turning point in performance is still at the point where a network is roughly 50% loaded with initial traffic. If the initial traffic is more evenly distributed, or from the opposite perspective, if the free, available resources in the network are more evenly distributed, then the performance in both recovery modes is expected to be better than cases where traffic or free resources are concentrated in a few specific areas.

For completeness, Figure 39 provides the communication overhead incurred during this simulation set. As expected from the analysis in Table 24, only TP and TPmax use constant overhead regardless of the amount of traffic in the network. When the amount of traffic F is much larger than the number of nodes (N) and links (L) in the network ($F >> L$ and $F >> N$), the overhead incurred by TP and TPmax is expected to be the smallest among the methods tested, excluding the disjoint method.

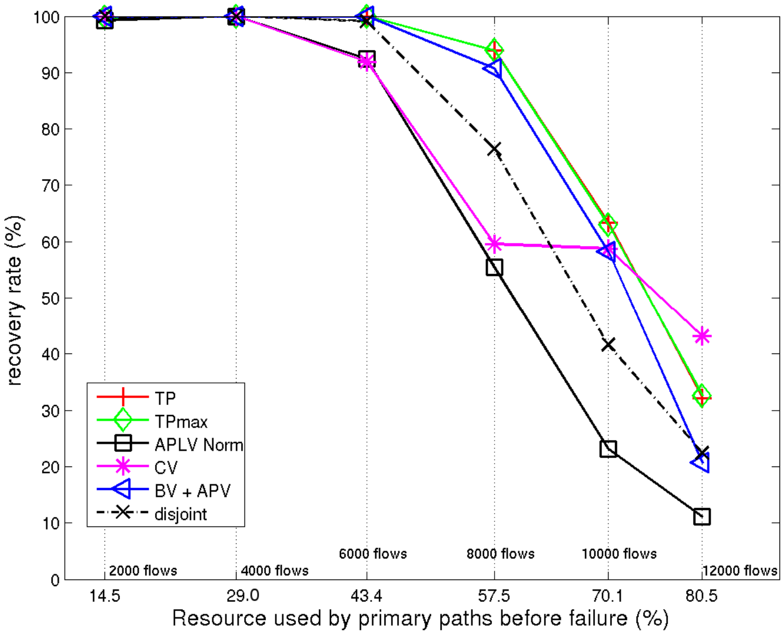


Figure 32. The percentage of re-routed flows that are successful after a failure when the initial traffic is not concentrated, among those that require re-routing.

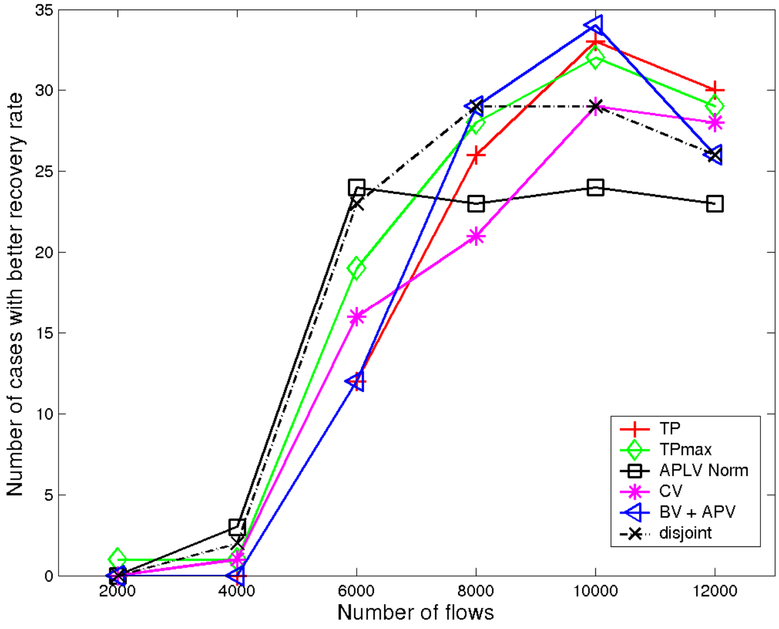


Figure 33. The number of failure cases with better recovery rates when the initial traffic is not concentrated.

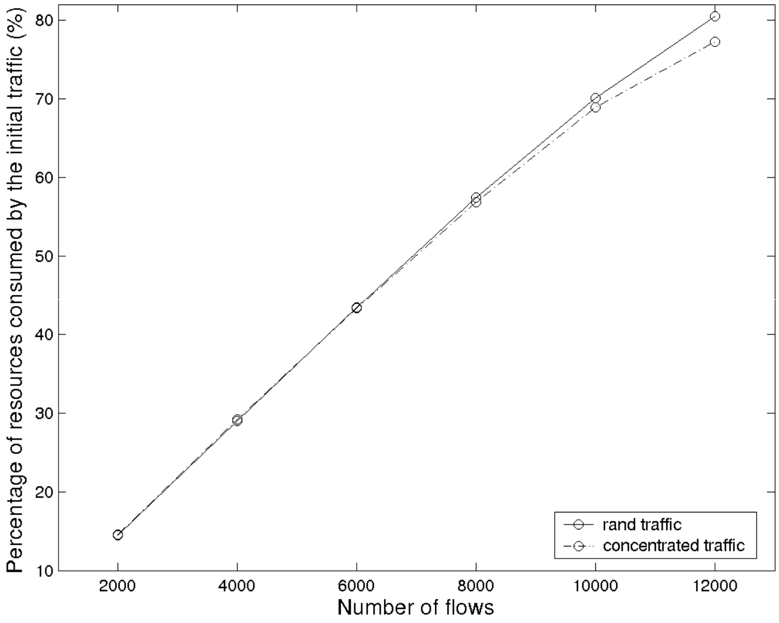


Figure 34. The percentage of resources used by the initial traffic under the condition described in Section 6.8.1.

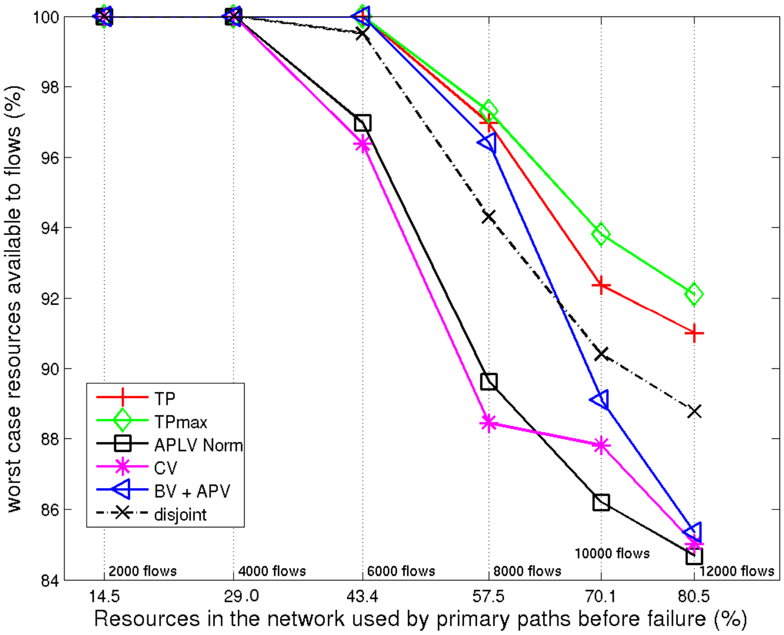


Figure 35. The resources available to flows that have at least one congested link along their forwarding path when the initial traffic is not concentrated.

