Fast network recovery

Amund Kvalbein

Doctoral Dissertation

Submitted to the
Faculty of Mathematics and Natural Sciences
University of Oslo
in partial fulfillment of the requirements for the degree
Philosophiae Doctor

March 2007

Series of dissertations submitted to the
Faculty of Mathematics and Natural Sciences, University of Oslo.
No. 622

ISSN 1501-7710

Cover: Inger Sandved Anfinsen.
Printed in Norway: AiT e-dit AS, Oslo, 2007.

Produced in co-operation with Unipub AS.
The thesis is produced by Unipub AS merely in connection with the
thesis defence. Kindly direct all inquiries regarding the thesis to the copyright
holder or the unit which grants the doctorate.

Unipub AS is owned by
The University Foundation for Student Life (SiO)

Every man who is high up likes to think that he has done it all himself, and the wife smiles and lets it go at that.

— *James M. Barrie*

Abstract

The Internet is increasingly used to transport time-critical traffic. Applications like video conferencing, television, telephony and distributed games have strict requirements to the delay and availability offered by the underlying network. At the same time, connectivity failures caused by failures in network equipment is a part of everyday operation in large communication systems. The traditional recovery mechanisms used in IP networks are not designed with real-time applications in mind. The distributed nature of popular intradomain routing protocols allows them to eventually recover from any number of failures that leaves the network connected, but this is a time consuming process that can lead to unacceptable performance degradations for some applications.

In this work, we argue that there is a need for fast recovery mechanisms that allow packet forwarding to continue over alternate paths immediately after a failure, before the routing protocol has converged on the altered topology. To give rapid response, such mechanisms should be *proactive* in the sense that an alternate route is readily available when a failure is discovered, and *local*, so that the recovery action can be effected by the node that discovers the failure. Further, care should be taken so that the shifting of recovered traffic to an alternate route does not lead to congestion and packet loss in other parts of the network.

We present and investigate mechanisms that can respond quickly to failures or unexpected traffic shifts in the network. First, we evaluate the recovery strategy used in a network protocol called Resilient Packet Ring (RPR). The ring topology used in RPR allows the implementation of very fast protection mechanisms. We look at the performance of these mechanisms, and propose improvements that reduce packet loss and shorten the experienced disruption time after a link or node failure. Then, in the main part of this work, we focus on fast recovery in general mesh networks. We present Re-

silient Routing Layers (RRL) and Multiple Routing Configurations (MRC), which are methods for near-instantaneous recovery from component failures in packet networks. We discuss and evaluate our mechanisms with respect to state requirements and distribution of the recovered traffic. For MRC, we move on to present methods for reducing the chances of congestion after a recovery operation. We show that if we have knowledge about the traffic demands, we can use this information to create MRC recovery paths that avoid the most heavily used parts of the network. Finally, we show how the concepts used in RRL and MRC to give recovery from component failures also can be used to avoid congestion when there are sudden shifts in the traffic distribution. Our method is more flexible than traditional traffic engineering methods used in connectionless IP networks, since it does not involve changing link weights to respond to a changed traffic situation.

Fast recovery mechanisms like those proposed in this work can help improve the stability and availability of IP networks. This is an important requirement for enabling new and existing real-time applications over general-purpose Internet infrastructure.

Acknowledgements

This dissertation could never have been written without help and support from many people around me.

First of all, my gratitude goes to my supervisors Stein Gjessing and Olav Lysne. Stein's excitement over new ideas and his thoroughness when reviewing text and solutions, has made working with him both inspiring and rewarding. Combined with Olav's creativity and ability to distinguish the important from the less important, they have made up a great supervisor team. I have learnt a lot from both of them that I will benefit from in the years to come.

Almost equally important has been the inspiring work environment in the Networks and Distributed Systems department at Simula. In particular, the close collaboration with Audun Fosselie Hansen and Tarik Čičić has been of immense importance for this work. Audun and I have shared office for the almost four years that this project has lasted. It is hard to see how I could have found a more inspiring officemate, both socially and intellectually. Tarik has led the "Resilient Networks" project that I have been a part of. With his hearty involvement in my thesis project, he has in many ways served as a third supervisor. I am thankful also to my other friends and colleagues at Simula for creating an environment that fosters both good research and a good laugh.

Learning is a life-long process. I have been able to write this dissertation not only because I stand on the shoulders of giants, but also because of the love and support I have received from my family and friends through my whole life. The most important things I have learnt, I have learnt from them.

Finally, I owe everything to my wife Siv Merete. Thank you for sharing joys and sorrows with me every day, and for reminding me that fast network recovery really is only a small part of the picture in the end.

Contents

Chapter 1

Introduction

In recent years the Internet has been transformed from a special purpose research network to an ubiquitous platform for a wide range of everyday communication services. It has now become an integrated part of the communications infrastructure that all modern societies rely on. As our dependency on the Internet has increased, the demands on its reliability and availability have increased accordingly. A failure in central network equipment has the potential to disconnect thousands of users from essential services like personal communications, financial transactions and online gaming. The increased demand for a robust Internet is further actualized by the migration of traditional services like television and telephony from special purpose transport networks over to IP based media. Users of such services have long been accustomed to high availability and reliability. We believe that increased availability and reliability will be critical for the adoption of the Internet as the preferred transport medium for new and existing real-time applications.

The Internet was originally developed for military purposes, and the ability to recover from failures has always been a central design goal [1]. Networks running the Internet Protocol are intrinsically robust, since routing protocols such as OSPF or IS/IS are designed to update the forwarding information based on the changed topology after a failure. In these protocols, upon detecting the failure of one of its attached links, a router broadcasts an update message to all other routers in the network domain. When the new state information is distributed, each router individually calculates new valid routing tables. Such distributed protocols allow IP networks to recover from any number of failures that leaves the network connected.

However, with the increased number of time-critical services being depen-

dent on the Internet infrastructure, the traditional recovery mechanisms are no longer sufficient. A key problem is that the process that is used to restore new valid routes in a network after a failure situation is slow. A component failure is typically followed by a period of routing instability before the network converges in a new state. During this period, packets will be dropped due to invalid routes, and this may give unacceptable performance degradations for some types of applications. A central theme in this work is fast recovery mechanisms that allow packet forwarding to continue uninterrupted in a failure scenario.

1.1 Context of this work

A large and distributed organism like the modern Internet faces a wide range of challenges to its operation. Examples of such challenges are natural faults in networking components, misconfigurations, operational errors, large scale natural disasters, and malicious attacks against hardware and protocols. In addition, the network must cope with unusual but legitimate conditions caused by flash crowds and high mobility of nodes and subnets. These challenges vary widely with respect to severity, complexity and underlying causes.

A networks ability to offer a reasonable service when faced with these challenges, is referred to as the *resilience* of the network [2, 3]. A resilient network needs a range of systems and mechanisms at different networking layers to respond to the different threats. For example, redundant paths and duplication of networking equipment is often used to withstand physical failures. To be resilient against misconfigurations and operational errors, a network needs operation and management systems that can continuously monitor the network and detect potential inconsistencies. Cryptographic methods can be used to secure the exchange of routing information, so that an attacker cannot give invalid routing information to the routers. Traffic monitoring systems are needed to be able to detect and counter outsider threats like denial of service attacks.

Even if much effort is spent on preventing potential threats from materializing into actual failures, the size and complexity of large networks implies that failures will occur. A network then needs mechanisms to recover from the failure state and revert to normal operation.

This dissertation presents and discusses some such mechanisms aiming at increasing network resilience. These mechanisms are in different ways

parts of a routing protocol, and are concerned with how traffic can be routed on an alternative path through a network when a failure or an unexpected traffic pattern occurs. A unifying characteristic of the methods we present is that they are all designed to respond to an anomaly in a very short time. All the mechanisms presented in this thesis are *proactive*, in the sense that they take measures in advance to prepare for a possible challenge to the operation. We build defense structures and mechanisms that are used to continue packet forwarding and to distribute the traffic in the network in an intelligent manner during the failure situation. Our main concern is network recovery after loss of connectivity in parts of the network due to a component failure.

1.1.1 Network recovery

By network recovery we mean the process of returning to an operational state after a failure situation in a network. Recovery is used as a common term for protection and restoration. In essence the difference is that protection schemes are *proactive*, meaning that they calculate backup routes in advance, while restoration schemes are *reactive*, calculating the backup routes upon detection of failures. Hence restoration offers more flexibility in deciding the recovery action based on type and localization of the failure. Restoration also avoids the extra amount of state that is needed in protection schemes to maintain the pre-calculated backup paths. The main advantage of protection mechanisms is recovery speed. Hence, protection mechanisms are often used to give fast recovery and prevent loss of traffic until a slower but more resource-efficient restoration mechanism has created a new set of valid paths. Regarding terminology on recovery, it is also common to distinguish between global, also known as end-to-end, and local recovery. With global recovery, several or all the nodes in the network must be informed about the failure and take the appropriate action. The recovery action is typically invoked by a node that is not local to the failure. With a local approach, it is usually the node that detects the failure that also performs the recovery. Since signalling is needed, global recovery typically reacts slower to failures than local recovery.

Physical failures in the networking equipment can occur due to e.g. cable cuts, power failures or failing interface cards. Such failures can lead to a partial or complete loss of connectivity. When these failures occur, a new path must be found in the network that avoids the failed element. In central

parts of a network, routers and links are often duplicated, so that a backup is instantly available when the primary fails. When this is not the case, a communications protocol must try to re-establish the connectivity through a different path. This can be done by a network layer protocol like IP, or at a lower protocol layer.

Recovery mechanisms at different networking layers have different strengths and weaknesses [4, 5, 6]. First, they differ in the scope of failures they can recover from. Mechanisms at lower protocol layers cannot recover from failures in an IP router or the forwarding software. Conversely, several logical IP links might share a common physical fiber, and hence a physical failure might affect several IP links. Recovery from the failure of such a link is difficult at the networking layer. Second, there is a difference with respect to the efficiency and granularity of the mechanisms. Optical layer mechanisms typically work at a much coarser granularity than network layer recovery protocols, and must hence reserve more backup capacity in the network. Finally, there has traditionally been a difference with respect to recovery speed. Protection mechanisms at the optical layer can give very fast (sub-50 ms) recovery, while at the networking layer, only much slower restoration mechanisms with recovery times in the order of several seconds are commercially available.

1.1.2 The problems we address

In IP networks running a link state routing protocol like OSPF or IS-IS, a component failure triggers a global re-calculation of new routes based on the altered topology. This network-wide re-convergence is a time consuming process, and a link or node failure is typically followed by a period of routing instability and packet loss. This phenomenon has been studied in both intradomain [7] and interdomain context [8], and has an adverse effect on real-time applications [9]. Events leading to a re-convergence have been shown to occur frequently, and are often triggered by external routing protocols [10].

Much effort has been devoted to optimizing the different steps of the convergence of IP routing, i.e., detection, dissemination of information and shortest path calculation, but the convergence time is still too large for applications with real time demands [11]. A key problem is that since most network failures are short lived [12], too rapid triggering of the re-convergence process can cause route flapping and increased network instability [7].

The IP convergence process is slow because it is *reactive* and *global*. It

reacts to a failure after it has happened, and it involves all the routers in the domain. We believe that in order to achieve the short recovery times that are required by real time applications, we need protection mechanisms that are *proactive* and *local* also at the networking layer. In this work we introduce, discuss and evaluate several such mechanisms that can give fast recovery and maintain packet forwarding during the IP re-convergence phase. We first look at the protection mechanisms in a ring-based link layer technology, before we present network layer mechanisms that give fast recovery in general mesh networks.

Networks are often carefully engineered by the operators so that the distribution of traffic is fitted to the capacities of the links. When traffic is moved from the original path and over to an alternative path by a recovery mechanism, this will disturb the original traffic distribution, and may lead to congestion and packet loss [13]. Some work has been done to address this, but none of the existing proposals take into consideration the use of a proactive protection mechanism. In this work, we discuss how a good load distribution can be achieved in the network immediately after a failure situation when our proactive recovery scheme is used.

Mechanisms for controlling the traffic distribution in IP networks with shortest path routing are normally based on finding a suitable set of link weights. These link weights are calculated based on an estimate of the traffic demands between the nodes in the network. However, the traffic demands are constantly changing, and finding a good estimate of the traffic matrix that is robust to such changes is a non-trivial task [14]. Frequently adapting the link weights to a changed traffic matrix has serious implications for the stability of the network [15]. In this work, we propose a method for rapidly adapting the load distribution in the network to changes in demands, while avoiding the stability issues involved in changing the link weights.

1.2 Contributions

In this work we present and discuss mechanisms for increasing network resilience against failures and unexpected changes in traffic demands[1]. Most

[1]Most of the mechanisms and results presented in this dissertation have previously been published in the proceedings of various international conferences. A list of these papers and a discussion of the contributions from the different authors is given in appendix A.

of our contributions are related to recovery from component failures, i.e., the task of finding an alternate route through the network when the primary route can not be used due to a failed or otherwise unavailable link, router or network segment. The proposed recovery mechanisms are designed for use within a single Autonomous System (AS), sometimes referred to as *intradomain recovery*. Also, our mechanisms are mainly focusing on *connectionless* technologies, where data traffic can be sent without establishing an explicit path from the source to the destination. Finally, all the mechanisms we present are *proactive*, in the sense that alternative routes are planned in advance and are ready to use when a failure or a change in the traffic is detected.

In the remainder of this section, we present the contributions we make in this work.

1.2.1 Resilient Packet Ring protection

Our first contribution is an analysis and improvement of the protection mechanisms in a link layer technology designed with fast recovery as a primary target. Resilient Packet Ring (RPR, IEEE Std. 802.17) is a recent standard for a packet based, ring topology network. RPR is a Medium Access Control (MAC) protocol which allows packets to be sent in both directions around the ring on two counter-rotating ringlets. Ring topologies have for a long time been used to give failure recovery, and RPR is a good example of how the ring properties can be used to create very fast protection mechanisms. The RPR standard claims that a failure will be repaired within 50 ms. At the same time, RPR maintains a guarantee that all packets sent on the ring will arrive at its destination in sending order. We analyze the RPR protection mechanisms, and find that in some cases the in-order delivery guarantee prevents real sub-50 ms recovery. We proceed to describe three different mechanisms for improving the RPR recovery time. We demonstrate that with these mechanisms, the recovery times are reduced to well below 50 ms while maintaining in-order delivery. The choice between these mechanisms is a tradeoff between complexity and the amount of changes to the original standard on the one hand, and recovery times and packet loss on the other.

1.2.2 Resilient Routing Layers

To achieve fast network layer recovery from link failures in general mesh topologies, we present Resilient Routing Layers (RRL). RRL is agnostic to the network layer technology, and is applicable for both connectionless and connection oriented protocols. The key element in RRL is to create a small number of logical network topologies termed *backup configurations*. These backup configurations are used to prepare valid alternate routing or forwarding entries in each router. The pre-computed backup entries allow RRL to give almost instantaneous recovery from any link failure. Inherent in RRL is a tradeoff between the extra state information that must be stored in each router and important properties like the backup path lengths and the ability to recover from more than one concurrent failure. We provide algorithms for creating backup configurations that balance these tradeoffs in different ways, and evaluate their performance. We show that for the evaluated networks, RRL can give recovery paths that are close to the optimal, with a high probability of recovering traffic even with multiple concurrent link failures.

1.2.3 Multiple Routing Configurations

We present Multiple Routing Configurations (MRC), which is a scheme for fast recovery from link and node failures in IP networks. MRC uses backup configurations to prepare alternative routes in the network. It can be seen as a refinement of RRL that is adapted to an IP setting and extended to also protect against node failures. Like RRL, MRC is proactive and local, and allows packet forwarding to continue on an alternate route immediately after the discovery of a failure. MRC guarantees recovery from both single link and single node failures with a single mechanism. We formally define the MRC mechanism, and evaluate its performance with respect to state overhead, backup path lengths and post-failure load distribution. As our evaluations show, the routing of recovered traffic over an alternate backup path gives a new load distribution in the network, and can in some situations lead to congestion and packet loss. Hence, we present methods that improve the load distribution in the network while recovered traffic is routed according to MRC. Given an estimate of the traffic demand matrix, we seek to avoid routing recovered traffic over highly utilized links. This is done without compromising on the load distribution in the failure free case. Our evaluations show that with this method, we can achieve a post-failure load distribution

than is better than what is achieved by a full shortest path re-convergence on the altered topology.

1.2.4 Multi Topology traffic engineering

Intradomain traffic engineering in connectionless IP networks is traditionally done by carefully tuning the link weights that determine the shortest paths and thus the load on each link in the network. We argue that these methods are not flexible enough to deal with uncertainties and random variations in the traffic demands. Instead, we propose a new traffic engineering method based on the recent concept of Multi-Topology routing. We present two different ways of utilizing this for load balancing purposes. Our evaluations show that our method is significantly better at handling demand variations than traditional methods.

1.3 Scientific framework

Computer science as a discipline was born in the 1940s, and centered around the use of the newly invented electronic computers for automating computing tasks. Since then, it has evolved and developed in many different directions. Today, most institutions for higher education have computer science courses in their curricula.

The field of computer science is rooted in at least three older disciplines. First, it is closely related to mathematics, and overlaps in areas like boolean logic, graph theory and formal proofs. Second, it is related to the natural sciences through the use of controlled experiments to investigate the properties of a (often man-made) system. Finally, computer science has much in common with various engineering disciplines, since much of the knowledge in the field is gained through designing and implementing prototypes and full scale systems. Not surprisingly, it is difficult to arrive at a concise definition of computer science, or even to agree on whether it is a meaningful term at all [16].

The Association for Computing Machinery (ACM) has done an effort to define the fields of computer science and computer engineering [17]. They describe three main paradigms in the field; theory, abstraction (modeling) and design. The theory paradigm is rooted in mathematics, and is concerned with giving formal proofs for different properties of a system. The abstraction

paradigm is rooted in experimental science, and is concerned with making models of a system, and conducting experiments to measure the validity of the model. Finally, the design paradigm is concerned with building systems with certain properties, and testing whether the system meets a given set of requirements.

Two closely intertwined disciplines are computer science and computer engineering. The authors of [17] state that they find no fundamental difference between the core material in the two fields, but that computer scientists focus most on analysis and abstraction, while computer engineers emphasize abstraction and modeling.

The starting point for this work is the observation that there is a need for mechanisms that can give rapid response to failures and sudden traffic shifts in data networks. Specifically, we see the need for methods that can give fast recovery from component failures in packet switched networks, and that does this in a resource-efficient way that minimizes the chance of congestion. With this in mind, we develop and design different methods to handle failure events in computer networks. To test functional aspects and evaluate the performance of our proposed methods, we build models of networks and protocols. Seen in the context of the ACM classifications, this places our work mainly in the abstraction and design paradigms.

1.3.1 Evaluation methodology

A number of new mechanisms are proposed in this work, and their performance is evaluated and compared to existing methods. Generally, the performance of new mechanisms can be measured in three different manners; by building and analyzing mathematical models of the system, by doing measurements on existing systems or prototypes of systems, or by means of simulations [18]. In this work, all performance evaluations are done by means of simulations. This was the only feasible possibility in our case. First, the size and complexity of the relevant networks that our mechanisms are designed for makes it hard to build analytical models that accurately capture the effects of our mechanisms. Second, building large scale test networks or doing measurements on operational networks was impossible due to both financial limitations and time constraints.

Simulation models are built to resemble the behavior of an existing or imagined real life system. In a simulation model, real world components are mapped to corresponding modelled entities. There are many problems

involved in building good simulation models of large networks, and even a "good" simulation model may not capture all relevant aspects of a system [19]. Not only does the Internet grow rapidly, it also changes substantially in unpredictable ways. The web and peer-to-peer programs for sharing of content between users on a global scale are examples of applications that in short time scales have given large changes in traffic patterns. In addition comes the difficulties in selecting performance metrics that accurately reflect the different implications of a new mechanism. These considerations make it difficult to make hard statements about the performance and future benefits of a new mechanism based on simulations.

When conducting simulations described in this work, we have taken measures to make our results as trustworthy as possible. First, our simulations are limited to a single network domain (Autonomous System) rather than several interconnected networks. This makes our modelling task much easier. Also, most of our evaluations are done on many different network topologies, both real and synthetically generated. We have taken care to use traffic models that are as realistic as possible.

Two different classes of simulations are used in this work. The first kind calculates the routing between nodes and the resulting traffic distribution, but it does not simulate individual packets and does not have a clock that advances simulated time. We use this approach to evaluate network and routing properties like recovery path lengths, state overhead and load distributions. For this purpose, we have built our own simulation framework in the Java programming language.

To evaluate packet loss, delays and other dynamic properties, we have used a discrete event packet simulator based on the J-sim framework [20]. This framework gives us tools for event scheduling, time management and output handling, as well as models of networking entities like routers, links, line cards and routing protocols. In this framework we have built our network models and incorporated our novel mechanisms.

The details about simulation setup is given along with each simulated scenario.

Chapter 2

State of the Art

This chapter will give an overview of network recovery and load balancing methods that are relevant in the context of this work. Figure 2.1 shows a schematic classification of different network technologies and corresponding recovery methods. The main contributions in this work are in the context of protection in connectionless networks. Hence, in this section we focus more on proactive than reactive mechanisms, and we focus more on connectionless than connection-oriented technologies. In addition to pure recovery mechanisms, we will also take a look at the state of the art within connectionless load balancing.

	Protection	Restoration
Connectionless		
Connection–oriented		

Figure 2.1: Classification of network technologies and recovery schemes

2.1 Recovery in connection-oriented protocols

With connection-oriented network protocols, data traffic if forwarded along predefined paths from a source to a destination. These paths must be established by an appropriate mechanism before the data exchange can start. Connection-oriented protocols are dominant at the physical layer, and are also popular at the networking layer thanks to the Multi Protocol Label Switching (MPLS) protocol [21]. Both protection and restoration mechanisms can be used to recover traffic after a failure in connection-oriented networks. Since the main topic in this work is proactive recovery, we will limit our discussion to protection mechanisms. These come in two flavors; global (path protecting) mechanisms that prepare an alternate end-to-end path from source to destination, and local (link or node protecting) mechanisms that prepare a local detour around the protected component.

2.1.1 MPLS protection

In MPLS, a Label Switched Path (LSP) is set up between each source and destination. An LSP can be protected either globally by setting up a disjoint end-to-end backup LSP [22], or locally by setting up a separate backup LSP for each node or link on the path [23]. With global protection, the detection of a failure must be signalled back to the ingress node, which is then responsible for switching to the backup LSP. With local protection, the discovering node locally diverts traffic to a pre-computed LSP that avoids the failed element. This does not require signalling back to the ingress node, and hence local protection can give shorter recovery times than global protection.