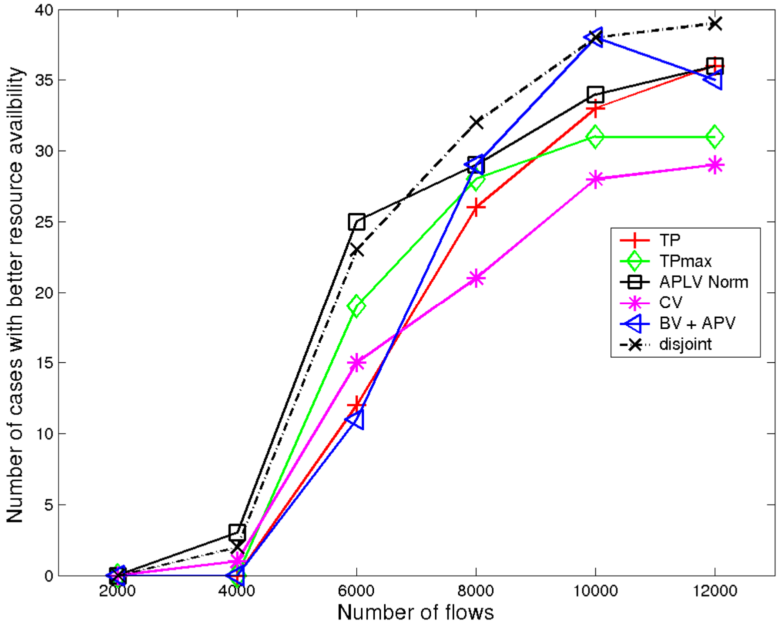


Figure 36. The number of cases with better service quality when the initial traffic is not concentrated.

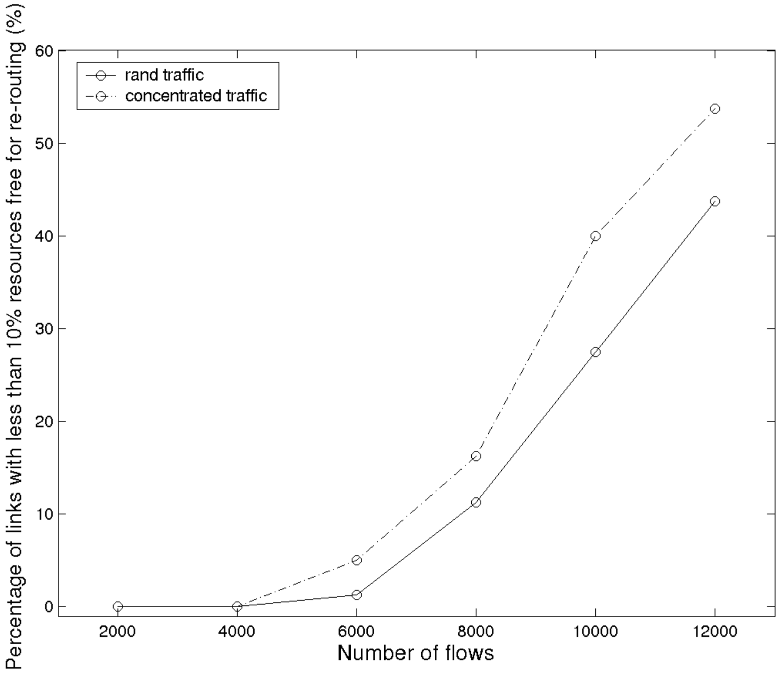


Figure 37. The percentage of links with less than 10% of free resources available for re-routing.

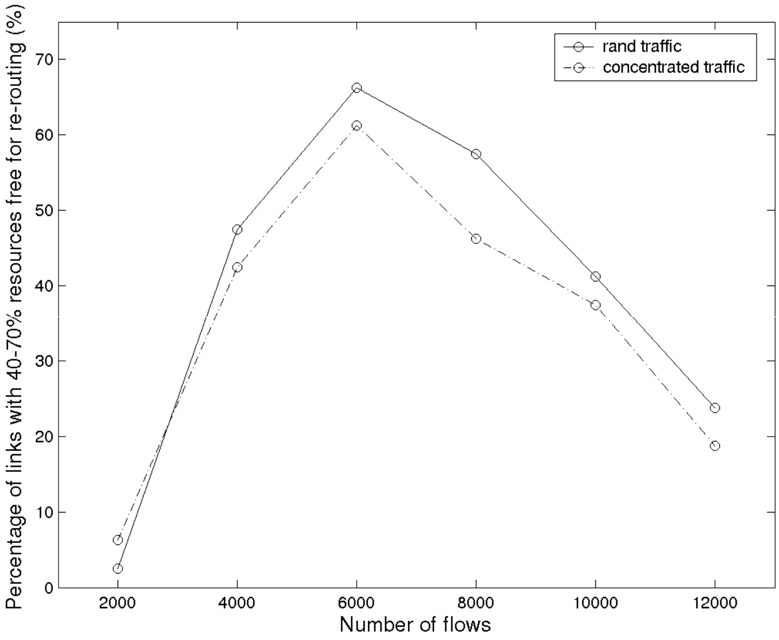


Figure 38. The percentage of links with 40-70% of free resources available for re-routing.

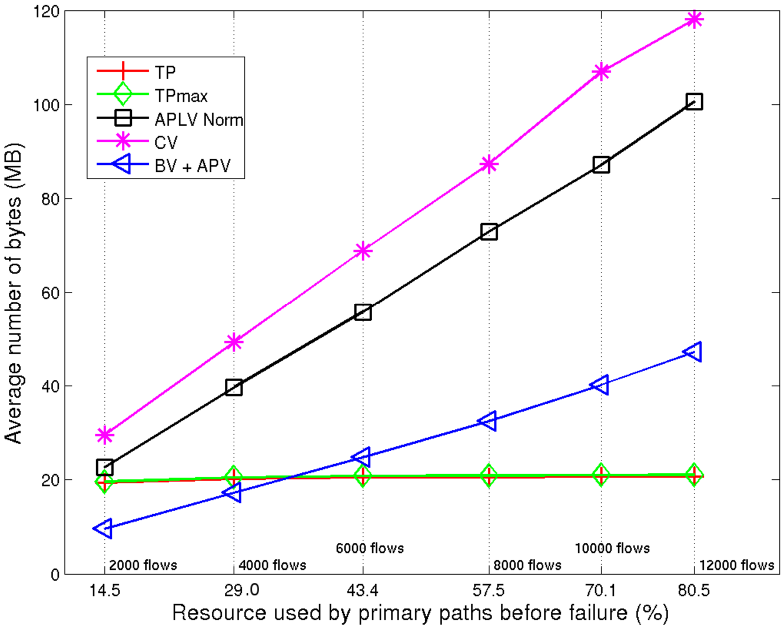


Figure 39. The overhead incurred to communicate extra routing information when the initial traffic is not concentrated.

6.8.3 Case summary

The general trend for this simulation set is the same as those in Sections 6.6 and 6.7. The twist in this simulation set is that, by keeping the initial traffic more evenly distributed, good performance can be obtained for more cases in the strict recovery mode. Performance in the relaxed recovery mode is also improved.

6.9 Chapter summary

Tests on the methods designed in Chapters 3 and 4, AvoidPBO, TP, and TP, are performed in this chapter to evaluate their performance, particularly in the face of changing topologies and traffic conditions. Additionally, the trade-off between performance and overhead is also examined. For comparison, several known methods in the same solution category are also tested. Due to its high overhead and unsuitability, results from AvoidPBO are only included and discussed in the first simulation set (Section 6.6), which shows that AvoidPBO conforms to the general trend, except for the overhead. Although any performance gain by AvoidPBO is dismissed in Sections 6.7 and 6.8, AvoidPBO still played in important role in the design of TP and TPmax.

In general, there is a division between light and heavy network nodes, and the division occurs roughly at the point where the network is half loaded with traffic. The strict recovery mode only works well in light traffic loads. Under the relaxed recovery mode, while more acceptable performance can be obtained by all methods, TPmax particularly performs well, with less dramatic decline in performance as the traffic load increases. Furthermore, TPmax obtains these results despite requiring only constant overhead in a fixed network, which is significantly lower overhead than flow-based methods.

In terms of the performance in a changing network, slightly better performance can be obtained if the free resources and traffic are evenly distributed. Although the methods that consider how traffic is routed incur more overhead, they are shown to adapt to different traffic conditions and network topologies. Such methods can make better use of the available resources by managing the relations among traffic flows and the paths they take. Overall, based on the performance, overhead, and adaptability, TPmax, especially when it is paired with relaxed recovery, is the best option when unreserved backup paths are used to improve service quality.

7 Conclusion and Future Work

7.1 Summary

Traditionally, a single path with enough resources reserved is used to provide the high QoS requested by a user. Although this path has enough resources reserved to satisfy the service quality of a flow, it suffers from single link failures. When a flow depends on only a single path, it can easily be disconnected in the event of a link failure in the network. A popular solution to handle flows in the event of a link failure is to use two reserved paths per flow, both a reserved primary path and a reserved backup path. Because the backup paths are reserved, this approach can guarantee the same service quality both before and after a link failure for all flows. However, since backup paths are rarely activated, the resources reserved solely for these backup paths are wasted most of the time.

The approach taken in this thesis is a less expensive approach that pairs a reserved primary path with an initially unreserved backup path. Several algorithms are designed to compute unreserved backup paths, each with different performance results and overhead requirements. Using different path computation algorithms and different techniques to utilize the backup paths once they are needed, competitive performance relative to that of reserving two paths can be achieved, thereby providing acceptable service quality and reliability to many applications inexpensively after a link failure.

Based on existing work, a basic algorithm, AvoidPBO, to compute backup paths that are intended to be unreserved prior to the failure is designed. To be successful, unreserved backup paths need to be chosen so that the activated backup paths do not fight over the same resources. While the backup paths are eventually calculated by Dijkstra's algorithm, AvoidPBO provides a way to determine the links that are expected to be under the most contention, and thus should not be chosen if possible. Its framework differs from existing methods in that it does not require intermediate nodes to construct/calculate any routing information. The source nodes are responsible for communicating useful routing information into the network, and the intermediate nodes are only required to exchange the routing information. As a basic algorithm, the useful information AvoidPBO source nodes send out into the network is the raw path information of their flows. While this is very costly, on the order of the number of flows in the network, by pushing all responsibility back to the source nodes, less expensive methods such as TP and TPmax could be derived from this framework. In contrast, methods that require intermediate nodes to help compute essential routing information always require raw path information of flows to be sent to the intermediate nodes. Although such methods may incur less overhead than AvoidPBO by reducing the number of nodes that receive per-flow information, their reduction in overhead cannot compete with methods such as TP and TPmax that only require topology-based information. Thus, even though AvoidPBO is likely unrealizable in practice, its design is instrumental in other practical and effective approaches.

TP and TPmax are two algorithms that compute unreserved backup paths using lower overhead, and they are derived from AvoidPBO. Based on how AvoidPBO uses the raw path routing information to determine undesirable links, a method to condense routing information to two fixed size arrays, *PTP* and *BTP*, is designed. This method condenses raw routing information such that the end result is useful and its size remains constant regardless of the amount of traffic in the network. In a fixed network, this method incurs constant overhead, where each source node sends out *PTP* and *BTP*, rather than the flow-based overhead required by AvoidPBO and other known methods. TP and TPmax are two methods developed to use *PTP* and *BTP* to compute backup paths. Their performance is generally second only to AvoidPBO while using asymptotically less routing information. AvoidPBO, TP, and TPmax as a group, also provides better performance in the majority of the test case, than the existing methods that also compute unreserved backup paths to provide reliable QoS. Thus, the design decision to push computation back to the source nodes allows significant overhead reduction and still provides a way to choose effective backup paths.

Merely carefully choosing unreserved backup paths is not always enough to provide acceptable service after a failure, especially when the traffic load is heavy. When the traffic load is heavy, most of the resources in the network are already in use and very few resources will be available. Under this circumstance, the traditional way a backup path is used, which ultimately requires resource reservation to be performed on the backup paths, results in poor performance for the majority of the traffic loads. Thus, while unreserved backup paths are less expensive than reserved backup paths and are simpler to set up prior to the failure, getting the best results out of unreserved backup paths is not as straight forward. If the original service quality is to be maintained, then some flows will be sacrificed and become disconnected (the strict recovery mode in Section 5.2). The alternative is to allow all flows to remain connected through the remaining resources, which may result in lower service quality (the relaxed recovery mode in Section 5.4). The best course of action to take depends on the network conditions.

With strict recovery, the general trend is that good performance can be provided by all the methods tested when the network is fairly light. Even the simple disjoint backup paths work well when the network load is extremely light. In other relative light network loads, depending on the network setup, recovery by all flows may not be possible under some failure scenarios. However, even in these situations, because the traffic load is relatively light, very few flows are sacrificed and become disconnected. As the network load becomes heavier, performances of different backup path computation methods become more varied. At the same time, unreserved backup paths result in more and more disconnected flows, and thus become less and less useful. Overall, with strict recovery, once the network is half full, the percentage of successful recovery decreases dramatically, down to less than 10% success rate at worst.