Local protection in MPLS comes in two different versions, called one-to-one backup and facility backup. With one-to-one backup, a separate backup LSP is set up for each LSP that traverses a protected link or node. The backup LSP starts in the discovering node, and merges with the protected LSP at some point downstream of the protected component. Hence, as many as $(n-1)$ backup LSPs are needed to protect an LSP against all possible node failures on a path of n hops. With the facility backup, several LSPs may be protected by the same backup LSP, if they share a common point of intersection downstream of the protected component. The facility backup option will still need $(n-1)$ backup LSPs to fully protect an LSP that traverses n nodes, but now each of those backup LSPs may cover a set of LSPs. Hence the total number of backup LSPs in the network can be reduced.

However, this might come at the cost of a higher backup capacity requirement [24]. As we will see below, the high number of backup LSPs required to achieve local protection, has been addressed by several connection-oriented protection schemes.

2.1.2 Shared path protection schemes

At the physical (optical) layer, ring topologies have traditionally been used to devise fast protection mechanisms such as the popular SONET/SDH Automatic Protection Switching [25]. Optical networks have later evolved into more general mesh networks, and many mechanisms have been proposed for both protection and restoration in such networks (see e.g. [26, 27, 28]). A key concern for these mechanisms is how to set up the backup paths in the most bandwidth-efficient way. This is achieved by so-called *shared protection*, where several working paths share the same backup path. A complete survey of such mechanisms is outside the scope of this work, but we will mention two schemes because they have later been adapted to be used in packet-switched technologies at higher protocol layers.

One such approach, based on building multiple spanning trees in the network, was introduced in [29]. Later, a more general version of this approach appeared under the name Redundant Trees [30, 31]. The basic idea in Redundant Trees is to construct two trees, named red and blue, so that any node is connected to the common root in at least one of the trees in case of a link or node failure. This way, traffic that passed through the failure can always be recovered in either the red or the blue tree. They describe polynomial time algorithms for creating trees that protect against both link and node failures. The Redundant Tree approach has later been proposed used for recovery in MPLS [32]. This method differs from [31] by building a separate set of trees for each destination node, using the destination node as root. In addition, they calculate optimal primary paths, using the blue and red trees only for recovery. The authors demonstrate that the approach requires few labels and that the backup path lengths are not considerably longer than for MPLS fast reroute.

Protection cycles (p-cycles) is another approach for shared path protection [33]. The main idea is to pre-configure one or more p-cycles visiting all nodes in the topology. When a link fails, the traffic is locally switched to be routed according to the p-cycles instead of the original path. They describe an integer linear programming formulation for how the p-cycles can be opti-

mally constructed with respect to backup capacity reservation, and describe a distributed protocol that approximates the optimal solution. P-cycles was originally designed to protect only against link failures, but has later been extended to also give protection against node failures using so-called node-encircling p-cycles [34]. Much work has been done on different aspects of the p-cycle concept, including the use of Hamiltonian cycles [35] and improved protection against multiple concurrent failures [36]. While originally designed for optical WDM networks, p-cycles have later been proposed for use in IP/MPLS networks [37]. P-cycles are then set up as LSPs, and traffic is routed around the failure through an MPLS tunnel along the p-cycle. A separate set of node-encircling p-cycles must be built to give protection against node failures.

2.2 Link layer recovery

In the IEEE 802 network protocol series, which includes among others Ethernet (802.3), Token Ring (802.5) and RPR (802.17), the Spanning Tree Protocol (STP) [38] is used to give loop-free routing when several LAN segments are interconnected by switches. As the name suggests, this is achieved through blocking ports in some of the switches, so that the remaining parts form a spanning tree. Upon the failure of a switch or a LAN segment, STP must be run again to form a new valid spanning tree. Even if the original STP has later been replaced by a faster variant known as the rapid Spanning Tree Protocol (rSTP), the convergence time of the protocol is still in the range of seconds.

Some methods have been proposed to give faster recovery at the link layer. Ethernet Automatic Protection Switching (EAPS) [39] can be used to protect a LAN segment with a ring topology. With EAPS, protection is managed by a master node. During normal operation, the master node uses only one of its two ports connected to the ring, thus preventing loops. Upon detection of a failure, the master node opens its second port to restore connectivity. EAPS can give sub-second recovery times, but imposing a tree structure on a ring topology gives poor resource utilization. The need for fast recovery and more efficient resource utilization on the link layer were motivating factors behind the development of RPR, which will be discussed in chapter 3.

The use of multiple spanning trees has been proposed to improve the

efficiency of switched LANs [40]. This has led to a multi-tree solution for fast recovery [41]. Their solution is based on building a number of spanning trees so that at least one of the trees remain connected after the failure of any link or node. To achieve this, the trees must be built so that all links are excluded in at least one tree, and all nodes are leaf nodes in at least one tree. Traffic in the different trees are distinguished by using a unique VLAN tag in each tree. When a failure occurs, failure messages are sent in all the affected trees, so that the traffic sources can move traffic over to an unaffected tree. With this approach, the authors claim that they can achieve recovery times in the range of 100 ms.

Ring topologies have been popular in LAN environments. Token Rings [42] were popular in the 80s and early 90s, and consists of nodes connected in a single logical ring. These rings were often implemented in a physical star topology using a so-called Multistation Access Unit (MSAU). This layout allows fast protection against node failures, since a station on the ring can be bypassed in the MSAU if it fails. Later, a similar ring network based on token passing called FDDI [43] was introduced. The most important improvements over Token Ring were that FDDI allows higher capacity links, and uses a dual ring topology. The dual ring topology allows protection against both link and node failures, by looping traffic that reaches a point of failure back on the ring in the opposite direction. This protection technique is called wrapping, and is also used in RPR networks, as will be discussed in chapter 3.

2.3 IP routing and recovery

Two main classes of routing protocols are used in IP networks, called link state protocols and distance vector protocols respectively. In link state protocols, each node broadcasts information about its directly connected links to all other nodes in the network. This way all nodes get a complete view of the network topology, and can run a shortest path algorithm in order to decide the next-hop towards each destination and build the routing tables. Link state routing protocols are used for routing within an AS. The most popular link state protocols in fixed network are OSPF [44] and IS-IS [45].

In distance vector protocols, the routers do not build a global view of the topology. Instead, each node informs its neighbors of its current cost (distance) to each destination. The routers then decide the shortest path

to a destination by comparing the distance through each neighbor. This way, distance vector protocols avoids the broadcasting needed by link state protocols, and can sometimes manage with less signalling. Distance vector protocols are used for routing both inside an AS (e.g. RIP [46] and EIGRP [47]), and between ASes (e.g. EGP [48] and BGP [49]). EGP and BGP are both examples of a refinement of the distance vector principle called path vector, where the entire AS path to a destination is specified instead of just the distance.

In the rest of this section, we will look at recovery mechanisms in connectionless IP networks. We will focus our discussion on link state Interior Gateway Protocols (IGPs), which are used for routing within an single AS.

2.3.1 IGP restoration

IP networks are intrinsically robust, since IGP routing protocols such as OSPF or IS-IS are designed to update the forwarding information based on the changed topology after a failure. This re-convergence assumes full distribution of the new link state to all routers in the network domain. When the new state information is distributed, each router individually calculates new valid routing tables. The updated routing table, often termed the Routing Information Base (RIB), must then be loaded into the Forwarding Information Base (FIB) that is stored on the interface cards in modern routers. This network-wide IP re-convergence is a time consuming process, and a link or node failure is typically followed by a period of routing instability. During this period, which can last from seconds to tens of seconds [7], packets may be dropped due to invalid routes.

The IS-IS re-convergence process is studied in detail in [11]. They divide the re-convergence process into detection, creation of a link state update message, flooding time, shortest path calculation, the time to update the RIB and finally the time to load the new forwarding information into the FIB. They measure the time consumed in each step on a modern IP router, and find that the main contributor to the total re-convergence time is actually the time it takes to load the updated forwarding information into the FIB. Based on simulations of the backbone network of a Tier-1 ISP, they conclude that sub-second IGP convergence can be achieved without compromising stability even in large networks. But still, the recovery time can be too long for some real-time applications.

2.3.2 Local restoration schemes

To further reduce the convergence time of IGP routing protocols, some *local* restoration schemes have been proposed. These schemes try to limit the number of routers that are notified of a failure. Only those routers that must change their routing will be informed, the rest of the routers can carry on as normal. The main challenge for schemes that do recovery without global dissemination of the failure is to avoid routing loops.

One such scheme is introduced in [50]. By representing the weight of each link in the network as a vector instead of a scalar, they are able to build restoration paths that are considered to have infinitesimally smaller weight by the nodes on the path. This way, they can guarantee loop-free routing by only informing the routers on the recovery path. An important advantage of their scheme is that it does not require changes in the forwarding logic in the router.

Another local restoration scheme is presented in [51]. For each failure they want to protect against, and for each destination, they calculate in advance a backup path to the closest *feasible next hop*. This is the closest downstream node that does not include the failed element on its shortest path to the destination. If the rerouting cannot be done locally without creating a loop, the minimum number of routers on the backup path must be notified and change their forwarding table. Since they need to distinguish recovered traffic from the normal traffic, this scheme needs to make changes in the forwarding logic in the routers.

2.4 Proactive IP recovery

Recently, a number of approaches have been proposed for giving fast, local protection in connectionless IP networks [52]. These schemes require no signalling to neighboring nodes after a failure. Instead, structures are prepared in advance so that the node that discovers the failure will always have a loop-free alternate next-hop ready for use. The methods we discuss in chapters 5 and 6 in this work also into this category.

One such proactive recovery scheme called O2 routing was introduced in [53]. The basic idea is to configure a network so that all nodes have two valid next-hops to all destinations. Traffic is split between the next-hops in the normal case, and they function as backup for each other in case of a failure.

In order to avoid loops, some links are excluded from packet forwarding for certain destinations in the normal case, and are only used as backup. More refined algorithms for configuring O2 networks have later been developed [54]. A drawback of the O2 scheme is that in order to guarantee the existence of two loop-free next-hops, the routes used in the normal case are often sub-optimal. Also, the network topology must be well connected for O2 to be able to give complete protection. In less connected networks, O2 will leave some links and nodes unprotected.

Failure Insensitive Routing (FIR) is a local protection scheme that guarantees protection against all single link failures in arbitrary biconnected networks [55, 56, 57]. With FIR, routers are not explicitly made aware of a failure through notification messages. Instead, they *infer* that a link failure must be present if a packet for a given destination arrives at an unusual interface. By calculating in advance which possible link failures that would give this unusual traffic pattern, they are able to build *interface-specific routing tables* that avoids the failed link. Thus, the next-hop for a packet is dependent both on the destination address and the packets incoming interface. Later, FIR has been extended to also handle node failures [58]. A weakness with the original FIR approach was that forwarding loops could occur if more than one link failed. A solution to this problem has later been proposed [59], but with this extension, FIR is no longer able to guarantee recovery from all link failures in arbitrary biconnected networks. A drawback of the FIR approach is the need for a non-standard routing protocol.

The need for a proactive recovery scheme in connectionless IP networks has also attracted significant interest from the IETF [60]. They have discussed several protection strategies with varying protection coverage. The most important contribution that has come out of their efforts is a protection scheme called Not-via, which guarantees protection against any single link or node failure in arbitrary biconnected networks [61]. Their approach is based on tunnelling packets that would normally be routed through the failure to the router beyond the failure, "not-via" the failed element. To protect against the failure of a node P, a special "N-not-via-P" address is assigned to each of P's neighbors. The shortest paths to these addresses are calculated by all routers in the network on a topology where P is removed. When the failure of P is detected, traffic that that would normally be routed through P to N is encapsulated and sent shortest path to "N-not-via-P". At node N the packet is decapsulated, and forwarded as normal to the destination. Since N is closer to the destination than P, we will never get routing

loops. This is an elegant solution that gives gives intuitive and well defined backup paths. A drawback with this method is the large number of shortest path computations that are needed to calculate the routing for all the not-via addresses. However, the authors claim that these calculations can be significantly reduced by relying on incremental algorithms. Another drawback is the use of tunnelling, which imposes an extra burden on the routers that must encapsulate and decapsulate packets.

Proactive recovery mechanisms at the IP layer are imagined used to maintain a valid routing and avoid packet loss during transient failures. If the failure is permanent, the normal IGP re-convergence process must be triggered, and a new set of global shortest paths must be calculated. During the re-convergence process, so-called *micro-loops* can occur, due to the asynchronous transition to the updated routing tables [62]. This can be avoided by controlling the order in which the routers update their FIBs [11]. The same technique can be used to avoid micro-loops after a link weight change in the network, as long as only a single link weight is changed at a time.

2.5 Connectionless load balancing

Related to the issue of recovery is the question of how the traffic is distributed in the network before and after a failure. In a well engineered network, the traffic load is spread on the available inks in a way that maximizes throughput and minimizes the chances that congestion occurs. In a recovery context, care should be taken so that the recovered traffic does not cause congestion and packet loss on the recovery path.

It has long been known that for a given network and a given traffic matrix, it is possible to find a set of routes that is optimal with respect to some objective function [63]. Recently, it has been shown that a good routing can be found with limited or no knowledge of the traffic demands [64]. They show that for their evaluated current ISP networks, a routing can be found that performs within 2 times the optimal for *any* traffic matrix. However, these schemes require the ability to route any fraction of the demands over selected and diverse paths. In MPLS networks, paths can be explicitly laid out in order to meet some traffic engineering objective [65, 66], and it is possible to obtain this optimal routing.

Connectionless IP protocols like OSPF and IS-IS on the other hand, are restricted to shortest path routing, and cannot in their present form achieve

optimal routing. It has been shown that it is possible to find an optimal routing so that all routes are shortest paths [67]. However, this requires the ability to split the traffic towards a destination in arbitrary fractions between several equal cost paths, which is not possible with existing routing protocols. An approach that approximates the optimal solution using only equal splitting between equal cost paths has later been developed [68]. Both these approaches require a central entity that calculates how each router should split its traffic towards each destination on the available links. Hence, each router can no longer make routing decisions independently based on simple link weight parameters as is done in OSPF and IS-IS today.

Much of the work on connectionless load balancing in IP networks focus on finding link weights that give a good load distribution for a given estimate of the traffic matrix. It has been shown that by using heuristics to carefully tune the link weights in a network, one can achieve a load distribution that is close to optimal for a given traffic matrix [69]. Extensions of this work has mainly followed two directions; finding link weights that are robust to uncertainties in the traffic matrix estimates, and finding link weights that are robust to link failures.

It is generally difficult to get good estimates of the traffic demand matrix in an operational IP network [14]. Hence, it is desirable to find a set of link weight that performs well under several possible traffic scenarios. A first approach to handle this problem is described in [15]. They devise a method for finding a single set of link weights that gives good routing with more than one traffic matrix. Such an approach is well suited to handle the well-known diurnal variations in network traffic. Further, they present a method for adapting the set of link weights to a new traffic situation (caused by e.g. a link failure or a sudden hot-spot) by only changing a very limited number of link weights. Later, a method has been proposed that takes into consideration the tradeoff between the average and the worst case when optimizing for several traffic matrixes [70].

Some work has also been done on finding link weights that are robust to link failures. An approach to find a single set of link weights that perform well in both the normal case and after any single link failure is presented in [71]. According to their evaluations, they can reduce link overload after a link failure by 40%, at the cost of a 10% performance degradation in the normal case. A problem with this approach is the extensive computations needed to predict the load distribution after all possible failures. Because of this, methods have been proposed that only optimize the link weights for a subset

of the most critical link failures [72, 73]. Very little work has previously been done with respect to the load distribution after a failure when a proactive recovery mechanism is used.

Chapter 3

Resilient Packet Ring Recovery

In this chapter, we look at the recovery mechanisms used in the recent Resilient Packet Ring (RPR) standard [74, 75]. This link layer technology takes advantage of the special ring topology to implement very fast recovery mechanisms. However, the in-order delivery requirements in RPR leads to a somewhat longer disruption in packet delivery after a failure. To improve on this, we propose a set of new mechanisms that reduce the response time and packet loss, while still maintaining the in-order delivery guarantees.

Most of the contents in this chapter has previously been published in [76] and [77].

3.1 Introduction

RPR is a standard for packet based, dual-ring topology networks. It has been developed for use in metropolitan and wide area networks, with link capacities of several Gigabits per second. Traffic in an RPR network may be sent shortest path along one of the two counter-rotating unidirectional *ringlets*. Each ringlet carries both data and control packets. A packet is stripped from the ring by the receiver node, which makes the bandwidth downstream of the receiver available for other data streams. This property is known as spatial reuse, and is illustrated in figure 3.1. Spatial reuse makes RPR more resource efficient than older ring technologies like Token Ring or FDDI, where packets traverse the whole circumference of the ring before they are removed.

Three packet priorities are supported in RPR, with strict delay guarantees

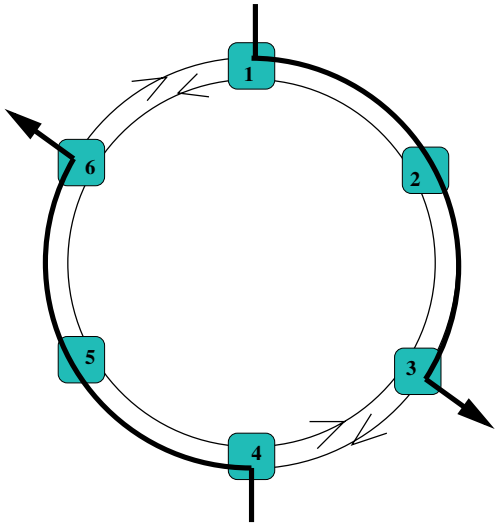


Figure 3.1: Spatial reuse allows several packets to be sent on different spans of the same ringlet at the same time.

for traffic with the highest priority. RPR is a buffer insertion ring, and can be implemented with a single or a dual transit buffer design. In a single buffer design, all packets flow through the same transit buffer. In a dual buffer design, the highest priority packets use one buffer, while the two lower priorities use the other. Traffic that is in transit on the ring has priority over local add traffic. To prevent starvation, a fairness mechanism is used to give each node a fair share of the available bandwidth. We have earlier analyzed this fairness algorithm, and found that it requires careful parameter setting in order to be stable [78]. We have also suggested improvements to increase the stability and decrease the convergence time of the RPR fairness algorithm [79].

RPR makes two important guarantees with respect to packet delivery. First, in normal operation, packets that are admitted on the ring are never dropped. If the demands are larger than the capacity, all packet dropping takes place at the ingress of the ring. Second, by default packets are deliv-

ered to the receiver in the same order as they were sent. This guarantee is maintained even in a failure scenario.

RPR is designed with a protection mechanism aiming at restoring traffic within 50 ms in case of a link or node failure. Since every node on the ring is reachable through either of the ringlets, one ringlet can serve as a secondary path for traffic on the other. The operations of the RPR protection mechanisms are transparent to higher layer protocols like IP, except for the performance degradation that might be experienced due to congestion and increased path lengths after a failure.

In this chapter, we discuss the recovery mechanisms used in RPR, with a special emphasis on packet reordering. The IP protocol does not guarantee in-order delivery of packets, and higher layer protocols must therefore be able to recover from any amount of packet reordering. In practice, however, many services are based on the assumption that packet reordering is sufficiently low. If this assumption is not fulfilled, the consequence with respect to performance can be severe. For example, TCP's fast retransmit optimization treats a reordering spanning more than a few packets as a loss [80]. Since TCP interprets packet loss as a sign of congestion, this has serious implications on throughput performance [81].

By default, packets sent over an RPR ring are guaranteed to arrive at the egress node in sending order. To achieve this, RPR uses a 40 ms, configurable from 10 to 100 ms, topology stabilization timer in the event of a failure. During the topology stabilization period, all strict order packets are discarded on the ring. We analyze the mechanism used in the RPR standard, and argue that the current mechanism is not satisfactory. Instead, we suggest three different alternatives, all giving reduced packet loss and disruption time compared to the original mechanism. The performance of the three new methods and the original are compared through simulations.

3.2 Recovery in Resilient Packet Rings

The RPR protection mechanisms are designed to restore traffic on the ring within 50 ms of a link or node failure, including the failure detection time.

3.2.1 Failure detection

A node or link failure can be discovered in two different ways in RPR. Many physical layer technologies can issue an alarm (e.g. SONET alarm) to the higher layers if connectivity is broken. Alternatively, an RPR node declares a link broken if it fails to receive a keep-alive packet from the neighbor in a specified time interval, which defaults to 3 ms.

RPR uses a hold off timer to prevent RPR protection mechanisms from declaring a link broken based on glitches in the received traffic. For example, such glitches can occur due to protection switching of RPR traffic by underlying physical layer protection mechanisms. RPR is currently defined over SONET/SDH and Ethernet/PacketPHY physical layers. The hold off timer can be up to 200 ms, the default value is zero.

All RPR nodes maintain a topology image, that includes information about the hop count to the other nodes on both ringlets. When a failure occurs, the nodes that discover the error broadcasts a topology update on the ring, informing the other nodes that the ring is broken. The reception of such a message causes each node to update its topology image. Traffic is moved over to the other ringlet as necessary.

3.2.2 Wrapping and steering

RPR offers two different recovery mechanisms, called *wrapping* and *steering*. Figure 3.2 shows an RPR ring in a normal, a wrapped and a steered condition respectively.

RPR nodes may support wrapping, in order to reduce packet loss in a failure situation. If used, it must be supported by all node on the ring. Wrapping works in much the same way as SONET/SDH Automatic Protection Switching. Traffic reaching a point of failure is wrapped over to the opposite ringlet, as shown in Fig. 3.2. Wrapping is used in RPR to prevent loss of the traffic in transit on the ring when a failure occurs. Note that wrapped packets will arrive at the destination on the same ringlet they were first transmitted on.

Steering is the default protection mechanism in RPR, and must be supported by all nodes. Steering relies on the source node to transfer traffic to the ringlet where the destination is still reachable, as shown in Fig. 3.2. This gives a more optimal utilization than wrapping of the bandwidth resources after a failure situation.

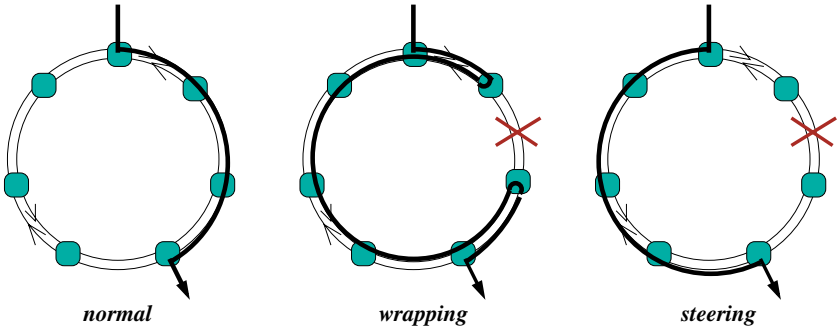


Figure 3.2: RPR ring in normal, wrapped and steered state.

Wrapping is normally used as the first step in a two-step wrap then steer protection strategy. This way, the number of packets lost is minimized, while the network utilization is maximized when the steering kicks in.

A wrap then steer protection strategy might introduce reordering of packets on the ring. By default, RPR packets are marked *strict order*, meaning that they are guaranteed to arrive in the same order as they were sent. Since wrapping can introduce packet reordering, strict order packets are never wrapped. Instead, the RPR standard prescribes that such packets should be dropped from the ring until the updated topology has been unchanged for one *context containment* period, and the new topology has been verified. The purpose of this period, which defaults to 40 ms, is to make sure that no packets that were sent in an old topology context, will arrive at a destination in a different topology context. When the context containment period expires, a checksum on the topology image is calculated, and sent to the neighboring nodes. The topology image in a node is declared stable when the locally calculated checksum equals that received from the two neighboring nodes. In the context containment period before the new topology is declared stable, all strict order packets are discarded at ingress and egress of all transit nodes, and no new strict order packets are accepted on the ring. The context containment period must be long enough to allow every node to completely empty its transit buffer. In this chapter we show that, because of the context containment period, RPR can not usually fulfil a 50 ms restoration guarantee for strict order traffic.

Wrapping is only performed on *wrap eligible* packets. Only non strict order packets can be marked as wrap eligible. A non strict mode packet that is not wrap eligible, is discarded when reaching a wrap point, but it can be steered without waiting for context containment period to expire. This way, we effectively have three different types of packets with respect to failure handling: strict order packets, non-strict order wrap eligible packets, and non-strict order non-wrap eligible packets.

3.3 Analysis of the RPR protection mechanism

The main goal of the RPR protection mechanism is to minimize the consequences for the traffic in case of a network failure. Specifically, RPR should guarantee sub-50 ms protection time. In this section, we will discuss to what extent the RPR protection mechanisms achieve this. Three important metrics in a failure situation are discussed, 1) the experienced disruption in the traffic at the receiving node, 2) the number of packets reordered, and 3) the number of packets lost.

3.3.1 Traffic disruption

When a failure occurs in an RPR network, the receiver will typically experience a period with no arriving traffic, before the protection mechanism restores the traffic on the secondary ringlet. With this in mind, we define the disruption time as experienced by the receiver, T_D, as the time between the arrival of the last packet that was not affected by the failure, and the first packet that was wrapped or steered by the protection mechanism. Figure 3.3 shows a typical arrival sequence during a failure situation.

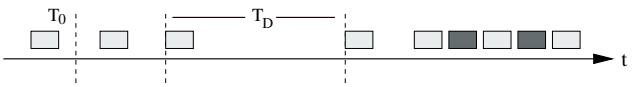


Figure 3.3: The receiver typically experiences a period T_D without arriving packets during a failure situation occurring at time T_0. The receiver might experience reordering of packets after the failure.

T_D depends on the size of the ring, traffic load, traffic priority, and the locations of the sender, receiver, and the point of failure. Topology updates are sent with the highest priority, while data packets can be sent with any of the three packet priorities. With steering and non strict order traffic, T_D is made up of (i) the time it takes for the adjacent nodes to discover the failure, (ii) the processing time it takes for these nodes to update their edge state and produce a topology update message, (iii) the propagation time, including buffering in the high priority transit buffers along the ring, of the topology update messages from the point of failure to the traffic source, (iv) the processing time to perform the topology update in the source node, and (v) the data packet propagation delay, including buffering in high or low priority buffers along the ring, from the traffic source to the destination along the new ringlet. Note that traffic in transit that has already passed the point of failure, will still reach the destination, and thus contribute to a shorter experienced disruption. The processing needed in (ii) and (iv) is not complex, and can be performed in order of a few microseconds in modern switches. Hence, the disruption time is dominated by points (i), (iii) and (v). The buffering delay for high priority packets in each transit node, is bounded by the time it takes to transmit one MTU. At the high bandwidths of an RPR network, this time is no considerable factor. Hence, a good estimate of the experienced disruption at the destination can be given by summing the error discovery time in (i), the propagation delays from (iii) and (v), and subtracting the time when traffic in transit is still arriving at the original ringlet. Formally, using the notation shown in Fig. 3.4, we can estimate the disruption time T_D as shown in Eq. 3.1.

$$T_D = T_{discovery} + \sum_{d_i \in SF} d_i + \sum_{d_i \in SD_{new}} d_i - \sum_{d_i \in FD} d_i \tag{3.1}$$

In Eq. 3.1, d_i denotes the propagation delay of a link between two nodes, including the buffering delay in the transit queue. SF is the set of links between the traffic source and the point of failure. FD is the set of links between the failure point and the destination. SD_{new} is the set of links between the traffic source and destination along the secondary ringlet. Note that the two last parts of Eq. 3.1 include buffering delays for the data packets in the transit buffers, and will therefore vary with the traffic load. Figure 3.4 shows a generic RPR ring with a source S, a destination D, and a point of failure F.

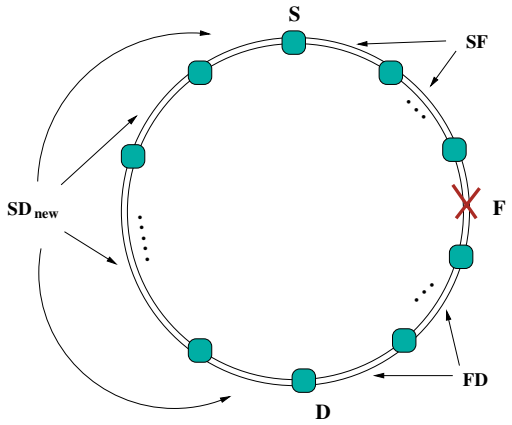


Figure 3.4: The experienced frame loss is dependent on the relative position of the source S, the destination D and the point of failure F.

Equation 3.1 above shows the situation for non strict order packets using steering. With a wrapping only scheme, the disruption time will normally be longer, since the wrapped traffic must travel the whole circumference of the secondary ringlet.

Simulated values for T_D for non strict order traffic is plotted in Fig. 3.5. In our simulations, we have used a discrete event packet simulator, based on the J-Sim framework [20]. The failure discovery time is set to 3 ms, and the simulations are made for a ring with 64 nodes. For all simulations in this chapter, the link propagation delay is 0.2 ms, corresponding to a link length of about 40 km. Node 0 is the traffic source, sending low priority traffic to node 31. The time plotted is the time between the receipt of the last packet on the primary ringlet and the first packet on the secondary ringlet. The quotient obtained by dividing the propagation delay over the FD links by the propagation delay over the SD_{new} links, denoted FD/SD_{new}, is increased along the x axis.

Turning first to a lightly loaded ring, the experienced disruption is close to the value derived from Eq. 3.1 with zero buffering delay in the transit nodes. With steering, T_D decreases as FD/SD_{new} increases, since the source quickly becomes aware of the failure, and traffic that has already passed the point of

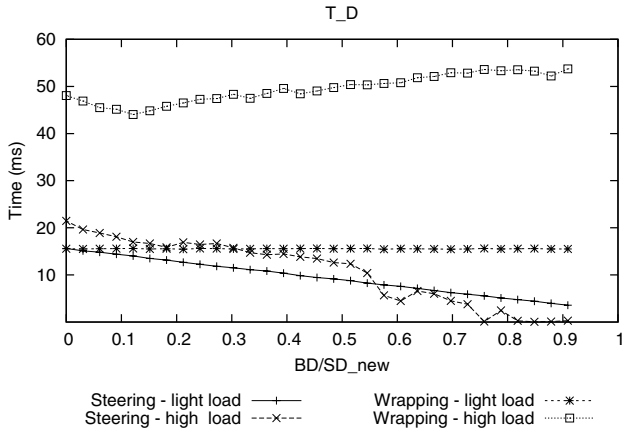


Figure 3.5: The disruption time, T_D, for non strict order traffic decreases if the secondary ringlet is not much longer than the primary. In heavily loaded networks, the transit buffer delays cause the disruption time to deviate from a lightly loaded scenario.

failure keeps arriving on the primary ringlet for some time after the failure. For wrapping, T_D is independent of FD/SD_{new}, since the wrapped packets always travel the whole circumference of the ring. For a heavily loaded ring, T_D varies more. Note that when FD/SD_{new} is close to 1, T_D might formally become negative when steering is used, since packets keep arriving on the primary ringlet even after traffic starts arriving on the secondary ringlet (large last term in Eq. 3.1). Since a negative disruption time intuitively seems meaningless, these values are plotted as zero in Fig. 3.5. With wrapping only, the transit buffer delays cause T_D to increase further, and even exceed 50 ms in our 64 node scenario.

3.3.2 Packet reordering

As explained above, we may get packet reordering on a heavily loaded ring when the ring is broken close to the source (few links in SF), and the distance along the secondary ringlet is not much longer than along the primary ringlet ($\mid SD_{old} \mid \approx \mid SD_{new} \mid$).

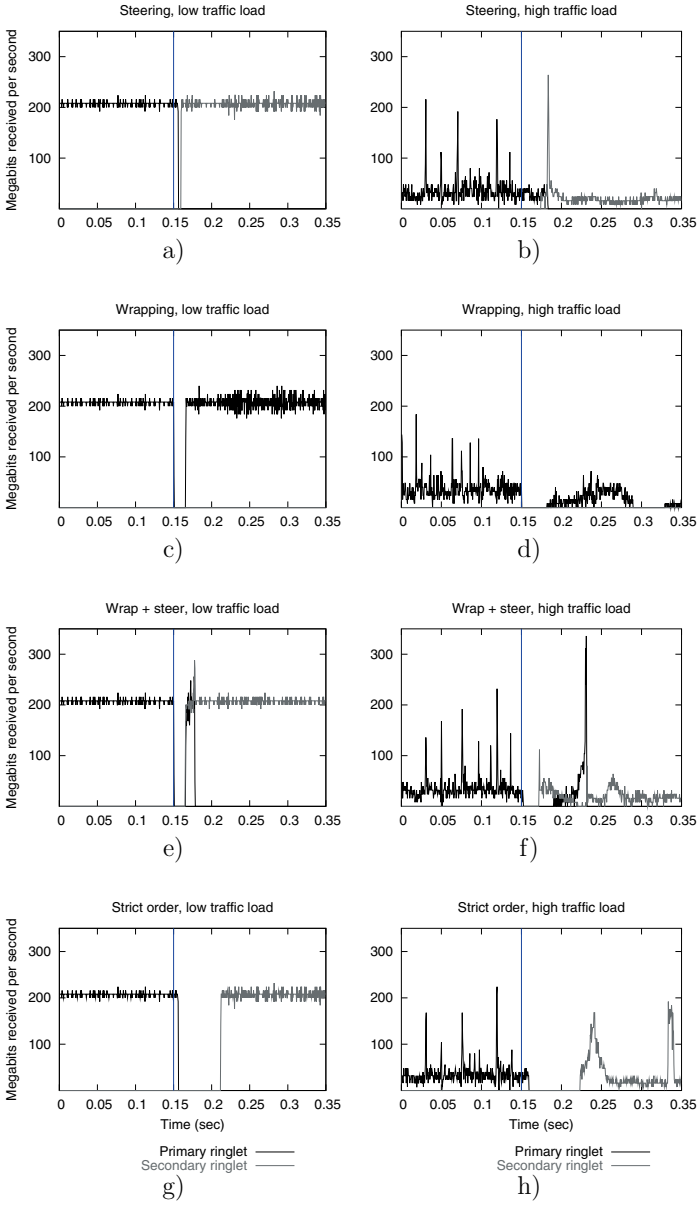


Figure 3.6: Received traffic on primary and secondary ringlet after a failure. Number of bytes received from node 0 in node 31. The vertical line marks the time of the failure. Failure occurs at node 1 in plots a, b, g, h, and at node 30 in plots c, d, e, f.

Fig. 3.6 shows the received traffic on the primary and secondary ringlets in a failure scenario. In our 64 node scenario, node 0 sends a continuous stream of 500 bytes low priority packets to node 31. All other nodes send traffic to random receivers. The background traffic is modelled to have self-similar characteristics, using superimposed Pareto-distributed ON/OFF traffic sources as outlined in [82]. A link on the primary path is broken at time 0.15. Figure 3.6 shows how traffic is shifted over to the secondary ringlet. The eight plots in Fig. 3.6 show the arrival rate from node 0 in node 31. Results are shown for both steering, wrapping, wrap + steer, and strict order steer, and for a heavily and a lightly loaded ring. With steering, failure occurs at the link between nodes 0 and 1, while with wrapping, the point of failure is between nodes 29 and 30. The failure points are chosen this way to highlight the differences between the schemes. A failure close to the source shows that reordering can occur with steering, while a failure close to the destination gives reordering when wrap + steer is used.

Figure 3.6a) shows that on a lightly loaded ring using steering, all the traffic that is sent on the primary ringlet arrives at node 31 before traffic starts arriving at the secondary ringlet. Thus, we get no reordering in this situation, since the primary path is shorter than the secondary path. Figure 3.6b) shows that when the load on the ring is increased, frames that were delayed in the transit buffers along the primary ringlet, keep arriving even after traffic starts arriving on the secondary ringlet.

In plots c) and d), only wrapping is used. In this scenario, there will be no reordering of frames, regardless of the load in the network. Remember that all wrapped packets are received from the primary ringlet. Wrapping not succeeded by steering results in poor bandwidth utilization after a failure. This is the reason for the low packet arrival rate after time 0.30 in plot 3.6d).

Figure 3.6e) and f), show that when wrapping then steering is used for protection, reordering of frames will occur regardless of the load on the ring. The reordering happens when the traffic source updates its topology image, and starts sending traffic directly on the secondary ringlet. This traffic might then arrive at the destination before the wrapped traffic, which has to traverse the whole circumference of the ring. In a heavily loaded ring, the period with packet reordering will be longer.

Finally, Fig. 3.6g) and h) shows the situation for strict order traffic. No reordering occurs, but the restoration time is increased by at least one context containment period (40 ms).

Note for all plots, that with low traffic load, the jitter increases after the

failure, because the failure gives an increased load on parts of the network. With heavy load, the experienced throughput is decreased, since the capacity is already fully used.

3.3.3 Packet loss

Another important metric is packet loss. The experienced packet loss is dependent on which protection method is used, and the relative placement of the source, the point of failure, and the destination. Packets that are in transit between the source and the point of failure, and packets that are sent by the source after the failure occurs, but before the topology update has reached the sender, will be lost if wrapping is not used. Let t_a denote the propagation + buffering delay from the source to the point of failure, t_b denote the propagation + high priority buffering delay from the point of failure back to the source, and consider a failure that occurs at time T. Then frames sent in the interval $\langle T - t_a, T + t_b \rangle$ will be lost. Figure 3.7 shows how the number of frames lost increases with SF in a 64 node ring when steering is used. Again, node 0 sends a continuous stream of low priority 500 byte packets to node 31.

If wrapping is used, the frames in transit will be wrapped back on the opposite ringlet, and are not lost. With wrapping, only frames in the failing node or at the failing link, and frames reaching a point of failure before the failure has been discovered (default 3 ms), are lost. As seen in Fig. 3.7, the packet loss is independent of the point of failure.

Strict order traffic experiences significantly more packet loss, due to the 40 ms context containment period following a failure, as seen in Fig. 3.7. Nodes in a context containment state, discard all strict order packets until the topology image is declared stable.

Note that after recovery from the failure, the nodes on the ring will experience lower throughput due to the reduced bandwidth, but no additional packets are lost on the ring.

The simulated results highlight the need for a better mechanism that can secure strict packet ordering without using the context containment period that gives a longer disruption time and increased packet loss.

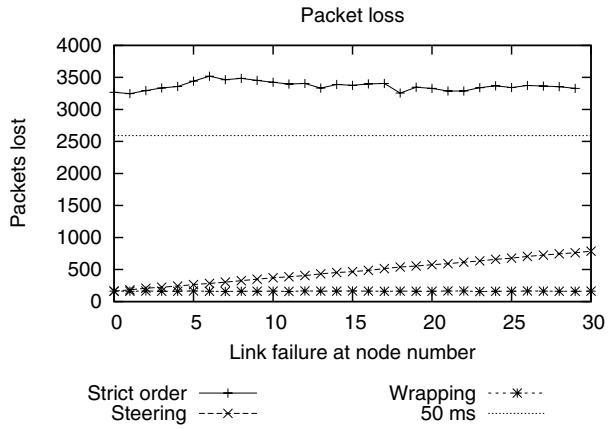


Figure 3.7: The experienced packet loss is significantly higher if strict order is required. The horizontal line shows 50 ms worth of packet production.

3.4 Improved protection mechanism for strict order traffic

In this section, we present three different ways to reduce the consequences of a link or node failure for strict order traffic. We refer to the methods as *Automatic*, *Receiver* and *Selective Discard* in the text below.

The *Automatic* method requires only very small changes to the existing specification, while the *Receiver* and *Selective Discard* methods demand some logic to be moved from the transit nodes to the receiver node. The *Selective Discard* method gives the most optimal performance, at the cost of some added complexity. None of the three proposed mechanisms rely on packets to be sent shortest path before a failure. Hence, they can easily be adapted to handle network elements that are restored on the ring, as well as failing elements.

3.4.1 Automatic setting of the topology stabilization timer

As explained above, the context containment period is used to make sure that no strict order packet in a particular flow sent in a new topology context, will arrive at the destination before a strict order packet from an old context. Once a node enters context containment, all strict order packets are discarded upon reception and transmission from the node. Hence, the period with context containment must be long enough to let a node on the ring empty its transit buffers. Once all packets have left the transit buffers in a node, the node is in the clear with respect to packet reordering. The aim of the *Automatic* method is to reduce packet loss by reducing the value of the topology stabilization timer to the minimal safe value.

In the existing RPR specification, the topology stabilization timer is configurable in the interval 10 to 100 ms. In many cases, even the minimal value of 10 ms is too restrictive with respect to reordering. The optimal value of the topology stabilization timer depends on the link bandwidth, the distribution of traffic in the different service classes, and the size of the transit buffer.

We observe that propagation delays between the source, destination and the point of failure do not influence the setting of the stabilization timer. Remember that a topology update is broadcast on both ringlets with the highest priority after a failure. This ensures that all nodes on the ring will receive notification of a failure and enter context containment before any steered traffic can reach that node. Once a node A enters context containment, no more strict order traffic is sent from that node for one context containment period. In other words, once a topology update that triggers context containment arrives on the next downstream node B, no more strict order packets will arrive from A for at least one context containment period. It is then sufficient with a context containment period that is long enough to guarantee that all strict order packets have left the transit queues in node B.

To find the minimal value for the topology stabilization timer, we need to know the maximal time a data packet can be delayed in the transit buffer. Figure 3.8 shows a simplified view of the data path for one ringlet in an RPR node. This data path implements the dual transit buffer design, where the highest priority class A traffic flows through the Primary Transit Queue (PTQ), while class B and C traffic uses the Secondary Transit Queue (STQ).

In the worst case, the STQ might be completely filled up by transit traffic.

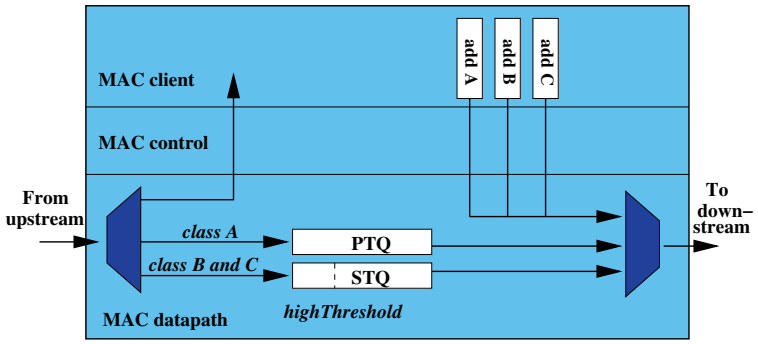


Figure 3.8: Simplified view of the transit path for one ringlet in an RPR node, with dual transit queues. Each service class has its own transmit buffer in the MAC client.

The time it takes for the last packet in the buffer to be transmitted, depends on the available link rate assigned to the STQ. As seen in Fig. 3.8, the output from the STQ has to compete for the outlink bandwidth with the traffic from the PTQ and the add traffic from the MAC client.

The class A packets from the PTQ and class A and B packets from the MAC client have precedence over packets from the STQ in the output selector. Class A and B traffic is shaped at each source node, to make sure no node exceeds its allowed rate. In the worst case, all nodes on the ring send class A traffic through this node on this ringlet. At the same time, the MAC client transmits class A and class B traffic at its maximum allowed rate.

Packets from the STQ have absolute precedence over added class C packets from the MAC client as long as the STQ is filled above a *highThreshold*, which defaults to one quarter of the STQ size. When the STQ level drops below this threshold, the available bandwidth is shared equally between the STQ transit traffic and the MAC client class C add traffic.

Equation (3.2) shows an estimate of the upper bound on the time T needed to completely empty the STQ. The numerator contains the size of the STQ. The part below the highThreshold is counted twice, since the available bandwidth is potentially halved below this threshold. The denominator

contains the outlink bandwidth[1], minus the worst case bandwidth used for traffic from the PTQ and class A and B traffic added from the MAC client.

$$T = \frac{(stqSize + highThreshold)}{outLinkRate - (\sum_{numNodes-1} rate_A) - rate_B} \tag{3.2}$$

In a node with a 1 Gb/s outlink, a STQ size of 256 kB, a total maximum of 10% of the bandwidth used for class A traffic, and a 10% total limit on class B traffic, the calculated maximum time required to flush the STQ, is about 3.2 ms. This is the minimum timeout value we can give the topology stabilization timer while still fulfilling the strict ordering guarantee.