When the network is more than half full, the recourse is to mitigate the effects of a link failure using the relaxed recovery mode. This sacrifices service quality to allow all flows to retain connection. The overall trend is still a drop in performance as the traffic load increases. Unlike the strict recovery, in the worst case, a flow can still expect to experience about 75% of the original service quality—a 25% drop in service quality. With TPmax, the service quality never goes below roughly 90%, a drop of at most 10%

service quality, even when the network is more than three quarters full. Thus, with relaxed recovery, all flows can remain connected, and the side effect is the possibility of small and possibly negligible service degradation when TPmax is used. As the network load increases, TP and TPmax perform better than other methods despite using less routing information. Relative to other methods, the service degradation from TP and TPmax is less dramatic as the traffic load increases.

This thesis developed several techniques that can be combined to provide inexpensive, scalable, and satisfactory service to flows that are serviced by single dedicated paths, in the event of a single link failure in the network. Using unreserved backup paths to mitigate the damage of a single link failure is much less expensive than using two reserved paths, but is not as straightforward. In many cases, the backup paths need to be carefully chosen and also used differently under different circumstances for the best performance. When the traffic load is light, service quality does not need to be sacrificed, whereas service quality has to be sacrificed to maintain flow connection when the traffic load is heavy. Strict recovery can be used at light networks loads and relaxed recovery should be used in heavy network loads. Considering both the performance of the backup paths and the overhead incurred, TP paired with static recovery results in the best performance in light network loads. TPmax paired with relaxed recovery results in the best performance in heavy network loads. Because TPmax uses the same routing information as TP, it is relatively easy to switch from TP to TPmax as the traffic load increases to improve the service quality. Thus, using more sophisticated backup path computation methods and various different techniques, acceptable service can be provided to overcome the effects of a single link failure, even without reserving resources on the backup paths. At the same time, the overhead is scalable and controlled.

7.2 Contributions to knowledge

Most existing research generally assumes that providing re-routing at high service quality in the event of a single point failure in the network requires reservation of two paths (Section 2.5) in advance, which is extremely costly and wasteful. In an effort to overcome this drawback, this thesis sets out to examine the use of unreserved paths to provide more reliability to improve QoS, particularly for the case of single link failures. A family of algorithms to compute unreserved backup paths is designed. Based on an initial basic framework, resource contention behaviors are categorized to ultimately design scalable and effective backup path computation methods. Performance evaluations show that these methods are sensitive to both the traffic and the topology to gain good recovery results. By accepting the limitations of unreserved backup paths, additional techniques that can be applied during the recovery process based on different network conditions are also developed to compensate for the fact that no resource provisioning is performed before a failure. Instead of forcing unreserved backup paths into the role of reserved backup paths, a different procedure, relaxed recovery, which makes better use of these unreserved backup paths, is employed. Simulations are performed, and trends and behaviors are studied so that the traffic load-recovery procedure-algorithm combinations that yield the best results are identified. The performance analysis shows that these unreserved backup paths are likely to be feasible in many cases. If they are well-planned, using different techniques under different conditions, unreserved backup paths, while not perfect, can provide inexpensive satisfactory recovery from a link failure.

In particular, two algorithms, TP and TPmax, are not only scalable, but also yield extremely competitive results. Using TP and TPmax to carefully choose backup paths, even if they are unreserved prior to the failure, they can still provide satisfactory protection against single link failures. For a service provider, in extremely light network loads (at most 20% full), nothing needs to be done in advance to overcome a link failure. Simply re-routing flows over a disjoint path will suffice. When the network is relatively lightly loaded but has enough traffic in it (20% to 50% full), unreserved backup paths will need to be more carefully calculated. In these situations, TP, using scalable routing overhead, can provide performance nearly as good as using two reserved paths which guarantees perfect recovery. When the network is more than 50% loaded, TPmax paired with relaxed recovery should be used, and the expected service downgrade is at most 10%. Under these circumstances, an application that can tolerate 10% of service degradation can be satisfied with this service. With a relatively small amount of degradation, it is likely that software correction or more sophisticated data encoding can be done at the end users or at the servers so that the small drop in service quality is all but indistinguishable to humans. To further improve upon network QoS, a service provider can be more mindful regarding the placement of primary paths by spreading them out more evenly. This way, the free resources can be more effective when re-routing is necessary.

7.3 Future work

As mentioned in Sections 3.9 and 4.6, while AvoidPBO and TP/TPmax as defined in Chapters 3 and 4 assume that the traffic of interest has the same service quality requirement, they can certainly be extended to accommodate more varieties of traffic and service guarantees. The ranking of traffic and nodes in Sections 3.7.1, 4.4.2, and 4.5.2 should also be examined further for any negative consequences of ranking. As mentioned in Section 3.7.1, AvoidPBO is currently defined to use the starting time of traffic flows to determine the ranking and priority of backup path traffic. A similar decision is made for TP and TPmax in Sections 4.4.2 and 4.5.2. Exactly how fair these decisions are should be examined further, and other options of ranking should also be explored. In addition to these extensions, by separating the routing information into how resources are used by primary and backup path, the design of AvoidPBO, TP, and TPmax can potentially adjust to even more non-uniform and disparate network topology and traffic models. For example, when primary path traffic is completely concentrated in a small region of the network, information regarding backup path traffic is likely unnecessary because the majority of the traffic will remain in that small region even when there is link failure in the network. Perhaps a better algorithm should first examine the disparity of traffic encoded in its routing information before sending the information out. By not sending out the negligible backup path traffic that can potentially confuse algorithms, not only is the condition in the network more accurately illustrated, there is also the likelihood of a reduction in the cost of broadcasting the routing information. In addition to enhancing performance, improvements on different failure modes, such as single node failures, should also be considered in the future. Alternatively, because complete failures are the extreme cases in a large spectrum of failure models, solutions for complete failures could possibly be adapted to ultimately provide fault tolerance for a range of failures, such as a non-negligible decrease in network capability. Sensitivity tests for other details in the proposed algorithms such as the input parameter *factor* can also be performed for deeper understanding of the behaviors of the algorithms..

As mentioned in Section 6.3, the design of the simulations presented in this thesis aims to explore the potential of unreserved backup paths and a guideline on how to use them. Just like there is a difference between light traffic and heavy traffic, it is likely the case that special handing is required for different types of traffic, such as audio or video traffic.

In terms of the directions of future research, QoS routing over multiple domains [82] is becoming increasingly important as multiple service providers and multiple types of services are increasingly becoming accessible. In such an environment, some issues such as the trade-off between the amount of routing information and performance will become more serious as the routing information will need to be distributed over a wider area. Furthermore, solutions also need to be provided to guard against security issues such as the misuse of crucial routing information. The question of how to recover from malicious or uncooperative servers may come to the forefront in the future when multiple domains are expected to work together while remaining conscientious of their service agreements to their own customers.