3.4.2 Discarding packets at the receiver

Reducing the topology stabilization timer to the optimal value gives a reduced packet loss count. However, the mechanism still does not differentiate between traffic from different sources. Packets that were not affected by the failure, will still be discarded during the context containment period. This is clearly sub-optimal: packet flows that are transmitted on the same ringlet before and after the failure, can never experience reordering, and should not be discarded.

Figure 3.4 shows a generic RPR topology with a source S, a destination D, and a failure point F. By default, RPR will send packets from the MAC client shortest path (minimum hop count) around the ring. However, the MAC client may choose to override this, by explicitly defining which ringlet should be used on a per packet basis. This could be done for instance to avoid congestion points on the ring, and thus improve the throughput.

Since the MAC client can choose which ringlet to transmit a packet on, a destination D will possibly receive packets from a source S on both ringlets when the ring is connected. However, when a connectivity failure occurs, there is only one valid path from S to D. Packets will only be sent along this path, irrespective of the preferences of the MAC client.

The *Receiver* mechanism seeks to avoid dropping unaffected packets, by exploiting the knowledge of where the ring is broken. With the *Receiver*

[1]Class A traffic is divided into subclasses A_0 and A_1. Subclass A_0 uses reserved bandwidth, which cannot be reused by other traffic classes. *unreservedRate* - $rate A_1$ would therefore be a more precise formulation than outLinkRate - rateA.

mechanism, transit nodes no longer drop strict order packets during context containment. Instead, packets are dropped selectively by the receiving node, based on the source address of the packet. Once the first topology update is received by a source S (remember that two topology updates will be received, one on each ringlet), the topology image is updated so that the source knows which nodes are reachable on each ringlet. Similarly the other way around, the topology update gives a destination D enough information to know which sources S can send packets to D on each ringlet, without passing a point of failure F. For each ringlet, the receiver can then decide which nodes are "valid" sources. With the *Receiver* mechanism, only packets received from nodes that are not "valid" are discarded, since only these frames are potentially received out of order.

Transferring the responsibility for discarding packets to the receiver somewhat increases the traffic load on the ring in the time following the failure compared to the standard RPR and *Automatic* methods, since packets are not discarded immediately by transit nodes. This also prevents the topology stabilization time from being set as low as indicated by (3.2), since packets from an old topology context will exist on the ring for a longer period of time. However, increasing the topology stabilization timer will not result in increased packet drop count in this situation, since only packets originating from beyond the point of failure F on the ringlet are discarded. Such packets will stop arriving as soon as the traffic is steered over on the secondary ringlet.

As shown below in Sec. 3.5, the *Receiver* method reduces the packet drop count, compared to the *Automatic* method. This gain in performance comes at the cost of some added complexity in the receiver node, since the receiver must do a check on each received packet during context containment, to decide whether it should be kept or discarded. On the other hand, complexity is removed from the transit nodes, since no check to decide if a strict order packet should be discarded is needed there. No extra state information is required in the nodes, as only information already present in the topology image is used to perform the checks.

3.4.3 Selective packet discarding at the receiver

With the *Receiver* method, all packets that were sent from beyond a point of failure in an old topology context, are discarded. However, packets sent in an old topology context will not necessarily lead to reordering. Only if

packets keep arriving on the primary ringlet even after packets start arriving on the secondary ringlet, will reordering occur. The idea with *Selective Discard*, is to accept packets from beyond a point of failure even during context containment, as long as no packets from that sender has yet arrived on the secondary ringlet.

Selective Discard requires the receiver node to maintain one bit per node on the ring indicating whether a packet has been received on the secondary ringlet after the topology was updated. Once this bit has been set, no more strict order packets are accepted from beyond the point of failure on the primary ringlet.

Remember that the MAC client in a sender node can choose which ringlet to transmit on, to achieve load balancing or avoid congestion points. If a source node sends traffic to the same receiver on both ringlets, the performance gain obtained with *Selective Discard* can be reduced, since traffic can be received on the secondary ringlet immediately after a topology update. In Fig. 3.4, if the source S sends traffic over both the primary and the secondary ringlet before a failure, packets in transit along the secondary ringlet can start arriving immediately after the topology update. The *Selective Discard* method cannot distinguish the packets that were sent along the secondary path due to MAC client preferences, from the packets that were steered over to the secondary path by the protection mechanism. Only the last category of packets can cause reordering.

One way to improve the performance of the *Selective Discard* algorithm when the source S sends packets to the receiver R along both ringlets, would be to introduce a bit in the frame header marking the packets that are sent a non-default way due to a protection event. This would allow the receiver to distinguish the potentially reordered packets from the packets that were sent along the secondary ringlet due to MAC client preferences. With this improvement, the *Selective Discard* mechanism would give the optimal achievable performance for a reorder avoidance mechanism. Only packets that were actually received out of order would be discarded.

3.5 Evaluation

In this section we compare the different mechanisms described above. The measurements are again performed with our discrete event packet simulator based on the J-sim framework. We do our measurements on a ring with 64

nodes, numbered 0 – 63, and a link length of 40 km. The link capacity is 1 Gb/s, and the STQ buffer size is 256 kByte where not otherwise specified.

3.5.1 Optimal topology stabilization timer

In the *Automatic* method, (3.2) is used to calculate the timeout value of the topology stabilization timer. The calculated value is dependent on several variables, among them the size of the secondary transit queue. In Fig. 3.9, the calculated timer value is shown as a function of the STQ size, along with the maximum experienced STQ transit delay in the network in our simulations.

The traffic pattern is designed to stress the occupancy level of the STQ, to achieve a high transit delay. In the simulated scenario, node 31 is a "hot receiver". Nodes 0 to 29 send class C traffic to node 31 at the maximum rate allowed by the fairness mechanism. When the fairness mechanism has stabilized, all nodes start sending class A traffic to node 31 at the maximum rate ($rateA$ in (3.2)). 0.2 seconds later, the ring is broken between nodes 30 and 31. The simulations show that the highest STQ transit delay occurs at node 29, which is the bottleneck node in this scenario. The values plotted are the highest experienced values after running the simulation 100 times with different seeds.

The simulation results show that the experienced STQ transit delay never exceeds the value calculated in (3.2). In fact, the experienced delays are normally quite far from the theoretical maximum. This is because the chance of actually filling the whole STQ before the fairness mechanism reduces the sending rate of the class C sources is very small. This effect increases when the size of the STQ grows.

3.5.2 Comparison of packet loss counts

Figure 3.10 shows the total number of strict order packets lost in the network after a link failure. Results are given for the different packet reordering avoidance mechanisms, with increasing traffic load.

In this scenario, all nodes on the ring send traffic to random receivers. The source picks a random receiver, and sends a stream of packets to that receiver for a random (Pareto distributed) time interval. Each node has ten independent traffic sources, and may send to several receivers at the same time. The traffic is modelled this way in order to show self-similar characteristics [82]. The traffic load is controlled by varying the load produced by

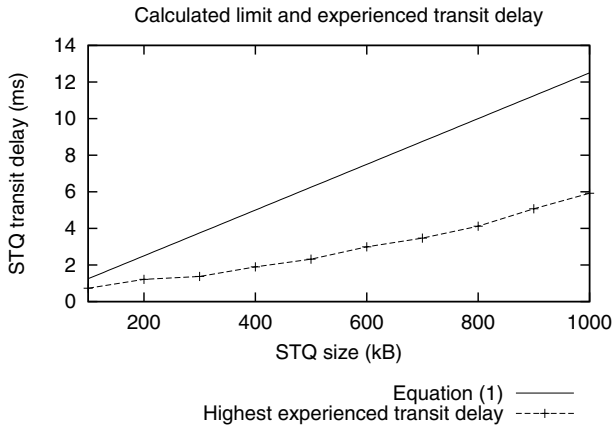


Figure 3.9: The maximum experienced transit queue delay in one node for different STQ sizes is plotted, along with the calculated maximum from (3.2).

the individual sub-sources. Traffic is always sent the default (shortest path) way around the ring.

The original RPR mechanism, using a default topology stabilization timer value of 40 ms, is shown in the upper graph. Optimizing the value of this timer gives a significant improvement, about 60% in our scenario, as seen by the second curve. *Receiver* and *Selective Discard* further reduces the experienced packet loss. The experienced reduction is between 80 and 90%. The difference between the *Receiver* and the *Selective Discard* methods is very small. In our scenario, the difference only appears with very high traffic load, and even then the difference is very limited (14 packets for load 4.0).

Note that since we do not use wrapping, packets in transit between the source and the failure before the topology image of the source node is updated will be lost with all mechanisms. These packets also contribute to the packet loss count in Fig. 3.10.

With the *Receiver* and *Selective Discard* mechanisms, the packet loss count for a specific traffic stream depends on the distance between the packet source and the point of failure. When the point of failure is close to the source, the time it takes for the topology update to reach the source is short. Thus, fewer packets are sent on the broken ringlet and lost. With increasing

Method	**Who discards packets?**	**Which packets are discarded?**
RPR default	Source and transit nodes	All strict order packets entering and leaving a node during the context containment period
Automatic	Source and transit nodes	All strict order packets entering and leaving a node during the calculated context containment period
Receiver	Receiver node	Packets sent from a node that cannot reach this receiver on the ringlet in question
Selective Discard	Receiver node	Packets sent from a node that cannot reach this receiver on the ringlet in question, if a packet has been received on the secondary ringlet

Table 3.1: Overview of the different reorder avoidance mechanisms.

distance from the source to the failure, more packets are sent on the wrong ringlet before the topology update can take place at the sender. This effect is not seen with the default RPR and the *Automatic* methods, since the context containment period is the same independent of where the failure occurs.

3.6 Summary

In this chapter, we have discussed the protection mechanisms used in the RPR standard. The wrapping and steering protection schemes have been discussed with respect to disruption time, packet reordering, and packet loss. We have discussed the different factors influencing the experienced disruption in connection with a failure on the ring.

Furthermore, we have identified the topology stabilization as the main obstacle that prevents sub 50 ms restoration for strict ordered traffic. We have presented three different improvements to the reorder avoidance mechanism.

The *Automatic* method seeks to minimize the period when packets are discarded, without compromising on the in order guarantee. This strategy

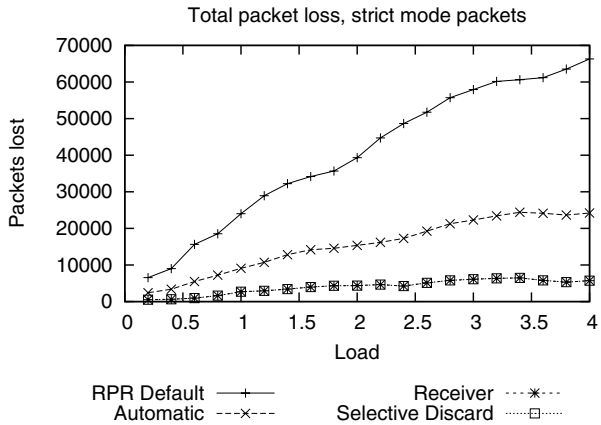


Figure 3.10: The total packet loss count in the network is plotted using the original RPR method, *Automatic*, *Receiver* and *Selective Discard*
.

implies setting the topology stabilization timer lower than the minimum value allowed in the RPR standard, but does not otherwise demand changes to the standard. In our simulated scenarios, packet loss is reduced by about 60%, compared to the default setting of the topology stabilization timer.

The other two methods move some logic from the transit path in RPR to the receiver station. The *Receiver* method does not require any state information to be maintained at the receiver, while the *Selective Discard* method requires the receiver to know which sources it receives packets from after a topology update. This is a small cost, but it requires an extra entry in the topology image of the RPR station.

As shown by the simulation results above, the difference between these methods with respect to packet loss is minimal, and can only be seen in situations with very high traffic load. These methods reduce the packet loss by almost 90% in our simulated scenarios. In a practical implementation, the *Receiver* method would probably be good enough, and it is unlikely that a vendor would implement the somewhat more complex *Selective Discard* method.

Chapter 4

Resilient Routing Layers

While in the previous chapter we focused on the protection mechanisms in one specific ring topology protocol, we now turn to the wider problem of protection in general packet switched networks. We introduce the concept of Resilient Routing Layers (RRL)[1]. RRL is a novel scheme for proactive recovery in packet switched networks.

Our main inspiration for this work is a layer-based approach used to obtain deadlock-free and fault-tolerant routing in irregular cluster networks based on a routing strategy called Up*/Down* [84]. Most of the contents in this chapter has previously been published in [83].

4.1 Introduction

There exist several deployed methods for fast protection at the physical and link layers, but at the networking layer recovery mechanisms have traditionally been *reactive* and *global*. When a failure is discovered, all the nodes in the network must be informed about the new topology situation before they can start the process of calculating new routing tables or paths, as explained in chapter 2. This process typically takes several seconds, which leads to unacceptable service degradations for real-time applications with strict delay bounds. Fast protection mechanisms at the networking layer have been developed [22, 23], but the methods that are deployed today are restricted

[1]To give a consistent presentation through this work, we will talk of *configurations* instead of the previously used *routing layers* [83] in our discussions. We have, however, opted to keep Resilient Routing Layers as a name for the concept described in this chapter.

to connection-oriented MPLS networks.

On this background we propose a simple, fast and flexible method for network layer recovery named Resilient Routing Layers (RRL). RRL is simple to understand and deploy, and offers a network administrator simple abstractions of the network after a failure. RRL is applicable to many types of networks, and can offer close to transparent resilience for both connectionless and connection-oriented networks.

The main idea behind RRL is to create a small set of connected, spanning sub-topologies termed *backup configurations*. The set of backup configurations is constructed so that each of the network elements we want to protect is *isolated* in at least one configuration, intuitively meaning that this configuration remains connected even if the network element fails. As will be shown later, this can be done using a surprisingly low number of configurations. In this work we focus on applying RRL for protection against link failures, but the RRL scheme can also be applied to protect against node failures [85].

In the remainder of this chapter, we first give an overview of the basic principles in RRL in section 4.2. We then present and discuss several algorithms for creating the backup configurations in section 4.3. In section 4.4, we present evaluation results concerning scalability, backup path length and the ability to recover from more than one failure, before we summarize in section 4.5.

4.2 RRL overview

RRL uses pre-calculated backup configurations to forward traffic along alternate routes after a link failure. Redundant Trees [29] and p-cycles [33], which were described in chapter 2, are examples of other protection schemes that use spanning sub-topologies to route around a failure. In contrast to the sub-topologies used in these schemes, the RRL backup configurations are not restricted to be cycles of trees, but can be general spanning graphs.

In RRL, backup configurations are created by removing links from the full topology, in such a way that every link is removed in at least one backup configuration. Thus, each backup configuration contains all nodes in the original network topology, but only a subset of the links. We say that a link is *isolated* in a backup configuration if the link is not present in that configuration. Many links can be isolated in the same backup configuration – the maximum number of links that can be removed in a configuration, is limited

by the invariant that all backup configurations must remain connected. In the discussions below, we say that a link can be removed from a topology unless it is an *articulation link*, meaning that removing the link would disconnect the topology. We speak of the backup configuration where a link a is isolated as the backup configuration of link a.

For an illustration of how a network topology can be covered by backup configurations, consider the example in figure 4.1. To the left, we have the full topology with six nodes and eleven links. To isolate every link in this topology, we must create backup configurations so that for each link, there is a configuration where the link is not present. Configuration 1 and configuration 2 in figure 4.1, makes an example of how all links in this topology can be isolated using only two backup configurations. Each backup configuration is a connected subgraph of the full topology, and every link is removed in (at least) one of the configurations. In configuration 2, we have removed the maximum number of links – no more links can be removed without partitioning the graph. Consequently, configuration 2 is a spanning tree. In configuration 1 on the other hand, we can still remove one more link while still keeping all nodes connected. Configuration 1 shows that, in general, the backup configurations are not cycle-free. It should be stressed that using configuration 1 and configuration 2 is not the only possible way to isolate every link in this example topology. In section 4.3, we return to the problem of how backup configurations should be constructed for a given topology.

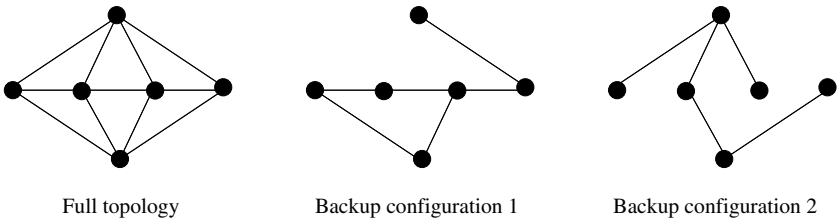


Figure 4.1: Example backup configurations

The configurations calculated in RRL are used as input to routing or path-finding algorithms that calculate a routing table or path table for each backup configuration, in addition to the normal full topology. Tables containing routing information for each backup configuration must be kept in

every node. For simplicity we say that traffic is forwarded in a backup configuration, meaning that the nodes on the path use the tables calculated for the backup configuration when the forwarding decisions are made.

In the failure free situation, RRL does not put any restrictions on the routing. When a failed link is detected, the nodes attached to the link start forwarding traffic that would normally go through the failed link in the backup configuration where the link is isolated. Packets forwarded in a backup configuration must be marked with a configuration identifier, so that the other nodes can keep forwarding it in the same configuration. This way, recovered traffic is forwarded from the point of failure to the destination (i.e. the egress node in the network) in the backup configuration where the failed link is isolated. Traffic that did not pass through the failed link is not affected, and is still routed in the original full topology. The decision on when to forward traffic in a backup configuration can be taken locally, which allows very fast reaction to a failure situation, without any signalling.

4.3 Configuration generation

An important choice when applying RRL is how to generate the backup configurations in an appropriate manner. The only restrictions imposed by the RRL framework, is that each backup configuration is a connected spanning subgraph of the network topology, and that all links are isolated in at least one configuration. These loose restrictions leave several degrees of freedom, and opens for tradeoffs between the amount of state (number of backup configurations), backup path lengths, multiple-fault resistance etc.

By using a small number of backup configurations, there will be less state that must be stored in each node, but this can give sparsely connected backup configurations with relatively long backup paths, as seen in figure 4.1. If we allow more backup configurations to be used, each configuration can have richer connectivity, giving more optimal routing after a failure. Using more configurations without increasing the connectivity in each configuration gives better resistance against multiple concurrent failures, since each link with a higher probability is isolated in more than one backup configuration.

In a practical deployment of RRL, the backup configurations can be designed manually or through an automated process. Manual design gives the network administrator greater flexibility and means to comply with special needs. For example, links that are particularly vulnerable or error prone,

could be isolated in a backup configuration with rich connectivity, so that the packet forwarding becomes more optimal in case of a failure. Similarly, paths between central nodes in the network can be made more resistant against multiple failures etc.

Sometimes, in particular for large networks, an algorithmic backup configuration generation will be preferred as a basis for the configuration design. An algorithm can be used to produce a specific number of backup configurations, with rich or sparse connectivity. In the following, we present three different backup configuration generation methods aimed at realizing quite different design goals. A formal algorithm is given for the first, while the two last are more briefly described. Tradeoffs between the amount of state, the backup path lengths and resistance against multiple failures are central in the choice of backup configuration generation algorithm.

4.3.1 Generating few configurations

RRL relies on keeping routing or forwarding information for each configuration in use. Hence, the amount of state needed in each node will in some way be dependent on the number of backup configurations used. In the *Minimum* algorithm presented below, we seek to isolate every link in a topology in exactly one backup configuration, using as few configurations as possible.

The *Minimum* algorithm works in two steps. First, it makes sure that all links in the topology are isolated in exactly one backup configuration, by creating one backup configuration at a time and isolating as many links as possible in each of them. The algorithm then iterates through the backup configurations created, and tries to "balance" them by moving links from the configuration containing the largest number of links to the configuration with the smallest number of links. This is done to avoid having one backup configuration with only very few links removed, and one with a large number of removed links. Such a situation would give poor backup path lengths in the backup configuration left with only a small number of links to route in.

Given a topology $G = (N, A)$, where N is the set of nodes and A is the set of links (arcs), the *Minimum* algorithm generates backup configurations as shown in algorithm 4.1. The input to the algorithm is the topology graph G, and the output is a set $\mathcal{C}$ of backup configurations. S denotes the set of links that have been isolated in a backup configuration, and the algorithm terminates when this set contains all the links in the topology.

Initially, all articulation links in G are added to set S (line 2). `artLinks`(G)

Algorithm 4.1: Minimum

```
1  C ← ∅
2  S ← artLinks(G)
3  while S ≠ A do
4      C_i ← G
5      A' ← A\S
6      forall a ∈ A' do
7          if a ∉ artLinks(C_i) then
8              C_i ← C_i\{a}
9              S ← S ∪ {a}
10         end
11     end
12     C ← C ∪ {C_i}
13     i++
14 end
15 repeat
16     C_max ← configuration with maximum|A|
17     C_min ← configuration with minimum|A|
18     δ ← |A|_max − |A|_min
19     move δ/2 links from C_max to C_min
20 until δ < 2
```

finds all articulation links in G, i.e. links that cannot be removed without partitioning G. These links can not be isolated without partitioning the backup configuration, and are therefore ignored by the algorithm. When a backup configuration is created, it is equal to G (line 4). We loop through all links that are not yet isolated, and isolate as many as possible in the current configuration. New configurations are created as needed until all links are isolated.

When all links are isolated, we proceed to balance the number of isolated links in each backup configuration (from line 15). It can trivially be shown that as long as $|A|_{max} > |A|_{min}$, it will always be possible to find a link that can be moved from C_{max} to C_{min}, since any configuration can be reduced to a spanning tree. The links to be removed must be selected so that C_{max} is still connected. This is done using a similar test as in line 7. We keep moving links from the largest to the smallest configuration until all configurations contain the same number of links.

For an arbitrary link in an arbitrary graph, it can be determined whether the link is an articulation link in $\mathcal{O}(|N| + |A|)$ time [86]. The running time of the first part of the algorithm is $\mathcal{O}(|\mathcal{C}| \cdot |A| \cdot (|N| + |A|))$, where $|\mathcal{C}|$ is the number of configurations needed. The number of configurations can never exceed the number of edges, and is usually much smaller. We show in the sequel that the number of configurations stays low even for large networks. The balancing part of our algorithm is less complex with regards to running time, so the total complexity, replacing $|\mathcal{C}|$ with $|A|$, will be $\mathcal{O}(|A|^3)$.

4.3.2 Improving Routing Efficiency

After a link failure, traffic that used to pass through the failed link is forwarded in the backup configuration where that link is isolated. Each backup configuration consists of a connected subgraph, containing all the nodes and some subset of the links in the original topology. The backup path lengths will depend on the structure and connectivity of the backup configurations.

To achieve shorter recovery paths, we can build more backup configurations with a richer connectivity than what is created with the *Minimum* algorithm. The purpose of the *Rich* algorithm is to create a chosen number of backup configurations, so that every link is isolated in exactly one configuration. By increasing the number of backup configurations, we can remove a smaller number of links in each of them, thus preserving more links that can take part in the packet forwarding after a failure. This way, shorter recovery

paths are traded for more state in the network nodes.

The *Rich* algorithm takes as input the number k of configurations that is attempted used. It then tries to remove an equal number of links from each configuration. If a link cannot be isolated in its intended backup configuration, we try to isolate it in one of the others, until all configurations are tried. The output of the algorithm is k configurations, each containing approximately $|A|(1-(1/k))$ links, where $|A|$ is the number of links in the original topology. If all links in the topology cannot be isolated using k configurations, the *Rich* algorithm will give up and terminate. The running time of this algorithm is the same as for the *Minimum* algorithm ($\mathcal{O}(|A|^3)$).

4.3.3 Resisting multiple failures

RRL recovers traffic on a failed link by forwarding packets in the backup configuration of the failed link. If more than one link fails and all the failing links are isolated in the same backup configuration, all the affected traffic will trivially be recovered. If the failed links are isolated in different configurations, RRL can not guarantee that this traffic is recovered. However, traffic can often still be saved, as long as the traffic is not routed through more than one failure. When two failing links are not isolated in the same configuration, the routing process decides what traffic can be recovered. Because RRL is agnostic with respect to routing, we only say that we can protect against more than one link failure when the links in question are isolated in the same configuration.

To achieve increased resistance against multiple failures, we introduce the *Sparse* algorithm. Like the *Rich* algorithm above, this algorithm creates a fixed number of backup configurations. But instead of isolating each link in exactly one configuration, the *Sparse* algorithm makes the configurations as sparse as possible, by removing the maximum number of links in each backup configuration. In effect, each configuration becomes a spanning tree.

In the *Sparse* algorithm, we make an effort to maximize the probability that any two (in general any k) links have a common backup configuration where they are both isolated. The links in the original topology are split into different groups, and combinations of links from different groups are isolated in the same backup configuration. The number of configurations used by the *Sparse* algorithm depends on how many simultaneous failures we want to protect against, and the connectivity of the network topology. Protecting against more simultaneous failures demands more configurations, while a

richer original connectivity allows us to reduce the number of configurations.

Note that the *Sparse* algorithm is designed to maximize the probability that two simultaneous link failures can be handled, not to make guarantees that this is the case. As shown in section 4.4.4, the *Sparse* algorithm will not protect against two simultaneous failures for all combinations of links.

Due to the splitting into several groups, the complexity of this algorithm is somewhat higher than for the others, and the running time is $\mathcal{O}(|A|^4)$.

4.4 Evaluation

A protection scheme incurs additional network state overhead. In order for a scheme to show satisfactory scalability properties, this overhead should not grow proportionally with the network size. In RRL, the state overhead is in the worst case proportional to the number of configurations. Therefore, it is important to show that the number of configurations remains limited for large network topologies.

Backup paths are calculated by the routing algorithm in the configuration that protects the failed link. With shortest path routing, the backup paths will usually be longer than the original path. A good protection scheme should however result in backup paths of acceptable lengths.

Two simultaneous link failures can be tolerated in our recovery scheme if there exists a backup configuration where both links are isolated. Hence, it is an advantage for our protection scheme if most link pairs have a backup configuration where they are both isolated.

For our quantitative evaluation, we focus on these three central metrics for protection schemes: scalability, backup path lengths, and resistance against two simultaneous failures. We first describe the evaluation method, and then present and discuss results for each of the metrics.

4.4.1 Method

We evaluate the performance of the *Minimal*, *Rich* and *Sparse* algorithms for the three above mentioned metrics. We use a wide range of relevant topologies, both real and synthetic. For our evaluations, we have implemented the above mentioned algorithms in a Java a software model. The model is used to calculate configurations and backup path characteristics for the evaluated topologies.

In section 4.4.3, we measure the increased path length that the affected traffic experiences after the failure. For the purpose of this evaluation, we use shortest path routing. We measure the *local* recovery path length given by RRL, meaning the path length from the point of failure to the destination in the backup configuration of the failed link. For reference, we also measure the *optimal* recovery path between the failure and the destination, meaning the shortest path calculated in a topology where only the broken link is removed. This is the shortest possible local backup path that can be found for the given failure.

When evaluating the resilience against two simultaneous failures, we measure how often there exists a backup configuration where two given links are isolated. If such a configuration exists, all traffic in the network can be recovered if the two links fail simultaneously. In the cases where no such configuration can be found, traffic originally passing through only one of the failing links can often still be recovered, as long as the backup path does not pass through the second failure. This is further discussed in [87].

Topologies

Both real and synthetic topologies are used in our evaluations. Real topologies provide performance indications in real-world settings, while the synthetic ones are important for generality, scalability and statistical relevance.

The real topologies are gathered from the Rocketfuel [88] database of inferred real network topologies. The topologies used are intra-ISP topologies on a POP level. We study only failures that occur on the biconnected portions of the topologies.

We used the BRITE [89] topology generator to generate synthetic topologies with different characteristics. However, the choice of the generation model was not obvious. It has been shown that most available power law topology models do not offer enough flexibility on how nodes are connected [90]. The Generalized Linear Preference (GLP) model is therefore proposed, adding some flexibility. However, GLP has been shown to have weaknesses in the node degree distribution of the topologies generated [91]. The Waxman model [92] describes another way of constructing synthetic topologies, giving less central hubs in the topology. However the Waxman implementation in BRITE limits the flexibility on the links-to-nodes ratio. Based on the limitations of both the GLP and the Waxman models, we have chosen to use topologies from both models for our evaluation.

4.4.2 Scalability

We present the number of configurations needed to protect real and synthetically generated topologies, as given by the *Minimum* algorithm.

Real topologies

The number of configurations needed to cover selected real topologies, using the *Minimum* algorithm, is given in table 4.1.

AS number	Nodes	Links	Configurations
4513	5	5	5
13129	5	7	3
2497	8	18	2
4565	12	17	5
11537	15	24	3
16631	20	30	4
1239	32	64	4
3320	68	353	2

Table 4.1: Number of configurations needed

We note that the number of configurations required is never above five in our topologies. This indicates good scalability properties, which are further tested for synthetically generated topologies below.

We see that the number of configurations required seems to depend mainly on the connectivity of the topology, i.e. the links-to-node ratio. When the connectivity is sparse (like in AS 4513, which is a ring), we need a larger number of configurations even for small topologies. Richer connectivity allows us to protect every link using a smaller number of configurations.

Synthetic topologies

Figure 4.2 shows the number of configurations needed to protect topologies with different size and connectivity, as given by the *Minimum* algorithm. Six of the bars show topologies generated with the Waxman model, with a links-to-node degree of two and three respectively. The bars for the GLP model has a ratio of about 1.8 links per node. Distributions are shown for networks

of three different sizes (32, 128, 512 nodes). The notation used on the x-axis, shows which model is used (Waxman or GLP), the number of nodes in the network, and the link-to-node ratio. The results shown are computed using 100 different topologies with the wanted characteristics.

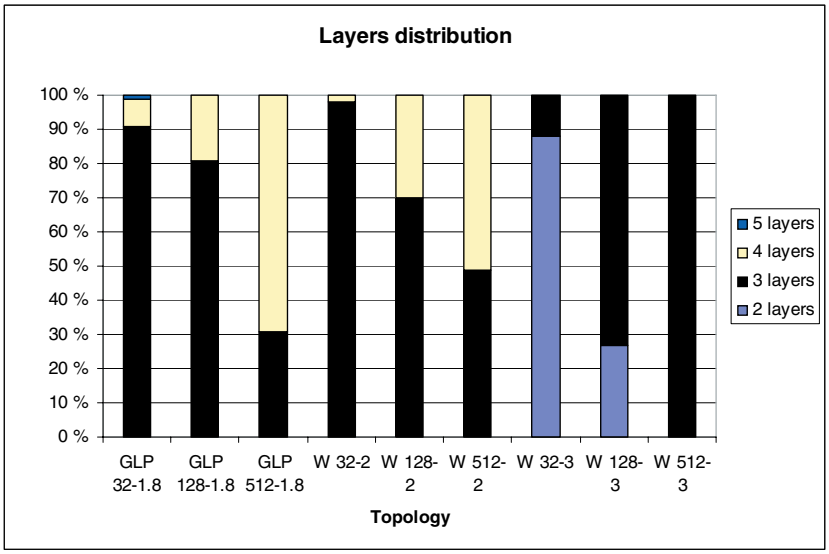


Figure 4.2: Minimum number of configurations needed

Figure 4.2 confirms that the most important factor determining how many configurations are needed is the link-to-node ratio. The number of configurations needed is modest, even for large topologies, but always lower for topologies with higher connectivity.

4.4.3 Backup path lengths

We present the distribution of recovery path lengths in a 32 node network, generated using the GLP model with a link-to-node ratio of 1.8, for the *Minimum*, the *Rich*, and the *Sparse* algorithms. The recovery path lengths are calculated by failing one link at a time, and then measuring the path

length from the point of failure to the destination. Tests were also performed on Waxman-generated topologies, with similar results.

Figure 4.3 shows that both the *Rich* and the *Minimum* algorithms achieve significantly shorter backup path lengths than the *Sparse* algorithm. This illustrates that the ability of the *Sparse* algorithm to resist more than one link failure by reducing each backup configuration to a spanning tree, comes at the cost of less optimal routing of recovered traffic.

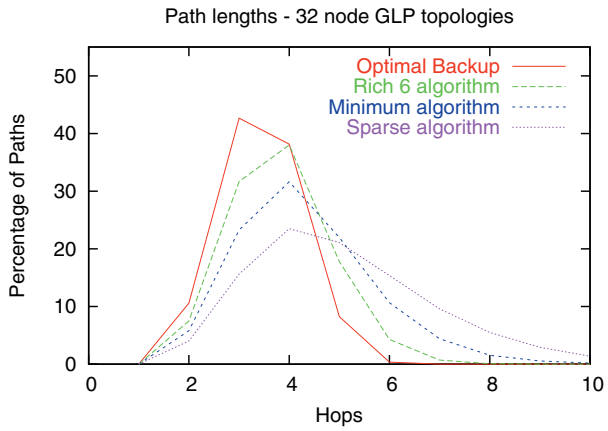


Figure 4.3: Recovery path length distributions

The shortest recovery paths are achieved using the *Rich* algorithm, creating six backup configurations. This is as expected, since spreading the isolated links over six backup configurations, allows us to keep a richer connectivity in each configuration. The *Minimum* algorithm creates three backup configurations for the topology used here. Using only three backup configurations gives more sparse connectivity in each configuration, and thus increases the backup path length.

The tradeoff between backup path length and the number of configurations used is also illustrated in figure 4.4. This figure shows the average recovery path length using the *Rich* algorithm for a 32 node topology created using the GLP model. We see that if we allow the use of more configurations, the average recovery path length approaches the best achievable. Again, very

similar results were obtained when evaluating topologies generated with the Waxman model.

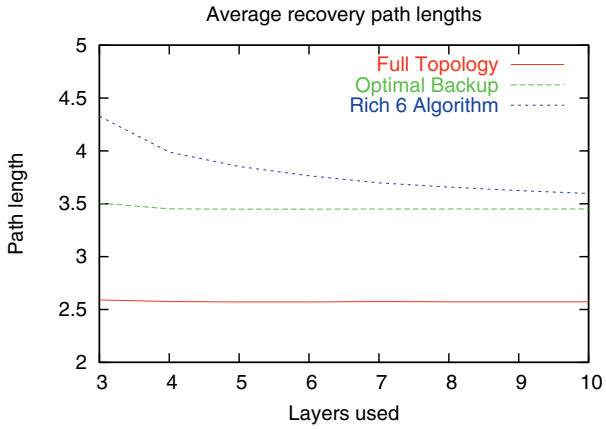


Figure 4.4: Recovery path length vs configurations used

4.4.4 Resisting more than one failure

We measure the ability of our recovery scheme to handle two simultaneous failures. In our evaluation, we look at networks with 32 and 128 nodes, and with a link-to-node ratio two and three. RRL is said to handle two simultaneous failures only when there exists a single backup configuration that contains both failed links. As mentioned above, traffic can often be recovered even if this is not the case, depending on the routing in the network.

The network topologies are generated using the Waxman model as discussed above. We have also evaluated GLP topologies, with similar results. Table 4.2 shows that the *Sparse* algorithm with six configurations provides almost 100 % resistance for two link failures when the link-to-node is 3. When this ratio is 2, the dual link failure resistance falls to about 85 %. Again, the results shown are averages from 100 generated topologies.

Method	Nodes	Links/node ratio	p
Sparse 6	32	2	0.860
Sparse 6	128	2	0.847
Sparse 6	32	3	0.995
Sparse 6	128	3	0.992

Table 4.2: Probability of two failure tolerance

4.5 Summary

In this chapter, we have presented Resilient Routing Layers as an approach for providing fast recovery from link failures at the network layer. RRL is based on computing fully connected sub-topologies called backup configurations. These backup configurations can be created manually or automatically. We have presented algorithms that create backup configurations optimized for minimizing the amount of state, reducing backup path lengths, and resisting dual link failures.

RRL does not influence the routing in the failure free situation. When a link failure occurs, packets are switched to the backup configuration where the failed link is isolated, and are routed according to the backup configuration to the destination. Traffic that does not pass through the failed link, is not affected by the failure.

Although RRL gives recovery path lengths longer than the minimum achievable, we have shown that the added path lengths typically consist of a few hops. Using backup configurations with a richer connectivity, we have demonstrated that RRL gives close to optimal recovery path lengths. We have also shown that RRL is scalable. In our evaluation using real and synthetically generated topologies, we found that we never needed more than 5 backup configurations to protect all links.

Chapter 5

Multiple Routing Configurations

The previous chapter explained how RRL can be used to give fast recovery from link failures in general packet networks. In this chapter we will extend the concepts introduced in RRL, and tailor them to be used in an intra-domain IP network running a connectionless link-state routing protocol like OSPF or IS-IS. The extended method is given the name Multiple Routing Configurations (MRC).

Like RRL, MRC is based on constructing a small number of backup configurations, and to use these to prepare alternate routes that avoid a failed component. However, instead of removing links in the backup configurations, MRC restricts the routing by strategic link weight assignment in each configuration. In addition to link failures, MRC also guarantees fast recovery from any single node failure in the network. Therefore, MRC is somewhat more complex than RRL. Also, being designed with only shortest path connectionless IGP routing in mind, MRC is more specific than RRL. The increased complexity makes it necessary to use a more formal language when we describe MRC than we have done so far.

Most of the contents in this chapter has previously been published in [93].

5.1 Introduction

MRC is a local protection scheme that is designed to guarantee fast recovery from any single link or node failure in arbitrary biconnected networks. Single

failures constitute a large majority of the component failures experienced in a network [12]. MRC assumes that the network uses shortest path routing and destination based hop-by-hop forwarding. With MRC, packet forwarding can continue over pre-configured alternative next-hops immediately after the detection of the failure. Using MRC as a first line of defense against network failures, the normal IP convergence process can be put on hold. This process is then initiated only as a consequence of non-transient failures. Since no global re-routing is performed, fast failure detection mechanisms like fast hellos or hardware alerts can be used to trigger MRC without compromising network stability [57].

In the literature, it is sometimes claimed that node failure recovery implicitly gives recovery also from link failures, since the adjacent links of the failed node can be avoided. This is true for intermediate nodes, but the egress node in a network domain must still be reachable if the link in the last hop fails ("The last hop problem", [13]). MRC solves the last hop problem by strategic assignment of link weights between the backup configurations.

MRC has a range of attractive features:

- It provides almost continuous forwarding of packets in the case of a failure. The router that detects the failure initiates a local rerouting immediately, without communicating with the surrounding neighbors.

- MRC helps improve network availability by allowing suppression of the re-convergence process. Delaying this process is useful to address transient failures, and pays off under many scenarios [57]. Suppression of the re-convergence process is further actualized by the evidence that a large proportion of network failures is short-lived, often lasting less than a minute [12].

- MRC uses a single mechanism to handle both link and node failures. Failures are handled locally by the detecting node, and MRC always finds a route to the destination (if operational). MRC makes no assumptions with respect to the *root cause of failure*, e.g., whether the packet forwarding is disrupted due to a failed link or a failed router.

- An MRC implementation can be made without major modifications to existing IGP routing standards. IETF recently initiated specifications of multi-topology routing for OSPF and IS-IS, and this approach seems well suited to implement our proposed backup configurations [94, 95].

- Link weights in MRC backup configurations are set independently from the normal link weights. The load of the recovery traffic can thus be balanced to reduce the danger of congestion without affecting the failure-free case.

The rest of this chapter is organized as follows. In section 5.2 we describe the basic concepts and functionality of MRC. We then define MRC formally and present an algorithm used to create the needed backup configurations in section 5.3. In section 5.4, we explain how the generated configurations can be used to forward the traffic safely to its destination in case of a failure. We present performance evaluations of the proposed method in section 5.5, before we summarize this chapter in section 5.6.

5.2 MRC Overview

MRC is based on building a small set of backup routing configurations, that are used to route recovered traffic on alternate paths after a failure. The backup configurations differ from the normal routing configuration in that link weights are set so as to avoid routing traffic in certain parts of the network. We observe that if all links attached to a node are given sufficiently high link weights, traffic will never be routed through that node. The failure of that node will then only affect traffic that is sourced at or destined for the node itself. Similarly, to exclude a link from taking part in the routing, we give it infinite weight. The link can then fail without any consequences for the traffic.

Our MRC approach is threefold. First, we create a set of backup configurations, so that every network component is excluded from packet forwarding in one configuration. Second, for each configuration, a standard routing algorithm like OSPF is used to calculate configuration specific shortest paths and create forwarding tables in each router, based on the configurations. The use of a standard routing algorithm guarantees loop-free forwarding within one configuration. Finally, we design a forwarding process that takes advantage of the backup configurations to provide fast recovery from a component failure.

In our approach, we construct the backup configurations so that for all links and nodes in the network, there is a configuration where that link or node is not used to forward traffic. Thus, for any single link or node failure,

there will exist a configuration that will route the traffic to its destination on a path that avoids the failed element. Also, the backup configurations must be constructed so that all nodes are reachable in all configurations, i.e., there is a valid path with a finite cost between each node pair. We formally describe MRC and the configuration generation in section 5.3

Using a standard shortest path calculation, each router creates a set of configuration-specific forwarding tables. For simplicity, we say that a packet is forwarded according to a configuration, meaning that it is forwarded using the forwarding table calculated based on that configuration. In this chapter we talk about building a separate forwarding table for each configuration, but we believe that more efficient solutions can be found in a practical implementation.

When a router detects that a neighbor can no longer be reached through one of its interfaces, it does not immediately inform the rest of the network about the connectivity failure. Instead, packets that would normally be forwarded over the failed interface are marked as belonging to a backup configuration, and forwarded on an alternative interface towards its destination. The selection of the correct backup configuration, and thus also the backup next-hop, is detailed in section 5.4. The packets must be marked with a configuration identifier, so the routers along the path know which configuration to use. Packet marking is most easily done by using specific values in the DSCP field in the IP header. If this is not possible, other packet marking strategies like IPv6 extension headers or using a private address space and tunneling (as proposed in [61]) could be used.

It is important to stress that MRC does not affect the failure-free original routing, i.e., when there is no failure, all packets are forwarded according to the original configuration, where all link weights are normal. Upon detection of a failure, only traffic reaching the failure will switch configuration. All other traffic is forwarded according to the original configuration as normal.

5.3 Generating Backup Configurations

In this section, we will first detail the requirements that must be put on the backup configurations used in MRC. Then, we propose an algorithm that can be used to automatically create such configurations. The algorithm will typically be run once at the initial startup of the network, and each time a node or link is permanently added or removed. We use the notation shown

Table 5.1: Notation

$G = (N, A)$	Graph comprising nodes N and directed links (arcs) A
C_i	The graph with link weights as in configuration i
S_i	The set of isolated nodes in configuration C_i
B_i	The backbone in configuration C_i
$A(u)$	The set of links from node u
(u, v)	The directed link from node u to node v
$p_i(u, v)$	A given shortest path between nodes u and v in C_i
$\mathcal{N}(p)$	The nodes on path p
$\mathcal{A}(p)$	The links on path p
$w_i(u, v)$	The weight of link (u, v) in configuration C_i
$w_i(p)$	The total weight of the links in path p in configuration C_i
w_{r}	The weight of a restricted link
n	The number of backup configurations

in table 5.1.

5.3.1 Configurations Structure

MRC configurations are defined by the network topology, which is the same in all configurations, and the associated link weights, which differ among configurations. We formally represent the network topology as a graph $G = (N, A)$, with a set of nodes N and a set of unidirectional links (arcs) A[1]. In order to guarantee single-fault tolerance, the topology graph G must be biconnected. A configuration is defined by this topology graph and the associated link weight function:

Definition. A *configuration* C_i is an ordered pair (G, w_i) of the graph G and a function $w_i : A \rightarrow \{1, \ldots, w_{\max}, w_{\mathrm{r}}, \infty\}$ that assigns an integer weight $w_i(a)$ to each link $a \in A$.

We distinguish between the normal configuration C_0 and the backup configurations $C_i, i > 0$. In the normal configuration, C_0, all links have "normal" weights $w_0(a) \in \{1, \ldots, w_{\max}\}$. We assume that C_0 is given with finite integer weights. MRC is agnostic to the setting of the link weights in C_0. In

[1]We interchangeably use the notations a or (u, v) to denote a link, depending on whether the endpoints of the link are important.

the backup configurations, selected links and nodes must not carry any transit traffic. Still, traffic must be able to depart from and reach all operative nodes. These traffic regulations are imposed by assigning high weights to some links in the backup configurations:

Definition. A link $a \in A$ is *isolated* in C_i if $w_i(a) = \infty$.

Definition. A link $a \in A$ is *restricted* in C_i if $w_i(a) = w_{\mathrm{r}}$.

Isolated links do not carry any traffic. Restricted links are used to isolate nodes from traffic forwarding. The restricted link weight w_{r} must be set to a sufficiently high, finite value to achieve that. Nodes are isolated by assigning at least the restricted link weight to all their attached links. For a node to be reachable, we cannot isolate all links attached to the node in the same configuration. More than one node may be isolated in a configuration. The set of isolated nodes in C_i is denoted S_i, and the set of normal (non-isolated) nodes $\overline{S}_i = N \setminus S_i$.