References

[1] Andersen, D., Balakrishnan, H., Kaashoek, F., and Morris, F. Resilient Overlay Networks. ACM Symposium on Operating Systems Principles (SOSP), Oct. 2001.

[2] Anderson, J., Doshi, B. T., Dravida, S., and Harshavardhana, P. Fast Restoration of ATM Networks. IEEE Journal on Selected Areas in Communications, Vol. 12, No. 1, pp. 128-138, Jan .1994.

[3] Andersson, L., Doolan, P., Feldman, N., Fredette, A., and Thomas, B. LDP Specification. RFC 3036, Jan. 2001.

[4] Apostolopoulos, G., Guerin, R., and Kamat, S. Implementation and Performance Measurements of QoS Routing Extensions to OSPF. IEEE Infocom, Mar. 1999.

[5] Apostolopoulos, G., Williams, D., Kamat, S., Guerin, R., Orda, A., and Przygienda, T. QoS Routing Mechanisms and OSPF Extensions. RFC 2676, Aug. 1999.

[6] Aukia, P., Kodialam, M., Koppol, P.V.N., Lakshman, T.V., Sarin, H., and Suter, B. RATES: A Server for MPLS Traffic Engineering. IEEE Network, Vol. 14, No. 2, pp. 34-41, Mar/Apr. 2000.

[7] Awduch, D., Berger, L., Gan, D., Li, T., Srinivasan, V., and Swallow, G. RSVP-TE: Extensions to RSVP for LSP Tunnels. RFC 3209, Dec. 2001.

[8] Awduche, D., Malcolm, J., Agogbua, J., O'Dell, M., and McManus, J. Requirements for Traffic Engineering over MPLS. RFC 2702, Sep. 1999.

[9] Banerjea, A. Fault Recovery for Guaranteed Performance Communications Connections. IEEE/ACM Transactions on Networking, Vol. 7, No. 5, pp. 653-668, Oct. 1999.

[10] Banerjea, A. Simulation Study of the Capacity Effects of Dispersity Routing for Fault Tolerant Realtime Channels. ACM SIGCOMM Computer Communication Review, Vol. 26, No. 4, pp. 194-205, Oct. 1996.

[11] Banerjea, A., Parris, C., and Ferrari, D. Recovering Guaranteed Performance Service Connections from Single and Multiple Faults. Technical report TR-93-066, University of California, Berkeley, 1993.

[12] Bejerano, Y., Breitbart, Y., Orda, A., Rastogi, R., and Sprintson, A. Algorithms for computing QoS Paths with Restoration. IEEE Infocom, Mar. 2003.

[13] Blake, S., Black, D., Carlson, M., Davies, E., Wang Z., and Weiss, W. An Architecture for Differentiated Service. RFC 2475, Dec. 1998.

[14] Braden, R., Clark, D., and Shenker, S. Integrated Services in the Internet Architecture: an Overview. RFC 1633, Jun. 1994.

[15] Chen, I. and Ito, M. R. A Study of Unreserved Backup Paths for Reliable QoS under Single Link Failure. IEEE International Conference on Computer Communications and Networks (ICCCN), Aug. 2008.

[16] Chen, I. and Ito, M. R. Relaxed Failure Recovery Process for Reliable Qualtiy-of-Service using Unreserved Backup Paths. IEEE International Conference on Advanced Information Networking and Applications (AINA), May 2007.

[17] Chen, I. and Ito, M. R. Reliable Strict Quality-of-Service with Unreserved Backup Paths. IEEE International Conference on Advanced Information Networking and Applications (AINA), May 2007.

[18] Chen, I. and Ito, M. R. Scalable Communication for High Performance and Inexpensive Reliable QoS using Relaxed Recovery. IEEE International Symposium on Network Computing and Applications (NCA), Jul. 2007.

[19] Chen, I. and Ito, M. R. Scalable Communication for High Performance and Inexpensive Reliable Strict QoS. International Conference on New Technologies, Mobility and Security (NTMS), May 2007.

[20] Chen, S. and Nahrstedt, K. An Overview of Quality-of-Service Routing for the Next Generation High-Speed Networks: Problems and Solutions. IEEE Network Magazine, Special Issue on Transmission and Distribution of Digital Video, Vol. 12, No. 6, pp. 64-79, November-December 1998.

[21] Clark, D. D. The Design Philosophy of the DARPA Internet Protocols. ACM SIGCOMM ’88: Computer Communication Review, Vol. 18, No. 4, pp. 106-114, Aug. 1988.

[22] Coltun, R. The OSPF Opaque LSA Option. RFC 2370, Jul. 1998.

[23] Das, S., Yamada, K., Yu, H., Lee, S. S., and Gerla, M. A QoS Network Management System for Robust and Reliable Multimedia Services. IFIP/IEEE International Conference on Management of Multimedia Services and Networks, Oct. 2002.

[24] De, S. and Das, S.K. Dynamic Multipath Routing (DMPR): An Approach to Improve Resource Utilization in Networks for Real-Time Traffic. International

Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, Aug. 2001.

[25] Elwalid, A., Jin, C., Low, S., and Widjaja, I. MATE: MPLS Adaptive Traffic Engineering. IEEE Infocom, Apr. 2001.

[26] Farrel, A., Vasseur, J.-P., and Ash, J. A Path Compuation Element (PCE)-Based Architecture. RFC 4655, Aug. 2006.

[27] Fortz, B. and Thorup, M. Internet Traffic Engineering by Optimizing OSPF Weights. IEEE Infocom, Mar. 2000.

[28] Fortz, B., Rexford, J., and Thorup, M. Traffic Engineering with Traditional IP Routing Protocols. IEEE Communication Magazine, Vol. 40, No. 10, pp. 118-124, Oct. 2002.

[29] Girard, A. and Hurtubise, S. Dynamic Routing and Call Repacking in Circuit-Switched Networks. IEEE Transactions on Communications, Vol. 31, No. 12, pp. 1290-1294, Dec. 1983.

[30] Girish, M.K., Zhou, B., and Hu, J. Formulation of the Traffic Engineering Problems in MPLS Based IP Networks. IEEE Symposium on Computers and Communications (ISCC), Jul. 2000.

[31] Gojmerac, I., Ziegler, T., Ricciato, F., and Reichl, P. Adaptive Multipath Routing for Dynamic Traffic Engineering. IEEE Globecom, Dec. 2003.

[32] Gummadi, K. P., Pradeep, M. J., and Murthy, C. S. R. An Efficient Primary-Segmented Backup Scheme for Dependable Real-Time Communication in Multihop Networks. IEEE/ACM Transactions on Networking, Vol. 11, No. 1, pp. 81-94, Feb. 2003.