Definition. A node $u \in N$ is *isolated* in C_i if

$$\begin{aligned} & \forall (u,v) \in A(u), w_i(u,v) \geq w_{\mathrm{r}} \\ \wedge \quad & \exists (u,v) \in A(u), w_i(u,v) = w_{\mathrm{r}} \end{aligned} \tag{5.1}$$

With MRC, restricted and isolated links are always attached to isolated nodes as given by the following rules. For all links $(u,v) \in A$,

$$w_i(u,v) = w_{\mathrm{r}} \Rightarrow (u \in S_i \wedge v \in \overline{S}_i) \vee (v \in S_i \wedge u \in \overline{S}_i) \tag{5.2}$$

$$w_i(u,v) = \infty \Rightarrow u \in S_i \vee v \in S_i \tag{5.3}$$

This means that a restricted link always connects an isolated node to a non-isolated node. An isolated link either connects an isolated node to a non-isolated node, or it connects two isolated nodes. Importantly, this means that a link is always isolated in the same configuration as at least one of its attached nodes. These two rules are required by the MRC forwarding process described in section 5.4 in order to give correct forwarding without knowing the root cause of failure. When we talk of a backup configuration, we refer to a configuration that adheres to (5.2) and (5.3).

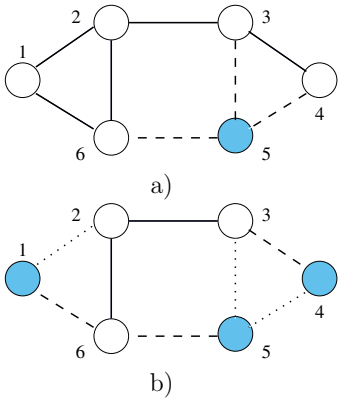


Figure 5.1: a) Node 5 is isolated (shaded color) by setting a high weight on all its connected links (stapled). Only traffic to and from the isolated node will use these restricted links. b) A configuration where nodes 1, 4 and 5, and the links 1-2, 3-5 and 4-5 are isolated (dotted).

The purpose of the restricted links is to isolate a node from routing in a specific backup configuration C_i, such as node 5 in figure 5.1a). In many topologies, more than a single node can be isolated simultaneously. In the example in figure 5.1b) three nodes and three links are isolated.

Restricted and isolated links are always given the same weight in both directions. However, MRC treats links as unidirectional, and makes no assumptions with respect to symmetric link weights for the links that are not restricted or isolated. Hence, MRC can co-exist with traffic engineering schemes that rely on asymmetric link weights for load balancing purposes.

MRC guarantees single-fault tolerance by isolating each link and node in exactly one backup configuration. In each configuration, all node pairs must be connected by a finite cost path that does not pass through an isolated node or an isolated link. A configuration that satisfies this requirement is called *valid*:

Definition. A configuration C_i is *valid* if and only if

$$\begin{aligned} \forall u, v \in N: \quad & \mathcal{N}(p_i(u,v)) \setminus (\overline{S}_i \cup \{u,v\}) = \emptyset \\ & \wedge \quad w_i(p_i(u,v)) < \infty \end{aligned} \tag{5.4}$$

We observe that all backup configurations retain a characteristic internal structure, in that all isolated nodes are directly connected to a core of nodes connected by links with normal weights:

Definition. A configuration *backbone* $B_i = (\overline{S}_i, A_i)$, $A_i \subseteq A$ consists of all non-isolated nodes in C_i and all links that are neither isolated nor restricted:

$$a \in A_i \Leftrightarrow w_i(a) \leq w_{\max} \tag{5.5}$$

A backbone is connected if all nodes in $\overline{S}_i$ are connected by paths containing links with normal weights only:

Definition. A backbone B_i is *connected* if and only if

$$\forall u, v \in B_i : a \in \mathcal{A}(p_i(u, v)) \Rightarrow w(a) \leq w_{\max} \tag{5.6}$$

An important invariant in our algorithm for creating backup configurations is that the backbone remains connected. Since all backup configurations must adhere to (5.2) and (5.3), we can show that a backup configuration with a connected backbone is equivalent to a valid backup configuration:

Lemma 5.3.1. *A backup configuration C_i is valid if and only if it contains a connected backbone.*

Proof. We first show that a connected backbone implies that C_i is valid. For each node pair u and v, zero, one or both of u and v are in S_i. Assume $u \in S_i \wedge v \in S_i$. From the definition of an isolated node and (5.2), $\exists u', v' \in \overline{S}_i$: $w_i(u, u') = w_{\mathrm{r}} \wedge w_i(v, v') = w_{\mathrm{r}}$. From (5.6) $a \in \mathcal{A}(p_i(u', v')) \Rightarrow w(a) \leq w_{\max}$. Thus,

$$w_i(p_i(u, v)) \leq 2w_{\mathrm{r}} + w_i(p_i(u', v')) < \infty \tag{5.7}$$

$$\mathcal{N}(p_i(u, v)) \setminus (\overline{S}_i \cup \{u, v\}) = \emptyset \tag{5.8}$$

and (5.4) follows. A subset of the above is sufficient to show the same if only one, or none, of u, v is in S_i.

For the converse implication, assume $u, v \in \overline{S}_i$ and node $x \in \mathcal{N}(p_i(u, v))$. From (5.4), $x \in \overline{S}_i$ and $w_i(p_i(u, v)) < \infty$. Since by (5.2) restricted links are always connected to at least one isolated node, such links can not be part of $\mathcal{A}(p_i(u, v))$, and all links in $\mathcal{A}(p_i(u, v))$ must have normal weights. □

In backup configurations, transit traffic is constrained to the configuration backbone. A restricted link weight w_{r} that is sufficiently high to achieve this can be determined from the number of links in the network and the maximal normal link weight:

Proposition 5.3.2. *Let x be a node isolated in the valid backup configuration C_i. Then, restricted link weight value*

$$w_{\mathrm{r}} = |A| \cdot w_{\max} \tag{5.9}$$

is sufficiently high to exclude x from any shortest path in C_i which does not start or end in x.

Proof. Since all links attached to the isolated node x have a weight of at least w_{r}, the weight of a path through x will be at least $2 \cdot w_{\mathrm{r}} = 2 \cdot |A| \cdot w_{\max}$. From the definition of an isolated node and (5.2), all isolated nodes are directly connected to the configuration backbone. From (5.4), any shortest path in C_i will be entirely contained in B_i, except possibly the first or the last hop. A valid configuration contains a connected backbone, and the total weight of the sub-path that is within B_i will be at most $|A_i| \cdot w_{\max}$. Since $|A_i| < 2|A|$, no shortest path will include x as the transit. □

To guarantee recovery after any component failure, every node and every link must be isolated in one backup configuration. Let $\mathcal{C} = \{C_1, ...C_n\}$ be a set of backup configurations. We say that

Definition. A set, $\mathcal{C}$, of backup configurations is *complete* if

$$\begin{aligned} & \forall a \in A, \exists C_i \in \mathcal{C} : w_i(a) = \infty \\ \wedge \quad & \forall u \in N, \exists C_i \in \mathcal{C} : u \in S_i \end{aligned} \tag{5.10}$$

A complete set of valid backup configurations for a given topology can be constructed in different ways. In the next subsection we present an efficient algorithm for this purpose.

5.3.2 Algorithm

The number and internal structure of backup configurations in a complete set for a given topology may vary depending on the construction model. If

more configurations are created, fewer links and nodes need to be isolated per configuration, giving a richer (more connected) backbone in each configuration. On the other hand, if fewer configurations are constructed, the state requirement for the backup routing information storage is reduced. However, calculating the minimum number of configurations for a given topology graph is computationally demanding. One solution would be to find all valid configurations for the input consisting of the topology graph G and its associated normal link weights w_0, and then find the complete set of configurations with lowest cardinality. Finding this set would involve solving the Set Cover problem, which is known to be NP-complete [96].

Instead we present a heuristic algorithm that attempts to make all nodes and links in an arbitrary biconnected topology isolated. Our algorithm takes as input the directed graph G, its associated normal link weights w_0, and the number n of backup configurations that is intended created. If the algorithm terminates successfully, its output is a complete set of valid backup configurations. For a sufficiently high n, the algorithm will always terminate successfully, as will be further discussed below.

Description

Algorithm 5.1 loops through all nodes in the topology, and tries to isolate them one at a time. A link is isolated in the same iteration as one of its attached nodes. The algorithm terminates when either all nodes and links in the network are isolated in exactly one configuration, or a node that cannot be isolated is encountered. We now specify the algorithm in detail, using the notation shown in table 5.1.

Main loop Initially, n backup configurations are created as copies of the normal configuration. A queue of nodes (Q_{n}) and a queue of links (Q_{a}) are initiated. The node queue contains all nodes in an arbitrary sequence. The link queue is initially empty, but all links in the network will have to pass through it. Method `first` returns the first item in the queue, removing it from the queue.

When a node u is attempted isolated in a backup configuration C_i, it is first tested that doing so will not disconnect B_i according to definition (5.6). The `connected` method at line 13 decides this by testing that each of u's neighbors can reach each other without passing through u, an isolated node, or an isolated link in configuration C_i.

Algorithm 5.1: Creating backup configurations.

```
1  for i ∈ {1...n} do
2      C_i ← (G, w_0)
3      S_i ← ∅
4      B_i ← C_i
5  end
6  Q_n ← N
7  Q_a ← ∅
8  i ← 1
9  while Q_n ≠ ∅ do
10     u ← first (Q_n)
11     j ← i
12     repeat
13         if connected(B_i \ ({u}, A(u))) then
14             C_tmp ← isolate(C_i, u)
15             if C_tmp ≠ null then
16                 C_i ← C_tmp
17                 S_i ← S_i ∪ {u}
18                 B_i ← B_i \ ({u}, A(u))
19         i ← (i mod n) + 1
20     until u ∈ S_i or i=j
21     if u ∉ S_i then
22         Give up and abort
23 end
```

If the connectivity test is positive, function `isolate` is called, which attempts to find a valid assignment of isolated and restricted links for node u as detailed below. If successful, `isolate` returns the modified configuration and the changes are committed (line 16). Otherwise it returns `null`, and no changes are made in C_i.

If u was successfully isolated, we move on to the next node. Otherwise, we keep trying to isolate u in every configuration, until all n configurations are tried (line 20). If u could not be isolated in any configuration, a complete set of valid configurations with cardinality n could not be built using our algorithm. The algorithm will then terminate with an unsuccessful result (line 22).

Function `isolate`(C_i, u)

```
1  Q_a ← Q_a + (u,v), ∀(u,v) ∈ A(u)
2  while Q_a ≠ ∅ do
3      (u,v) ← first (Q_a)
4      if ∃j : v ∈ S_j then
5          if w_j(u,v) = w_r then
6              if ∃(u,x) ∈ A(u)\(u,v) : w_i(u,x) ≠ ∞ then
7                  w_i(u,v) ← w_i(v,u) ← ∞
8              else
9                  return null
10         else if w_j(u,v) = ∞ and i ≠ j then
11             w_i(u,v) ← w_i(v,u) ← w_r
12     else
13         if ∃(u,x) ∈ A(u)\(u,v) : w_i(u,x) ≠ ∞ then
14             w_i(u,v) ← w_i(v,u) ← ∞
15         else
16             w_i(u,v) ← w_i(v,u) ← w_r
17             Q_n ← v + (Q_n \ v)
18             Q_a ← (v,u)
19 end
20 return C_i
```

Isolating links Along with u, as many as possible of its attached links are isolated. The algorithm runs through the links $A(u)$ attached to u (lines 2-3

in function `isolate`). It can be shown that it is an invariant in our algorithm that in line 1, all links in Q_a are attached to node u. The node v in the other end of the link may or may not be isolated in some configuration already (line 4). If it is, we must decide whether the link should be isolated along with u (line 7), or if it is already isolated in the configuration where v is isolated (line 11). A link must always be isolated in the same configuration as one of its end nodes. Hence, if the link was not isolated in the same configuration as v, it *must* be isolated along with node u.

Before we can isolate the link along with u, we must test (line 6) that u will still have an attached non-isolated link, in accordance to the definition of isolated nodes. If this is not the case, u can not be isolated in the present configuration (line 9).

In the case that the neighbor node v was *not* isolated in any configuration (line 12), we isolate the link along with u if there exists another link not isolated with u (line 14). If the link can not be isolated together with node u, we leave it for node v to isolate it later. To make sure that this link can be isolated along with v, we must process v next (line 17, selected at line 10 in algorithm 5.1), and link (v, u) must be the first among the links originating from node v to be processed (line 18, selected at line 2).

Output

We show that successful execution of algorithm 5.1 results in a complete set of valid backup configurations.

Proposition 5.3.3. *If algorithm5.1 terminates successfully, the produced backup configurations adhere to (5.2) and (5.3).*

Proof. A link is only given weight w_r or ∞ in the process of isolating one of its attached nodes, and (5.3) follows. For restricted links, (5.2) requires that only one of the attached nodes are isolated. This invariant is maintained in line 7 in function `isolate` by demanding that if a node attached to a restricted link is attempted isolated, the link must also be isolated. Hence it is impossible to isolate two neighbor nodes without also isolating their connecting link, and (5.2) follows. □

Proposition 5.3.4. *If algorithm5.1 terminates successfully, the backup configurations set $\mathcal{C} = \{C_1, C_2, \ldots, C_n\}$ is complete, and all configurations $C_i \in \mathcal{C}$ are valid.*

Proof. Initially, all links in all configurations have original link weights. Each time a new node and its connected links are isolated in a configuration C_i we verify that the backbone in that configuration remains connected. When the links are isolated, it is checked that the node has at least one neighbor not isolated in C_i (line 14 in function `isolate`). When isolating a node, we also isolate as many as possible of the connected links. A link is always isolated in the same configuration as one of its attached nodes. If this is not possible, the node is not isolated (`isolate`, line 9). From Lemma 5.3.1, the altered configuration remains valid.

The algorithm runs through all nodes. If one node cannot be isolated, the algorithm aborts (line 22 in algorithm 5.1). If it does terminate with success, all nodes and links are isolated in one configuration, thus the configuration set is complete. □

Termination

The algorithm runs through all nodes trying to make them isolated in one of the backup configurations and will always terminate with or without success. If a node cannot be isolated in any of the configurations, the algorithm terminates without success. However, the algorithm is designed so that any biconnected topology will result in a successful termination, if the number of configurations allowed is sufficiently high.

Proposition 5.3.5. *Given a biconnected graph $G = (N, A)$, there will exist $n \leq |N|$, so that algorithm5.1 will terminate successfully.*

Proof. Assume $n = |N|$. The algorithm will create $|N|$ backup configurations, isolating one node in each backup configuration. In biconnected topologies this can always be done. Along with a node u, all attached links except one, say (u, v), can be isolated. By forcing node v to be the next node processed (`isolate` line 17), and the link (v, u) to be first among $A(v)$ (line 18), node v and link (v, u) will be isolated in the next configuration. This can be repeated until we have configurations so that every node and link is isolated. This holds also for the last node processed, since its last link will always lead to a node that is already isolated in another configuration. Since all links and nodes can be isolated, the algorithm will terminate successfully. □

A ring topology is a worst-case example of a topology that would need $|N|$ backup configurations to isolate all network elements. In section 5.5 we

analyze the number of backup configurations created by algorithm 5.1 for different input network topologies.

Complexity

The complexity of the proposed algorithm is determined by the loops and the complexity of the `connected` method. This method performs a procedure similar to determining whether a node is an articulation point in a graph, bound to worst case $\mathcal{O}(|N|+|A|)$. Additionally, for each node, we run through all adjacent links, whose number has an upper bound in the maximum node degree Δ. In the worst case, we must run through all n configurations to find a configuration where a node can be isolated. The worst case running time for the complete algorithm is then bound by $\mathcal{O}(n\Delta|N||A|)$.

5.4 Local Forwarding Process

Given a sufficiently high n, the algorithm presented in section 5.3 will create a complete set of valid backup configurations. Based on these, a standard shortest path algorithm is used in each configuration to calculate configuration specific forwarding tables. In this section, we describe how these forwarding tables are used to avoid a failed component.

When a packet reaches a point of failure, the node adjacent to the failure, called the *detecting node*, is responsible for finding a backup configuration where the failed component is isolated. The detecting node marks the packet as belonging to this configuration, and forwards the packet. From the packet marking, all transit routers identify the packet with the selected backup configuration, and forward it to the egress node avoiding the failed component.

Consider a situation where a packet arrives at node u, and cannot be forwarded to its normal next-hop v because of a component failure. The detecting node must find the correct backup configuration without knowing the root cause of failure, i.e., whether the next-hop node v or link (u, v) has failed, since this information is generally unavailable.

Let $C(u)$ denote the backup configuration where node u is isolated, i.e., $C(u) = C_i \Leftrightarrow u \in S_i$. Similarly, let $C(u, v)$ denote the backup configuration where the link (u, v) is isolated, i.e., $C(u, v) = C_i \Leftrightarrow w_i(u, v) = \infty$. Assuming that node d is the egress (or the destination) in the local network domain, we can distinguish between two cases. If $v \neq d$, forwarding can be done in

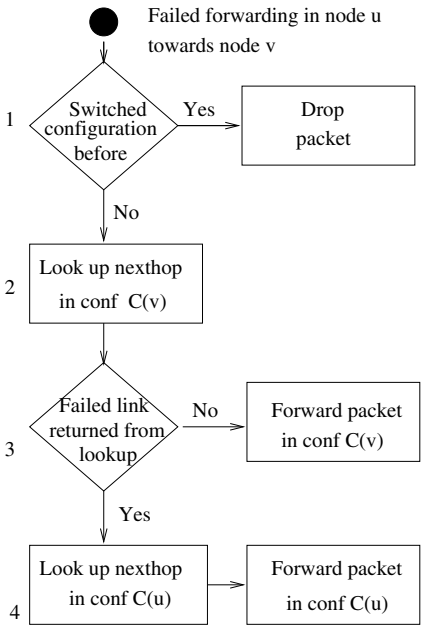


Figure 5.2: Packet forwarding state diagram.

configuration $C(v)$, where both v and (u, v) will be avoided. In the other case, $v = d$, the challenge is to provide recovery for the failure of link (u, v) when node v is operative. Our strategy in this case is to forward the packet using a path to v that does not contain (u, v). Furthermore, packets that have changed configuration before, and still meet a failed component on their forwarding path, must be discarded. This way packets loops are avoided, also in the case that node d indeed has failed. The steps that are taken in the forwarding process by the detecting node u are summarized in figure 5.2.

Assume there is only a single component failure in the network, detected by node u on path to the network-local destination d via node v.

Proposition 5.4.1. *Node u selects configuration C_i so that $v \notin \mathcal{N}(p_i(u, d))$, if $v \neq d$.*

Proof. Node u selects $C(v)$ in step 2. Node v is isolated in $C(v)$ and will not

be in the shortest path $p_i(u, d)$ according to proposition 5.3.2. □

Proposition 5.4.2. *Node u selects configuration C_i so that $(u, v) \notin \mathcal{A}(p_i(u, d))$.*

Proof. If $v \neq d$, node u selects $C(v)$ in step 2, and neither node v nor link (u, v) will be in the shortest path $p_i(u, d)$.

Assume that v is the egress node for destination d. Remember that according to (5.3), $C(u, v) = C(u) \vee C(u, v) = C(v)$. We distinguish between three possible cases, illustrated in figure 5.3.

If $C(u) = C_i$ and $C(v) = C_i$ as in figure 5.3a), then $C(u, v) = C_i$ according to the definition of an isolated node and (5.2). Forwarding step 2 will select $C(v) = C_i$ and $\mathcal{A}(p_i(u, v))$ does not contain (u, v).

If $C(u) = C_i, C(v) = C_j, i \neq j$, and $C(u, v) = C_j$ as in figure 5.3b), forwarding step 2 will select $C(v) = C_j$ and $\mathcal{A}(p_j(u, v))$ does not contain (u, v).

Finally, if $C(u) = C_i, C(v) = C_j, i \neq j$, and $C(u, v) = C_i$ as in figure 5.3c), forwarding step 2 will select $C(v) = C_j$. Link (u, v) is not isolated in C_j, and will be returned as the next hop. Step 3 will detect this, and step 4 will select $C(u) = C_i$ and $A(p_i(u, v))$ does not contain (u, v). □

5.4.1 Implementation issues

The forwarding process can be implemented in the routing equipment as presented above, requiring the detecting node u to know the backup configuration $C(v)$ for each of its neighbors. Node u would then perform at most two additional next-hop lookups in the case of a failure. However, all nodes in the network have full knowledge of the structure of all backup configurations. Hence, node u can determine in advance the correct backup configuration to use if the normal next hop for a destination d has failed. This way the forwarding decision at the point of failure can be simplified at the cost of storing the identifier of the correct backup configuration to use for each destination and failing neighbor.

For the routers to make a correct forwarding decision, each packet must carry information about which configuration it belongs to. This information can be either explicit or implicit. An explicit approach could be to use a distinct value in the DSCP field of the IP header to identify the configuration. As we will see shortly, a very limited number of backup configurations are needed to guarantee recovery from all single link or node failures, and

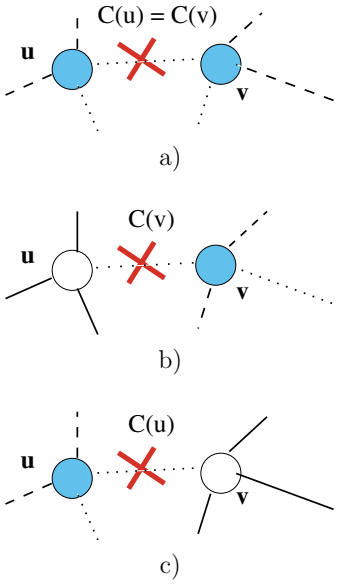


Figure 5.3: When there is an error in the last hop $u \rightarrow v$, a packet must be forwarded in the configuration where the connecting link is isolated. The figure shows isolated nodes (shaded color), restricted links (dashed), and isolated links (dotted). In cases (a) and (b), $C(u, v) = C(v)$, and the forwarding will be done in $C(v)$. In case (c), $C(u, v) \neq C(v)$, and the forwarding will be done in $C(u)$.

hence the number of needed values would be small. A more implicit approach would be to assign a distinct local IP address space for each backup configuration. Each node in the IGP cloud would get a separate address in each configuration. The detecting node could then encapsulate recovered packets and tunnel them shortest path in the selected backup configuration to the egress node. The packets would then be decapsulated at the egress and forwarded from there as normal towards the final destination. The drawback with this method is the additional processing and bandwidth resource usage associated with tunneling.

Recent IETF standardization work on Multi Topology routing mechanisms [94, 95] provides a useful framework for MRC implementation. These IETF drafts specify how routers can exchange information about the link weights used in several logical topologies, and build topology specific forwarding tables. Use of these drafts for providing proactive recovery is sketched in [97].

5.5 Performance Evaluation

MRC requires the routers to store additional routing configurations. The amount of state required in the routers is related to the number of such backup configurations. Since routing in a backup configuration is restricted, MRC will potentially give backup paths that are longer than the optimal paths. Longer backup paths will affect the total network load and also the end-to-end delay.

Full, global IGP re-convergence determines shortest paths in the network without the failed component. We use its performance as a reference point and evaluate how closely MRC can approach it. Note that MRC yields the shown performance immediately after a failure, while IP re-convergence can take seconds to complete.

5.5.1 Method

We compare the network performance when MRC is used to recover traffic to the performance in the failure-free case (denoted "IGP normal") and after a full global re-convergence ("IGP rerouting").

We have implemented the algorithm described in section 5.3.2 and created configurations for a wide range of biconnected synthetic and real topologies.

The synthetic topologies are obtained from the BRITE topology generation tool [89] using the Waxman [92] and the Generalized Linear Preference (GLP) [90] models. The number of nodes is varied between 16 and 512 to demonstrate the scalability. To explore the effect of network density, the average node degree is 4 or 6 for Waxman topologies and 3.6 for GLP topologies. For all synthetic topologies, the links are given unit weight. The real topologies are taken from the Rocketfuel topology database [88].

For each topology, we measure the minimum number of backup configurations needed by our algorithm to isolate every node and link in the network. Based on the created configurations, we measure the backup path lengths (hop count) achieved by our scheme after a node failure. For a selected class of topologies, we evaluate the backup path lengths dependence on the number of backup configurations.

The shifting of traffic from the normal path to a recovery path changes the load distribution in the network, and can in some cases lead to congestion and packet loss. We therefore test the effect our scheme has on the load distribution after a failure. To do this, we have performed simulations of the European COST239 network [98] shown in figure 5.4, designed to connect major cities across Europe. All links in the network are given an equal abstract capacity of 100. To achieve a good load distribution and minimize the chances of congestion in the failure-free case, we adopt the link weight optimization heuristic introduced in [69]. They define a piecewise linear cost function Φ that is dependent on the load $l(a)$ on each of the links a in the network. Φ is convex and resembles an exponentially growing function. They then introduce a local search heuristic that tries to minimize the value of Φ by randomly perturbing the link weights. This local search heuristic has been shown to give performance that is close to the optimal solution that can be achieved by a connection oriented technology like MPLS.

The COST239 network is selected for this evaluation because of its resilient network topology. By using this network, we avoid a situation where there exists only one possible backup path to a node. The differences with respect to link loads between different recovery strategies will only be visible when there exists more than one possible backup path. In the COST239 network each node has a node degree of at least four, providing the necessary maneuvering space.

For our load evaluations, we use a traffic matrix where the traffic between two destinations is based on the population of the countries they represent

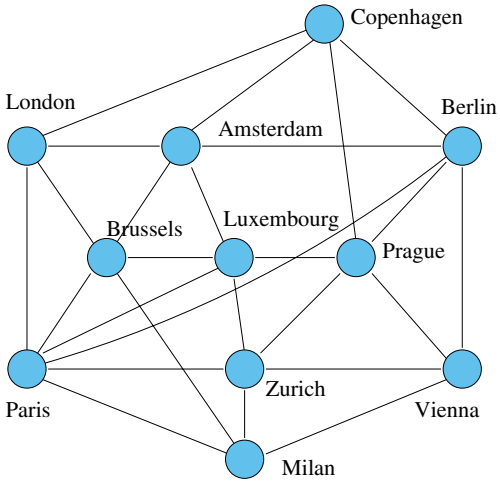


Figure 5.4: The COST239 network

[98]. For simplicity, we look at constant packet streams between each node pair. The traffic matrix has been scaled so that the load on the most utilized link in the network is about 2/3 of the capacity. We use shortest path routing with equal splitting of traffic if there exists several equal cost paths towards a destination.

5.5.2 Number of Backup Configurations

Figure 5.5 shows the minimum number of backup configurations that algorithm 5.1 could produce in a wide range of synthetic topologies. Each bar in the figure represents 100 different topologies given by the type of generation model used, the links-to-node ratio, and the number of nodes in the topology. Table 5.2 shows the minimum number of configurations algorithm 5.1 could produce for selected real world topologies.

The results show that the number of backup configurations needed is usually modest; 3 or 4 is typically enough to isolate every element in a topology. No topology required more than six configurations. In other words, algorithm 5.1 performs very well even in large topologies. The algorithm fails

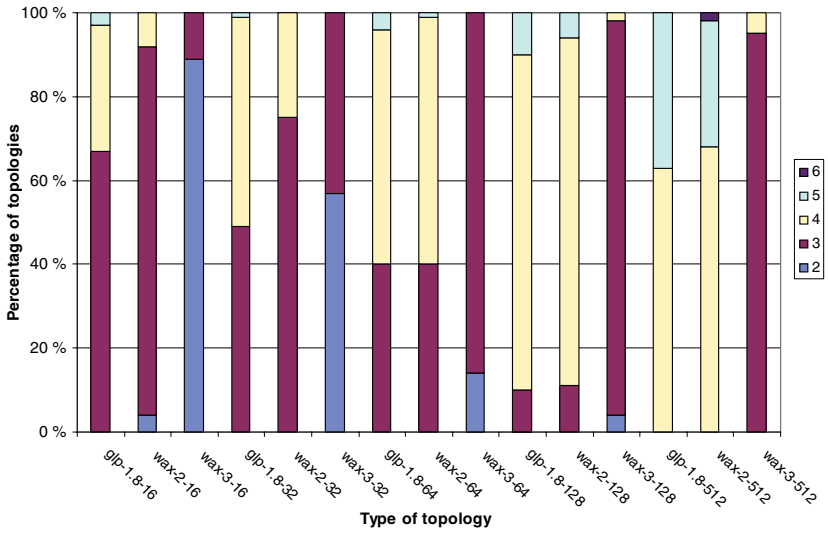


Figure 5.5: The number of backup configurations required for a wide range of BRITE generated topologies. As an example the bar name wax-2-16 denotes that the Waxman model is used with a links-to-node ratio of 2, and with 16 nodes.

Table 5.2: Number of backup configurations for selected real world networks

Network	**Nodes**	**Links**	**Configurations**
Sprint US	32	64	4
German Tel	10	17	3
DFN	13	37	2
Geant	19	30	5
COST239	11	26	3

only if it meets a node that if isolated disconnects the backbone in each of the n backup configurations. The algorithm often goes through all network nodes without meeting this situation even if n is low, and is more successful in topologies with a higher average node degree.

In section 5.3.2 we stated that the problem of finding a minimal complete set of valid configurations can be transformed to the Set Covering problem. It has long been known that heuristic algorithms can efficiently approximate an optimal solution to this problem [99], which makes the good performance of algorithm 5.1 less surprising.

It is difficult to quantify exactly the amount of extra state that must be stored in the forwarding tables of the routers in order to support MRC, because this would be highly dependent on the FIB design of the specific router. It is however clear that the state requirements will is some way be dependent on the number of backup configurations used. A modest number of backup configurations shows that our method is implementable without requiring a prohibitively high amount of state information.

5.5.3 Backup Path Lengths

Figure 5.6 shows path length distribution of the recovery paths after a node failure. The numbers are based on 100 different synthetic Waxman topologies with 32 nodes and 64 links. All the topologies have unit weight links. Results for link failures show the same tendency and are not presented.

For reference, we show the path length distribution in the failure-free case ("IGP normal"), for all paths with at least two hops. For each of these paths, we let every intermediate node fail, and measure the resulting recovery path lengths using global IGP rerouting, local rerouting based on the full topology except the failed component ("Optimal local"), as well as MRC with 5 backup configurations.

We see that MRC gives backup path lengths close to those achieved after a full IGP re-convergence. This means that the affected traffic will not suffer from unacceptably long backup paths in the period when it is forwarded according to an MRC backup configuration.

Algorithm 5.1 yields richer backup configurations as their number increases. In figure 5.7 we have plotted the average backup path lengths for the 75 of the 100 Waxman-32-64 input topologies that could be covered using 3 backup configurations. The figure shows that the average recovery path length decreases as the number of backup configurations increases.

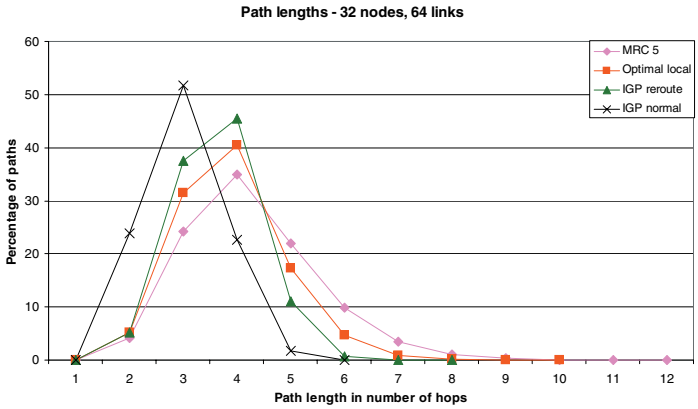


Figure 5.6: Backup path lengths in the case of a node failure.

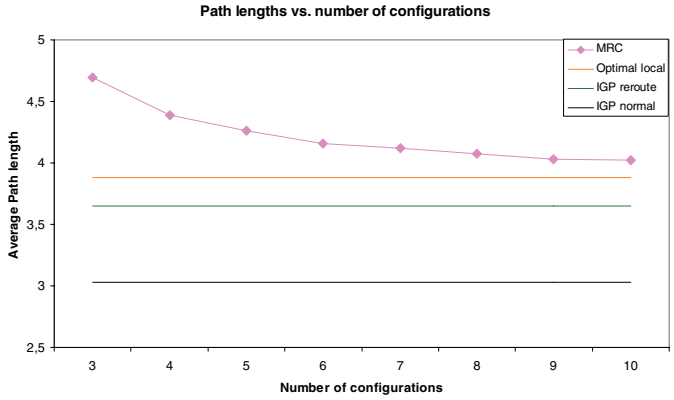


Figure 5.7: Average backup path lengths in the case of a node failure as a function of the number of backup configurations.

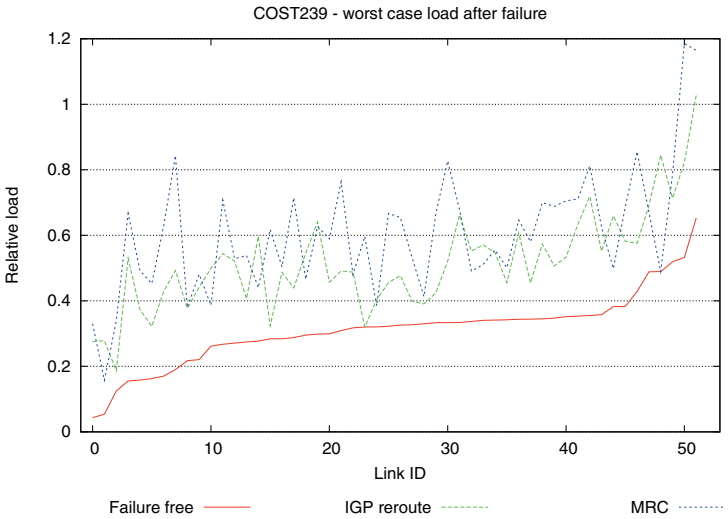


Figure 5.8: Load on all unidirectional links in the failure free case, after IGP re-convergence, and when MRC is used to recover traffic. Shows each individual links worst case scenario.

5.5.4 Load on Individual Links

In order to evaluate the routing performance while MRC is used to recover traffic, we measure the throughput on each unidirectional link for every possible link failure. We then find the maximum link utilization over all failures for each link. Five backup configurations were used.

Figure 5.8 shows the maximum load on all links, which are indexed from the least loaded to the most loaded in the failure-free case. The results indicate that the restricted routing in the backup topologies result in a worst case load distribution that is comparable to what is achieved after a complete IGP rerouting process.

However, we see that for some link failures, MRC gives a somewhat higher maximum link utilization in this network. The maximum link load after the worst case link failure is 118% with MRC, compared to 103% after a full IGP re-convergence. In the next chapter, we discuss a method for improving the

post failure load balancing with MRC.

5.6 Summary

In this chapter we have presented Multiple Routing Configurations as an approach to achieve fast recovery in IP networks. Like RRL described in chapter 4, MRC is based on providing the routers with additional routing configurations, allowing them to forward packets along routes that avoid a failed component. MRC guarantees recovery from any single node or link failure in an arbitrary biconnected network. By calculating backup configurations in advance, and operating based on locally available information only, MRC can act promptly after failure discovery.

MRC operates without knowing the root cause of failure, i.e., whether the forwarding disruption is caused by a node or link failure. Different from RRL, this is achieved by using careful link weight assignment according to the rules we have described. The link weight assignment rules also provide basis for a forwarding procedure that successfully solves the last hop problem.

The performance of the algorithm and the forwarding mechanism has been evaluated using simulations. We have shown that MRC scales well: 3 or 4 backup configurations is typically enough to isolate all links and nodes in our test topologies. MRC backup path lengths are comparable to the optimal backup path lengths—MRC backup paths are typically zero to two hops longer.

We have evaluated the effect MRC has on the load distribution in the COST239 network while traffic is routed in the backup configurations. In some cases, the restricted routing in the backup configurations gives an unwanted shift of recovered traffic to already highly loaded links. In the next chapter, we will present a method for mitigating this effect.

Chapter 6

MRC Routing Performance

In the previous chapter, we introduced MRC as a proactive recovery scheme that can route traffic around a failed element immediately after the failure is detected. In this chapter, we investigate the routing performance aspects of the MRC method, and propose a strategy for reducing the chances of congestion while traffic is routed according to the backup configurations.

Most of the contents in this chapter will also be published in [100].

6.1 Introduction

The motivation for introducing fast recovery mechanisms at the networking layer is to avoid packet loss during the IGP re-convergence phase, and to increase network stability by handling failures without triggering a global re-convergence. Existing proactive IP recovery schemes are limited to guaranteeing loop-free connectivity in the network after a failure, and do not consider the post-failure load distribution. The shifting of traffic to alternate links after a failure can lead to congestion and packet loss in parts of the network [13]. If the routing calculated by the proactive recovery scheme leads to congestion, it limits the time that the the backup paths can be used to forward traffic before the global routing protocol is informed about the failure. This reduces the chance that a transient failure can be handled without a full global routing re-convergence. Hence, it is important that the reduced packet loss is not spoiled by creating congestion in other parts of the network.

A proactive recovery scheme should not only guarantee connectivity after a failure, but also do so in a manner that does not cause an unacceptable load

distribution. This requirement has been noted as being one of the principal challenges for precalculated IP recovery schemes [52]. We believe that a well engineered distribution of recovered traffic will be crucial for the adoption of any fast IP recovery method.

Important work on traffic engineering in OSPF/IS-IS networks focus on optimizing link weights, so that traffic is well distributed across the available links. The work in this area has focused either on the failure free case [69, 67, 68], or on finding link weights that work well both in the normal case and when the routing protocol has converged after a single link failure [71, 72, 73]. A major drawback of these solutions is that they compromise performance in the failure free case in order to give reasonable performance after a failure. Also, these schemes focus on the load distribution after the convergence of the IGP routing protocol, and are not designed to work with fast IP recovery schemes. Very little work has been done on the traffic engineering properties of proactive IP recovery methods.

With MRC, the link weights are set individually in each backup configuration. This gives great flexibility with respect to how the recovered traffic is routed. The backup configuration used after a failure is selected based on the failure instance, and hence we can choose link weights in the backup configurations that are well suited for only a subset failure instances.

6.1.1 Our contributions

In this chapter, we discuss how we can achieve a good load distribution in the network immediately after a link failure, when MRC is used as a fast recovery mechanism. We present an algorithm to create the MRC backup configurations in a way that takes the traffic distribution into account. Then, we present a heuristic aimed at finding a set of link weights for each backup configuration that distributes the load well in the network after any single link failure. Our scheme is strictly proactive; no link weights need to be changed after the discovery of a failure.

With MRC, all recovered traffic is routed in the backup configurations. This allows us, unlike previous proposals, to optimize for link failures without compromising performance in the failure free case. Also, our work is the first to address the issue of load balancing after a failure in the context of a proactive IP recovery scheme.

Our solution consists of three phases; first the link weights in the normal

configuration are optimized while only taking the failure free situation into account, second we take advantage of the load distribution in the failure free case to construct the MRC backup configurations in an intelligent manner, and third we optimize the link weights in the backup configurations to get a good load distribution after any link failure.

Our method for link weight setting is based on perturbing link weights using a local search heuristic. The link weights in the backup configurations are optimized to give good performance after any link failure. However, optimizing for all possible link failures does not scale well as network size increases, because of the number of evaluations needed. To overcome this problem, we assume that only a few link failures are *critical* with respect to the load distribution after failure. A link failure is more critical if it is likely that it leads to more congestion. After identifying the most critical link failures, we use only these failures in our optimization process [72].

We have evaluated our approach using simulations on several real and synthetically generated network topologies, and we find that we achieve a load distribution while using MRC that is usually better than after a full OSPF/IS-IS re-convergence with original link weights. Our results approach those of a method aimed at a good load distribution after the routing protocol has converged on the new topology [68], with the additional benefits that our method does not compromise on the performance in the failure free case.

The rest of this chapter is structured as follows. In section 6.2, we discuss what decides the post-failure load distribution under MRC, and present our algorithm for creating the backup configurations and our link weight optimization heuristic. Then we evaluate our method in section 6.3, before we conclude and offer directions for further work in section 6.4.

6.2 Routing optimization with MRC

MRC recovers from a link or node failure in the network by redirecting the affected traffic using predefined backup configurations. For the purpose of the optimizations in this chapter, we restrict ourselves to only look at link failures. For a given traffic demand matrix, the load distribution in the network after a link failure depends on three factors:

1. The link weight assignment used in the normal configuration C_0.

2. The structure of the backup configurations, i.e. which links and nodes are isolated in each $C_i \in \{C_1, \ldots, C_n\}$.

3. The link weight assignments used in the backup configurations $C_1, \ldots, C_n$.

Given a network $G = (N, A)$ and a demand matrix D, let Φ be the cost of routing the traffic load through the network. Φ depends on how the load is distributed in the network, and the exact definition of Φ could depend on whether we want to minimize delay, avoid congestion etc. Our method is agnostic with respect to the choice of a particular function Φ, as long as it penalizes the use of heavily loaded links. The cost function we use in our evaluations is defined in section 6.3.

With the shortest path routing used in OSPF/IS-IS, the cost Φ is determined by the network graph G, the demand matrix D, and the weight assignment w used in the network. Our goal is to minimize the cost Φ in both the normal case and after any single link failure for a given G and D. Our strategy for achieving this is threefold. First, we use a heuristic to optimize the link weights in the normal configuration C_0. Second, we take advantage of the knowledge of the load distribution in the failure free case when we create the backup configurations $C_1, \ldots, C_n$. Third, we again use a heuristic to optimize the link weights used in the created backup configurations.

6.2.1 The failure free case

With MRC, all traffic is routed according to C_0 in the failure free case. When there is a failure, all recovered traffic is routed according to the appropriate backup configurations. This logical separation gives us great flexibility to distribute the recovered traffic across available links without sacrificing performance in the normal case. One of the attractive features of our solution, is that we can optimize the weights w_0 used in the normal configuration C_0 for the failure free case only, without taking the post-failure load distribution into account.

To optimize w_0, we adopt a modified version of the local search heuristic presented in [69]. We use this heuristic because it is well known and has been shown to give good performance with modest complexity, but in principle we could use any other weight search heuristic with the same objective of minimizing the cost function Φ.

The heuristic starts with a weight assignment w_0 where $w_0(a) = w_{max}/2$ for all $a \in A$, and calculates the load $l(a)$ on each link and the value of the cost function Φ resulting from w_0. Then a given number of iterations are performed. In each iteration, Φ is evaluated for a subset of the *neighborhood* of w_0. A neighbor of w_0 is a weight assignment obtained by changing the link weight of a single link. For each link in the network (one at a time), a new link weight from the range $\{1, \ldots, w_{max}\}$ is randomly picked, and Φ is evaluated after each change. The neighbor that gives the lowest value of Φ, is selected as the new w_0. To escape from local minima in the search space, the heuristic randomly changes the weight of a fraction of the links if there is no improvement after a given number of iterations. A hashing function is used to avoid looping between solutions. For a detailed explanation of the search heuristic, see [69].

6.2.2 Creating the backup configurations

The structure of the backup configurations is important for the load distribution after a failure. Traffic that is recovered in configuration C_i is forwarded only in the backbone B_i, except in the first and last hops. A configuration where many nodes and links are isolated gives a *sparse* (less connected) backbone. Such a configuration gives few options with regards to where recovered traffic should be routed. Conversely, a backup configuration with a *rich* backbone leaves more choices with respect to routing, and increases the possibilities to get a good distribution of load after a failure.

With MRC, the distribution of recovered traffic depends on the interaction between the structure of the backup configurations, and the weight assignments $w_1, \ldots, w_n$. Ideally, we would like to create the backup configurations and decide $w_1, \ldots, w_n$ at the same time in such a way that the cost Φ is minimized. However, such a solution would probably have to involve heavy computations, and in this work we instead settle for a solution where we first create the backup configurations, and then decide the link weight assignments. Joint optimization of the backup configuration structure and the link weight assignments $w_1, \ldots, w_n$ is left for future study.

The intuition behind our algorithm for creating backup configurations, is that we want the amount of traffic that is potentially recovered in each backup configuration to be approximately equal. We want to avoid that the failure of heavily loaded links results in large amounts of traffic being recovered in backup configurations with a sparse backbone. Instead, this

traffic should be routed in a rich backbone, where we have a better chance of distributing it over less loaded links by setting appropriate link weights. The algorithm described here resembles Alg. 5.1, with the major difference that while Alg. 5.1 tries to balance the number of isolated elements in each backup configuration, we here try to balance the amount of recovered traffic.