[33] Gupta, A., Gupta, A., Jain, B. N., and Tripathi, S. QoS Aware Path Protection Schemes for MPLS Networks. International Conference of Computer Communication, Aug. 2002.

[34] Gustafsson, E. and Gunnar, K. A Literature Survey on Traffic Dispersion. IEEE Network, Vol. 11, No. 2, pp. 28-36, Mar/Apr. 1997.

[35] Hamming, R. W. Error Detecting and Error Correcting Codes. The Bell System Technical Journal, Vol. 26, No. 2, pp. 147-160, Apr. 1950.

[36] Han, S. and Shin, K. G. A Primary-Backup Channel Approach to Dependable Real-Time Communication in Multihop Networks. IEEE Transactions on Computers, Vol. 47, No. 1, pp. 46-61, Jan. 1998.

[37] Han, S. and Shin, K. G. Experimental Evaluation of Behavior-based Failure-Detection Schemes in Real-time Communication Networks. IEEE Transactions on Parallel and Distributed Systems, Vol. 10, No. 6, pp. 613-626, Jun. 1999.

[38] Han, S. and Shin, K. G. Fast Restoration of Real-Time Communication Service from Component Failures in Multi-hop Networks. ACM SIGCOMM, Sep. 1997.

[39] Han, S., Shin, K. G. Efficient Spare-Resource Allocation for Fast Restoration of Real-Time Channels from Network Component Failures. IEEE Real-Time Systems Symposium, Dec. 1997.

[40] Haskin, D. and Krishnan, R. A Method for Setting an Alternative Label Switched Paths to Handle Fast Reroute. Internet draft ‘draft-haskin-mpls-fast-reroute-05.txt’, (work in progress), Nov. 2000.

[41] Huston, G. Next Steps for the IP QoS Architecture. RFC 2990, Nov. 2000.

[42] Jamoussi, B., Andersson, L., Callon, R., Dantu, R., Wu, L., Doolan, P., Worster, T., Feldman, N., Fredette, A., Girish, M., Gray, E., Heinanen, J., Kilty, T., and Malis, A. Constraint-Based LSP Setup using LDP. RFC 3212, Jan. 2002.

[43] Juttner, A., Szviatovszki, B., Szentesi, A., Orincsay, D., and Harmatos, J. On-demand Optimization of Label Switched Paths in MPLS Networks. International Conference on Computer Communications and Networks (ICCCN), Oct. 2000.

[44] Karol, M., Krishnan, P., and Li, J. J. VoIP Network Failure Detection and User Notification. Technical report ALR-2003-031, Avaya Labs Research, Jul. 2003.

[45] Katz, D., Kompella, K., and Yeung, D. Traffic Engineering (TE) Extensions to OSPF Version 2. RFC 3630, Sep. 2003.

[46] Kawamura, R., Sato, K., and Tokizawa, I. Self-Healing ATM Networks Based on Virtual Path Concept. IEEE Journal on Selected Areas in Communications, Vol. 12, No. 1, pp. 120-127, Jan 1994.

[47] Khanna, A. and Zinky, J. The Revised ARPANET Routing Metric. ACM SIGCOMM, Sep. 1998.

[48] Kim, S. and Shin, K. G. Improving Dependability of Real-Time Communication with Preplanned Backup Routes and Spare Resource Pool. International Workshop on Quality of Service (IWQoS), Jun. 2003.

[49] Kim, S., Qiao, D., Kodase, S., and Shin, K. G. Design and Evaluation of Routing Schemes for Dependable Real-Time Connections. IEEE International Conference on Dependable Systems and Networks (DNS), Jul. 2001.

[50] Kodialam, M. and Lakshman, T. V. Dynamic Routing of Bandwidth Guaranteed Tunnels with Restoration. IEEE Infocom, Mar. 2000.

[51] Kodialam, M. and Lakshman, T. V. Dynamic Routing of Locally Restorable Bandwidth Guaranteed Tunnels using Aggregated Link Usage Information. IEEE Infocom, Apr. 2001.

[52] Kuhn, D. R. Sources of Failure in the Public Switched Telephone Network. IEEE Computer, Vol. 30, No. 4, pp. 31-36, Apr. 1997.

[53] Kurose, J. Open Issues and Challenges in Providing Quality of Service Guarantees in High-Speed Networks. ACM Computer Communication Review, Vol. 23, No. 1, pp. 6-15, Jan. 1993.

[54] Lee, S. S. and Gerla, M. Fault Tolerance and Load Balancing in QoS Provisioning with Multiple MPLS Paths. International Workshop on Quality of Service (IWQoS), Jun. 2001.

[55] Li, G., Wang, D., Kalmanek, C., and Doverspike, R. Efficient Distributed Path Selection for Shared Restoration Connections. IEEE Infocom, Jun. 2002.

[56] Mahesh, K. and Lakshminaraynan, S. An Efficient Rerouting Mechanism for Dynamic Network Resource Management. IEEE International Conference on Multimedia Computing and Systems, Jun. 1999.

[57] Maxemchuk, N. Dispersity routing. International Communications Conference, Jun. 1975.

[58] Metz, C. At the Core of IP Networks: Link-State Routing Protocols. IEEE Internet Computing, Vol. 3, No. 5, pp. 72-77, Sep/Oct. 1999.

[59] Moy, J. OSPF Version 2. RFC 2328, Apr. 1998.

[60] Nahrstedt, K. End-to-End QoS Guarantees in Networked Multimedia Systems. ACM Computing Surveys, Vol. 27, No. 4, pp. 613-616, Dec. 1995.

[61] Nahrstedt, K. and Steinmetz, R. Resource Management in Networked Multimedia Systems. IEEE Computer, Vol. 28, No. 5, pp. 52-65, May 1995.

[62] OPNET. http://www.opnet.com .

[63] Orda, A. and Sprintson, A. Efficient Algorithms for Computing Disjoint QoS Paths. IEEE Infocom, Mar. 2004.

[64] Parris, C. and Banerjea, A. An Investigation into Fault Recovery in Guaranteed Performance Service Connections. Technical report TR-93-054, International Computer Science Institute, University of California, Berkeley, Oct. 1993.

[65] Parris, C., Zhang, H., and Ferrari, D. A Mechanism for Dynamic Re-routing of Real-time Channels. Technical Report tr-92-053, University of California, Berkeley, Aug. 1992.

[66] Parris, C., Zhang, H., and Ferrari, D. Dynamic Management of Guaranteed Performance Multimedia Connections. Multimedia Systems, Vol. 1, No. 6, pp. 267-284, 1994.

[67] Patterson, D., Gibson, G. A., and Katz, G. A Case for Redundant Arrays of Inexpensive Disks (RAID). ACM SIGMOD, Jun. 1988.

[68] Perkins, C. and Bhagwat, P. Routing over Multi-hop Wireless Network of Mobile Computers. ACM SIGCOMM Computer Communications Review, Vol. 24, No. 4, pp. 234-244, Oct. 1994.

[69] Pope, C. and Yantchev, J. Fault-tolerant Real-time Communication with Reduced Resource Overhead. Australian Computer Science Communications, Vol. 17, No. 1, pp. 441-448, Feb. 1995.

[70] Rabbat, R., Laberteaux, K., Modi, N., and Kenney, J. Traffic Engineering Algorithms using MPLS for Service Differentiation. IEEE International Conference on Communications (ICC), Jun. 2000.

[71] Raghavan, S. and Manimaran, G. A Note on Dependable Real-time Communication in Multihop Networks. Computer Communications, Vol. 25, No. 17, pp. 1673-1683, Nov. 2002.

[72] Ramakrishnan, K.G. and Rodrigues, M.A. Optimal Routing in Shortest-Path Data Networks. Bell Labs Technical Journal, Vol. 6, No. 1, pp. 117-138, Jan-Jun. 2001.

[73] Ramanathan, P., and Shin, K. G. Delivery of Time-Critical Messages using a Multiple Copy Approach. ACM Transactions on Computer Systems, Vol. 10, No. 2, pp. 144-166, May 1992.

[74] Ranjith, G., Krishna, G. P., and Murthy, C. S. R. A Distributed Primary-Segmented Backup Scheme for Dependable Real-Time Communication in Multihop Networks. IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems (WFTPDS), Apr. 2002.

[75] Ricciato, F., Monaco, U., and Ali, D. Distributed Schemes for Diverse Path Computation in Multidomain MPLS Networks. IEEE Communications Magazine, Vol. 43, No. 6, pp. 138-146, Jun. 2005.

[76] Rosen, E., Viswanathan, A., and Callon, R. Muliprotocol Label Switching Architecture. RFC 3031, Jan. 2001.

[77] Sahoo, A., Devalla, B., Guan, Y., Bettati, R., and Zhao, W. Adaptive connection Management for Mission Critical Applications over ATM Networks. IEEE National Aerospace and Electronic Conference (NAECON), Jul. 1998.

[78] Savage, S., Anderson, T., Aggarwal, A., Becker, D., Cardwell, N., Collins, A., Hoffman, E., Snell, J., Vahdat, A., Voelker, G., and Zahorjan, J. Detour: A Case for Informed Internet Routing and Transport. IEEE Micro, Vol. 19, No. 1, pp. 50-59, Jan. 1999.

[79] Shand, M. IP Fast-reroute Framework. Internet draft 'draft-ietf-rtgwg-ipfrr-framework-01.txt', (work in progress), Jun. 2004.

[80] Shin, K. G. and Han, S. Fast Low-Cost Failure Recovery for Reliable Real-Time Multimedia Communication. IEEE Network, Vol. 12, No. 6, pp. 56-63, 1998.

[81] Sinha, K. and Patek, S.D. OpIATE: Optimization Integrated Adaptive Traffic Engineering. Technical report SIE-020001, University of Virginia, Nov. 2002.

[82] Sprintson, A., Yannuzzi, M., Orda, A., and Masip-Bruin, X. Reliable Routing with QoS Guarantees for Multi-Domain IP/MPLS Networks. IEEE Infocom, May 2007.

[83] Swallow, G. MPLS Advantages for Traffic Engineering. IEEE Communications Magazine, Vol. 37, No. 12, pp. 54-58, Dec. 1999.

[84] Uludag, S., Lui, K. S., Nahrstedt, K., and Brewster, G. Comparative Analysis of Topology Aggregation Techniques and Approaches for the Scalability of QoS Routing. Technical Report TR05-010, DePaul University, May 2005.

[85] Varadarajan, S. and Chiueh, T. Automatic Fault Detection and Recovery in Real Time Switched Ethernet Networks. IEEE Infocom, Mar. 1999.

[86] Wang, Z. and Crowcroft, J. Quality-of-Service Routing for Supporting Multimedia Applications. IEEE Journal on Selected Areas in Communications, Vol. 14, No. 7, pp. 1228-1234, Sep. 1996.

[87] Wang, Z. and Crowcroft, J. Shortest Path First with Emergency Exits. ACM SIGCOMM, Sep. 1990.

[88] Xiao, X., Hannan, A., Bailey, B., and Ni, L.M. Traffic Engineering with MPLS in the Internet. IEEE Network, Vol. 14, No. 2, pp. 28-33, Mar/Apr. 2000.

[89] Yannuzzi, M., Masip-Bruin, X., Sanchez, S., and Domingo-Pascual, J. On the Challenges of Establishing Disjoint QoS IP/MPLS Paths Across Multiple Domains. IEEE Communications Magazine, Vol. 44, No. 12, pp. 60-66, Dec. 2006.

[90] Younis, O. and Famy, S. Constraint-based Routing in the Internet: Basic Principles and Recent Research. IEEE Communications Surveys and Tutorials, Vol. 5, No. 1, pp. 2-13, Third Quarter, 2003.

[91] Zhang, L., Deering, S., Estrin, D., Shenker, S., and Zappala, D. RSVP: A New Resource ReserVation Protocol. IEEE Network, Vol. 7, No. 5, pp. 8-18, Sep. 1993.

[92] Zhao, W., Olshefski, D., and Schulzrinne, H. Technical report CUCS-003-00, Columbia University, 2000.

[93] Zheng, Q. and Shin, K. G. Fault-Tolerant Real-Time Communication in Distributed Computing Systems. IEEE Transactions on Parallel and Distributed Systems, Vol. 9, No. 5, pp. 470-480, May 1998.

[94] Zorpette, G. Keeping the Phone Lines Open. IEEE Spectrum, Vol. 26, No. 6, pp. 32-36, Jun. 1989.

Appendices

Appendix A

Table 25 is the adjacency matrix representation of the randomly generated graph used in the experiments in Sections 3.10, 4.8.2, 6.6, and 6.8. The actual graph is shown in Figure 40. For clarity of node positions, the edges are not drawn with arrows, though each undirected edge in Figure 40 represents two directed edges, one in each direction, in the simulation setup. As mentioned in Section 3.10, this is a randomly generated graph of 20 nodes, each with a degree of four. The diameter of the graph is three. This graph was chosen for its potential existence of a moderate number of alternate paths. With adeguate alternatives, such a topology allows for assessment of the planning and negotiating abilities of different algorithms under a moderate environment. If the nodes are too well-connected, then there is likely no real difference among the methods tested in Chapter 6. If there are too few alternatives, then the unreserved backup path approach as a whole is not appropriate.

Table 25. The adjacency matrix representation of the randomly generated graph with 20 nodes, each of which has a degree of four.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	1	0	0	0	0
1	0	0	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0
2	0	1	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	0	0
3	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	1	0	0
4	0	0	0	0	0	1	0	1	0	0	0	1	0	0	1	0	0	0	0	0
5	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	1
6	1	0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0
7	0	0	0	0	1	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0
8	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0
9	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0
10	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	1
11	0	1	0	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0
12	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0
13	0	0	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	1	0
14	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1
15	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0
16	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	1
17	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0
18	0	1	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0
19	0	0	0	0	0	1	0	0	0	0	1	0	0	0	1	0	1	0	0	0

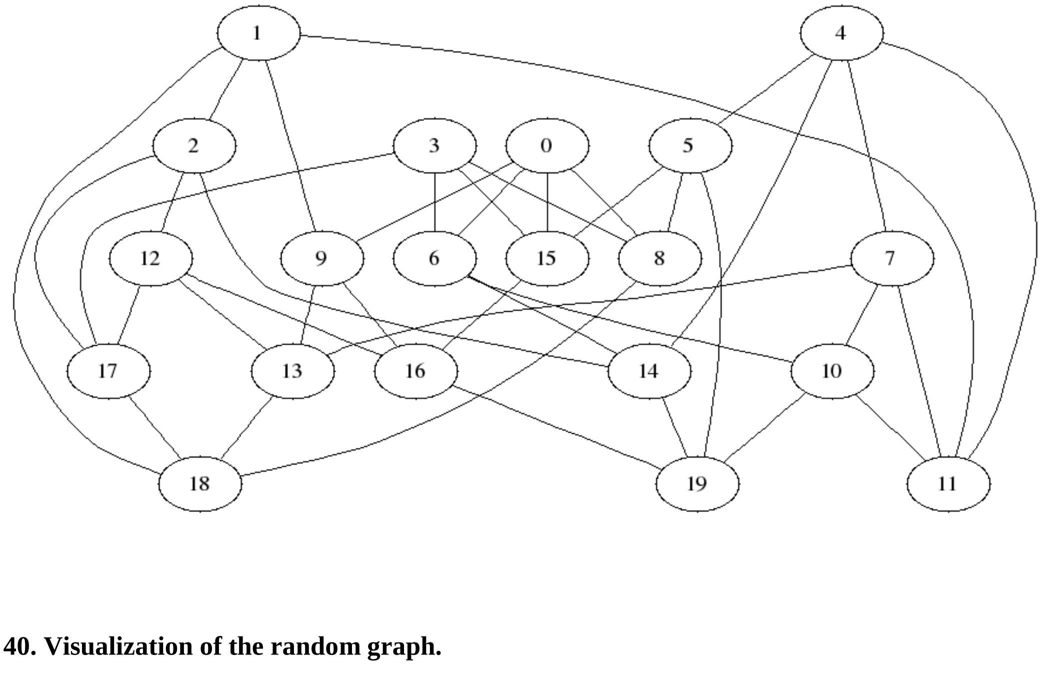


Figure 40. Visualization of the random graph.

Appendix B

One of the goals of these simulations is to determine the behavioral trends of these algorithms. Both results from light and heavy traffic loads are of interest. One point of interest in particular is how the broadcasting of routing information used in AvoidPBO, TP, and TPmax compares to BV+APV, which only sends routing information over primary paths. To capture the case where any form of broadcasting is more costly than flow-based algorithms like BV+APV, the amount of traffic needs to be relatively low. Conversely, the amount of traffic needs to be high enough to capture the case where distributing flow-based routing information, even along very short primary paths, is more costly than broadcast. All of these factors together led to this particular setting where the resource limit is set to 370 units per link and the traffic loads range from 2000 flows to 12000 flows.

Appendix C

Table 26 is the adjacency matrix representation of the randomly generated graph plus four extra links used in the experiments in Section 6.7. This graph is derived from the graph represented by Table 25. The extra links are indicated in bold and italics. Figure 41 is a drawing of the graph, with the additional edges in red. Like before, for clarity of node positions, the edges are not drawn with arrows, though each undirected edge in Figure 41 represents two directed edges, one in each direction, in the simulation setup. This graph has a diameter of three, and each node has a degree of either four or five. The additional edges in this graph are intended to test whether the algorithms can exploit these new resources by rearranging the backup paths. These additional links should result in better performance, but it is also important that algorithms actually discover these additional links and use them well together with all the other links.

Table 26. The adjacency list representation of the graph with four additional links.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	1	0	0	0	0
1	0	0	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0
2	0	1	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	0	0
3	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	1	0	0
4	0	0	0	0	0	1	0	1	0	0	0	1	0	0	1	0	0	0	0	0
5	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	0	***1***	0	1
6	1	0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0
7	0	0	0	0	1	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0
8	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0
9	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	***1***	1	0	0	0
10	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	1
11	0	1	0	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0
12	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0
13	0	0	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	1	0
14	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1
15	1	0	0	1	0	1	0	0	0	***1***	0	0	0	0	0	0	1	0	0	0
16	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	1
17	0	0	1	1	0	***1***	0	0	0	0	0	0	1	0	0	0	0	0	1	0
18	0	1	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0
19	0	0	0	0	0	1	0	0	0	0	1	0	0	0	1	0	1	0	0	0

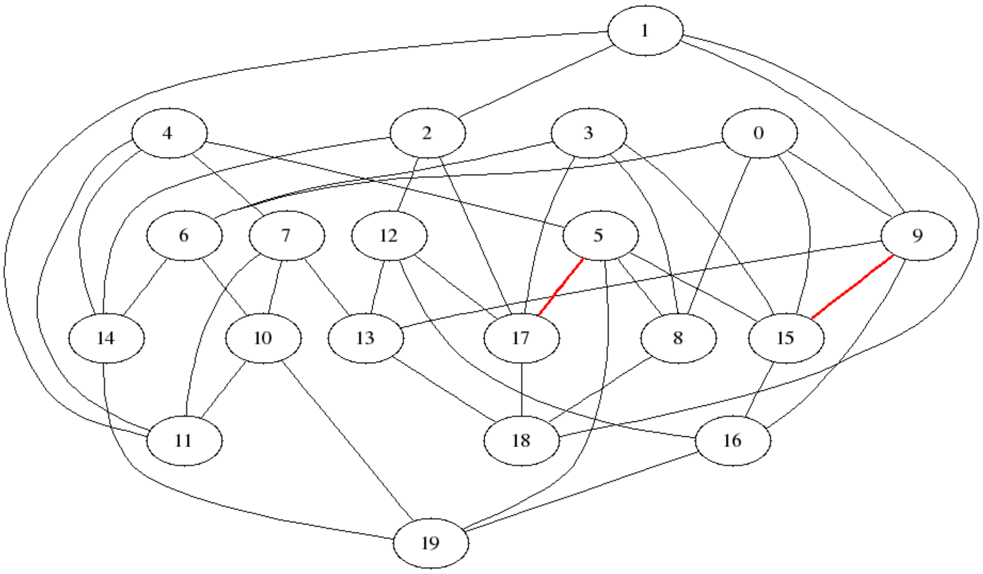


Figure 41. Visualizaion of the graph with four additional links.