When we have decided the weight assignment w_0, the load on each link in the failure free case is given. We use this information to decide the *potential* of each node in the network and the potential of each backup configuration.

Definition. The potential $\gamma(u)$ of a node u is the sum of the load on all its incoming and outgoing links:

$$\gamma(u) = \sum_{v \in N} (l(u,v) + l(v,u)) \tag{6.1}$$

Definition. The potential γ_i of a backup configuration C_i is the sum of the potential of all nodes that are isolated in C_i:

$$\gamma_i = \sum_{u \in S_i} \gamma(u) \tag{6.2}$$

The input to our algorithm for generating backup configurations is the normal configuration C_0, and the number n of backup configurations we want to create. As shown in section 5, n can be set surprisingly low; 3 or 4 backup configurations is usually sufficient to isolate all elements in a network. In section 6.3, we evaluate the effect the choice of n has on the post failure load distribution.

Our modified backup configuration construction method is defined in Alg. 6.1. We start by ordering all nodes with respect to their potential (line 6 in Alg. 6.1). Then each node u is assigned to a tentative backup configuration $C_{\mathrm{T}}(u)$ in line 7, so that the potential γ_i of each backup configuration is approximately equal:

$$\gamma_i \approx \gamma_j, i, j \in \{1, \ldots, n\} \tag{6.3}$$

The nodes with the smallest potential are assigned to C_1, those with somewhat higher potential to C_2, and so on with the nodes with the highest potential in C_n.

We then go through all nodes in the network, and attempt to isolate each node u in its tentative backup configuration (line 11). The function

`isolate` is defined in section 5.3.2. For some nodes, this might not be possible without breaking the definition of a valid configuration given in (5.4). This node is then attempted isolated in backup configuration C_{i+1}, C_{i+2} and so on (line 19), until all backup configurations are tried. If a node can not be isolated in any of the backup configurations, we give up and abort. Note that when nodes can not be isolated in the backup configuration it was assigned to, this will disturb the desired property of equalizing γ_i among the backup configurations. However, in our experience this typically only happens for a very limited number of nodes, and the consequences are not severe.

Algorithm 6.1: Load-aware backup configurations.

```
1  for i ∈ {1...n} do
2      C_i ← (G, w_0)
3      S_i ← ∅
4  end
5  Q_n ← N
6  sort(Q_n, γ, ascending)
7  assign_C_T(Q_n)
8  Q_a ← ∅
9  while Q_n ≠ ∅ do
10     u ← first (Q_n)
11     i = C_T(u)
12     while u ∉ S_i and not all configurations tried do
13         if connected(B_i \ {u}) then
14             C_tmp ← isolate(C_i, u)
15             if C_tmp ≠ null then
16                 C_i ← C_tmp
17                 S_i ← S_i ∪ {u}
18             else
19                 i ← (i mod n) + 1
20     if u ∉ S_i then
21         Give up and abort
22 end
```

The outcome of this algorithm is dependent on the network topology and the traffic demand matrix D. If the load is close to equally distributed on the links before a failure, we end up with approximately the same number

of nodes isolated in each backup configuration. If the traffic distribution is more skewed (as is the case with the traffic model used in our evaluations), the algorithm typically ends up with isolating many nodes with a small potential in C_1, while only very few nodes, with a high potential, are isolated in backup configuration C_n. This is in accordance with the goal of having a rich backbone in which to reroute traffic after the failure of heavily loaded links.

6.2.3 Optimizing link weights in the backup configurations

When we have created the backup configurations $C_1, \ldots, C_n$, the next challenge is to decide the weight assignments $w_1, \ldots, w_n$. We use a similar search heuristic as in the failure free case. The straightforward way of doing this would be to evaluate the cost of the network for all possible link failures and for each candidate set of weight assignments. However, evaluating a candidate weight assignment is a rather expensive operation in terms of computing resources. The large number of evaluations needed to cover all failure instances makes this unfeasible for large networks. We therefore apply a strategy where we assume that a limited number of link failures are the most critical with respect to the load distribution, as introduced in [72]. Our method for deciding the weight assignments $w_1, \ldots, w_n$ in the backup configurations then consists of two subproblems. First we need to find the critical links, i.e the subset of links whose failure has the most grave impact on the load distribution in the network. Then we evaluate each candidate set of weight settings against the failure of the small set of critical links only. This gives a significant reduction in the number of cost evaluations needed, and makes our method feasible also for larger networks.

Identifying critical links

Let Φ^a denote the cost of routing the demands through the network when link a has failed. We define the critical link set L_C as the k links that give the highest value of Φ^a upon failure, i.e. L_C is the set of links with cardinality k so that $\forall a \in L_C, b \notin L_C : \Phi^a \geq \Phi^b$. Note that the initial calculation of L_C is performed after we have optimized w_0, but before we have optimized $w_1, \ldots, w_n$.

There are two potential dangers with this choice of critical links. First, there might be links whose failure will give a high cost under *any* weight assignment, e.g. if there is only one possible backup path. Trying to optimize for the failure of such links is obviously futile. Second, the impact of a link failure on the network cost is a function of the current set of weight assignments. A failure that has little impact with one weight assignment, might have a grave impact with another weight assignment. We might thus end up with a situation where the failures that are in fact most damaging for the routing performance with the final weight assignment, are not included in the critical link set.

These considerations have led the authors of [73] to propose a strategy that incorporates both the failure instance *and* the routing when selecting the critical links. They observe that assigning a high weight to a link vaguely resembles the failure of this link. They then exploit the high number of weight assignments evaluated while optimizing for the failure free case, by gathering statistical information about the cost of the network when a link has "failed" in this way. However, this method does not work well with MRC. Since recovered traffic is routed according to backup configurations with completely independent weight assignments, setting a high weight on a link in C_0 does not give a good indication of what will happen if that link fails.

However, the independent routing of recovered traffic in the backup configurations greatly reduces the second point of criticism against our method for selecting critical links stated above. We only manipulate the weight assignments $w_1, \ldots, w_n$ used in the backup configurations in the second phase of our heuristic, and never change w_0. Hence, it is only the *recovered* traffic that is affected by the different weight settings evaluated. This makes L_C less dependent on the current weight assignments. To compensate for the dependency that still exists, we recalculate L_C a few times during our search. While the first objection against our measure of criticality still holds (some failures give high cost independent of weight assignment), we will see that our selection of L_C gives good performance.

The MRC forwarding strategy explained in section 5.4 gives a fixed dependency between a particular link failure and the two backup configurations used to route the recovered traffic. Hence, the cost of the network after the failure of a link in L_C is influenced only by the weight assignments w_i and w_j used in these two configurations, and not by the assignments used in the

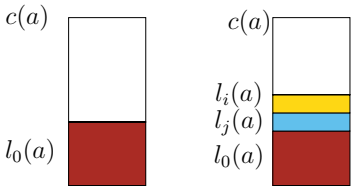


Figure 6.1: Traffic on link a before and after a failure.

other backup configurations. For each backup configuration C_i, we define $L_i \subseteq L_C$ as the set of critical links whose failure results in recovered traffic being routed according to C_i:

Definition. The set of critical links L_i of a configuration C_i is

$$L_i = \{a \in L_C | a \notin B_i\} \tag{6.4}$$

Local search heuristic

When we have defined the critical links of each backup configuration, we perform a local search to optimize the weight assignments $w_1, \ldots, w_n$. Note first that according to the MRC forwarding strategy, traffic is diverted to two different backup configurations after a failure, depending on the destination. After a failure, we will in general have traffic in two backup configurations i and j (in addition to the normal configuration). Letting $l_i(a)$ denote the load on link a that is routed according to configuration C_i, we have that $l(a) = l_0(a) + l_i(a) + l_j(a)$, as illustrated in figure 6.1.

The traffic distribution after a failure is thus dependent on the weight assignment in more than one backup configuration. Because of this, we can not optimize the weight assignments one at a time. Instead, we use an algorithm that tries to optimize all weight assignments $w_1, \ldots, w_n$ at the same time.

Like in the optimization of w_0 described above, we start with weight assignments where $w_i(a) = w_{max}/2, a \in A_i, p \in P$. We then perform a given number of iterations, evaluating the cost function Φ over the critical link failures with different weight assignments. In our search heuristic, the aim

is to minimize the sum Ψ of the cost of the network after the failure of each link in L_C:

$$\Psi = \sum_{a \in L_C} \Phi^a \tag{6.5}$$

In each iteration step, we perform the following operations:

1. First we select the next backup configuration C_i in a round robin fashion.

2. For each link a in the backbone B_i of this configuration (one link at a time), we choose a random link weight $w_i(a)$ from the interval $[1, \ldots, w_{max}]$. This corresponds to evaluating $1/w_{max}$ of the neighborhood of w_i.

3. We evaluate Ψ for each of these candidate weight assignments.

Note that for the failure of the links that are not included in L_i for the current configuration, the evaluation performed in the third step will always yield the same Φ, irrespective of w_i. Hence, these values can be reused for all candidate weight assignments. We only have to recompute the cost of the network for the failures of the links in L_i. This significantly reduces the number of evaluations we have to perform in our heuristic.

If we do not see an improvement of Ψ after a given number of consecutive iterations, we jump to another area of the search space by randomly changing the link weight of a fraction of the links in the network.

Complexity

Optimizing n different weight assignments for a multitude of potential link failures is a complex task. An important goal in our approach has therefore been to create a heuristic that scales to networks of hundreds of nodes. This is achieved through the use of the critical link set L_C, and the further division of this into a set of critical links L_i for each backup configuration.

We can get an idea of the complexity of our heuristic by counting the number of evaluations of the network cost Φ^a we need to perform, compared to the methods in [72, 73]. These methods try to optimize a single link weight assignment only, and use the same strategy of only evaluating the most critical link failures. In each iteration, they need to calculate the value of Φ^a $|L_C|$

times for each candidate weight assignment. With our heuristic, we only need to evaluate Φ^a $|L_i|$ times for each candidate weight assignment. The number of links in L_i is dependent on the size of L_C and the number of backup configurations used to protect the network. The failure of a link (u, v) results in recovered traffic being diverted to one or two backup configurations, depending on whether u and v are isolated in the same configuration. The failure of a link can thus give traffic in at most 2 out of n backup configurations. If we assume that the number of isolated nodes are not very different between the configurations, we have that, on average, $|L_i|$ is roughly $|L_C| \cdot \frac{2}{n}$.

In each iteration, we only alter the link weights of links A_i in the backbone B_i of the current configuration. The weights of the isolated and restricted links that are not included in B_i are decided by MRC, and can not be changed. Obviously, the number of links $|A_i|$ in each backbone B_i is less than $|A|$.

To sum up our discussion so far; if we use i iterations with the methods described in [72, 73], evaluating the network cost Φ^a $|A|$ times with $|L_C|$ different link failures in each iteration, we will perform a total of $i \cdot |A| \cdot |L_C|$ evaluations of Φ^a. With our method, if we perform i iterations for *each* backup configuration, we end up with a total of

$$n \cdot i \cdot \overline{|A_i|} \cdot \overline{|L_i|} < 2 \cdot i \cdot |A| \cdot |L_C| \tag{6.6}$$

evaluations of Φ^a. This means that even if we let the number of backup configurations grow, we never need more than twice the number of evaluations needed by [72] and [73].

Evaluating Φ^a involves calculating a shortest path tree for each destination in the network. This can be done in a more efficient way by relying on incremental calculations [101] when evaluating Φ^a for different failures. Evaluating Φ^a is somewhat more expensive when using MRC, since we need to calculate shortest paths in one or two backup configurations in addition to the normal configuration. On the other hand, since we optimize for a smaller number of failures in each backup configuration, we have found that we can decrease the number of iterations used per configuration, and still achieve good results. All in all, our experience is that the running time of our heuristic is comparable to that of [73].

6.3 Performance evaluation

We have evaluated our approach using simulations for a range of real and synthetically generated network topologies. We use the network cost and the maximum link load after failure as performance metrics.

6.3.1 Method

Topologies and traffic

We have tested our mechanism on topologies from four existing or planned real-world network topologies from the Rocketfuel [88] database: Sprint US (PoP level, 32 nodes, 64 links), COST239 (11 nodes, 26 links), Geant (19 nodes, 30 links) and German Telecom (10 nodes, 17 links). We have also performed tests on synthetically generated topologies. We generated topologies of four different classes - 32 nodes and 64 links, 32 nodes and 96 links, and 128 nodes and 256 links. The synthetic topologies were generated using the Waxman topology model [92]. For all the topologies, both real and synthetic, all links have an equal abstract link capacity of 1 in our tests.

To evaluate the link load changes after the failure, it is necessary to know the traffic demands between all network origins and destinations. Even for real networks, this data is generally unavailable, due to its confidentiality and difficulties in collecting it. We chose to synthesize the origin-destination (OD) flow data by drawing flow values from a probability distribution, and matching the values with the OD pairs using the heuristic described in [102]. In short, we sorted the OD pairs according to their node degree and the likelihood of one of them being used as the backup node in the case of a single link failure. Then, we matched the sorted OD pair list with the sorted list of flow intensities generated using the gravity model, which is suited for this purpose [103].

Once the OD matrix is generated, it needs to be scaled to the link capacities so that it can provide a meaningful evaluation of the effect of link failures on the flows. It has proven hard to find a general parameter setting that achieves this for all networks. We chose to tune the load so that the maximum link load after the worst case failure is about 100%. In most cases, this corresponds to a maximum link load in the failure-free case of approximately 2/3 of the link capacity.

Routing and cost function

We used shortest path routing in all calculations. When multiple equal cost paths toward a destination were available, the load was split equally among them.

To evaluate a given weight assignment, we must define the cost Φ of routing a given traffic demand through the network. In this work we choose to adopt the commonly used cost function introduced in [69]. Using this cost function, each link a is given a cost ϕ_a dependent on its load $l(a)$ and its capacity $c(a)$. The total network cost $\Phi = \sum_{a \in A} \phi_a$ is then the sum of the cost of each link. The cost $\phi_a(l(a))$ of a link is defined as the continuous function with $\phi_a(0) = 0$ and derivative:

$$\phi'_a(x) = \begin{cases} 1 & for \quad 0 \leq x/c(a) < 1/3, \\ 3 & for \quad 1/3 \leq x/c(a) < 2/3, \\ 10 & for \quad 2/3 \leq x/c(a) < 9/10, \\ 70 & for \quad 9/10 \leq x/c(a) < 1, \\ 500 & for \quad 1 \leq x/c(a) < 11/10, \\ 5000 & for \quad 11/10 \leq x/c(a) < \infty \end{cases} \tag{6.7}$$

The cost function $\phi_a(l(a))$ is illustrated in figure 6.2. It is defined so that it is cheap to send traffic over lightly-loaded links, while adding traffic to a link a that is already overloaded gives a very high value of ϕ_a.

Evaluation setup

In our experiments, we optimized C_0 for the failure free case using the heuristic described in section 6.2.1 with 1000 iterations. We jumped to another area of the search space by randomly perturbing weights if we saw 200 iterations without an improvement. When optimizing the link weights in the backup configurations $C_1, \ldots, C_n$, we used as little as 20 iterations per backup configuration, and did a random perturbation after 10 non-improving iterations. We used a critical link set size $|L_C| = 20$.

As an evaluation benchmark in our experiments with the GEANT network, we compare our method to an unrealistic full rerouting approach where link weights are optimized to fit the new topology after each specific link failure. This optimization is done in the same way as the optimization of the failure free C_0. Performing this operation for every link failure takes much computing resources, and is only feasible in our experiments for small

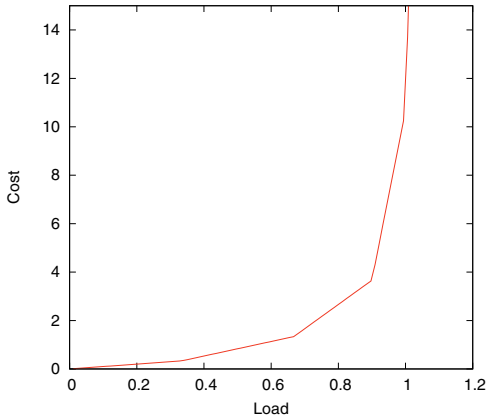


Figure 6.2: Link cost $\phi(l(a))$ as a function of $l(a)$ for link capacity $c(a) = 1$.

networks. To test the performance of our weight setting heuristic, we also compare to an idealized MRC approach where link weights in the backup configurations are optimized to fit a single link failure only. We use the same heuristic as before, but in each iteration we evaluate Φ^a for a single link only, instead of taking all critical links into account.

In our evaluation of real and synthetic networks shown in table 6.1, we show the performance of MRC using 5 and 10 backup configurations. We compare this to the results given by a complete OSPF/IS-IS re-convergence on the normal configuration. Also, in lack of other proactive recovery mechanisms that try to optimize the routing after a failure, we compare MRC performance against the method for robust routing described in [73]. This method constructs a single set of link weights that performs well in both the failure free case and with a single link failure. It is not designed to work with any fast reroute mechanism, and the load distribution is hence only achieved after a full shortest path re-convergence on the new topology. A drawback with this method is that its performance can not be optimized for failure free operation only. In our experiments, parameters are set so that we allow a cost increase of up to 20% in the failure free case with this method.

We use the cost Φ and the load on the most loaded link in the network

as our evaluation parameters. To be able to compare networks of different size, we normalize Φ with the cost of routing the demand through the same network with unlimited link capacities, i.e. a network where $\phi_a = l(a)/c(a)$ according to (6.7).

6.3.2 Results and discussion

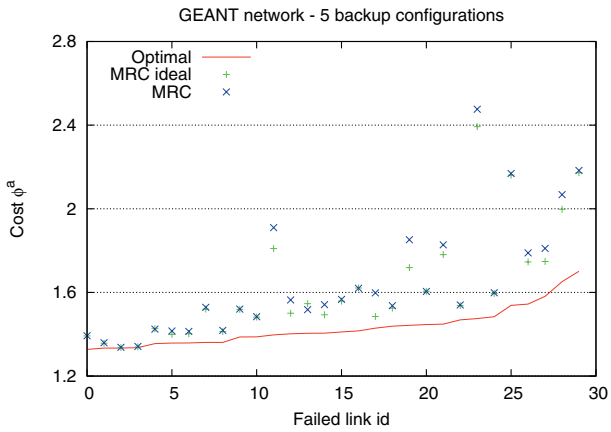
Cost and link loads in a single network

Figure 6.3a shows the network cost Φ^a after the failure of each link in the GEANT network topology. The cost is shown for the unrealistic optimal shortest path rerouting, idealized MRC, and our MRC approach. The link failures are sorted on the x-axis after increasing cost in the optimal case. The traffic demand is scaled so that the cost Φ is 1.33 in the failure free case, giving a maximum link load of 0.67. Figure 6.3b) shows the maximum link load in the network after the same link failures.

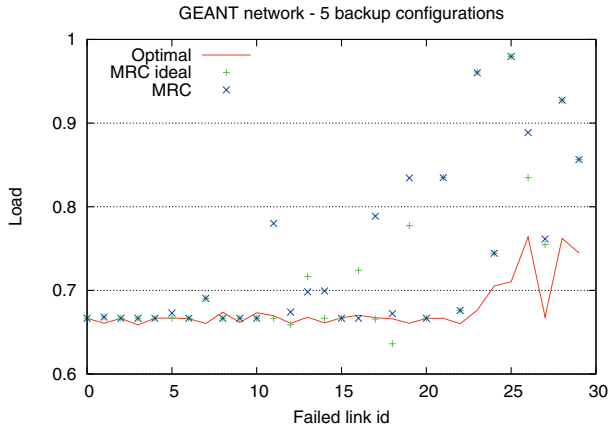
The graphs show that for most failures, MRC performance is close to that of the unrealistic optimal rerouting. For a few link failures, our MRC approach diverts more from the optimal. In these cases, the MRC backup configurations are constructed so that recovered traffic is routed over links that are already somewhat loaded. We see that when this happens, the performance of our heuristic is close to that of the idealized MRC. This indicates that if we want to further improve the performance of MRC, we could expect the best results by improving the backup configuration construction algorithm, instead of creating a better weight search heuristic. Note that MRC sometimes gives a lower maximum link load than the optimal shortest path rerouting. This happens when MRC is forced to create longer recovery paths (giving a higher Φ) due to the restrictions in the backup configurations, but this happens to avoid the most heavily loaded link that would otherwise be used.

Varying the number of backup configurations

Figure 6.4a) shows the network cost Φ^a after the worst case link failure for a synthetically generated network with 32 nodes and 64 links, using a varying number of backup configurations. Since our weight setting search contains an element of randomness, we sometimes experience cost values that deviate significantly from what is expected. To mitigate this effect, the values shown



a)



b)

Figure 6.3: Cost Φ^a and maximum link load in the network after each link failure.

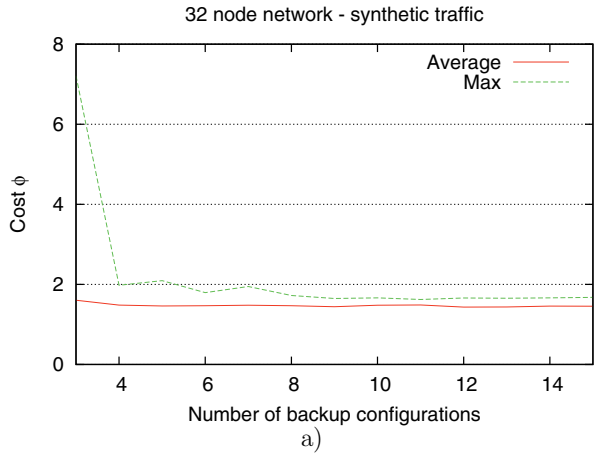
are the median value obtained by running our algorithm 20 times with a different seed.

As expected, we see that the cost is highest when the minimum number of backup configurations (3 for this network) is used. The load balancing improves when we increase the number of backup configurations used. Since each node in the network is isolated in exactly one backup configuration, increasing the number of backup configurations gives richer backbones to route the recovered traffic in. We see that increasing the number of configurations used beyond 8 gives a very limited effect for this network. We have observed similar trends for other networks. This indicates that it is possible to achieve a good load balancing using a modest number of backup configurations. As seen in figure 6.4b), the maximum link load after the worst case failure shows more variation than the maximum Φ^a. This is a result of the piecewise linear nature of the cost function (6.7), which does not prefer two links with load 0.95 to one link with load 0.90 and one with load 1.00.

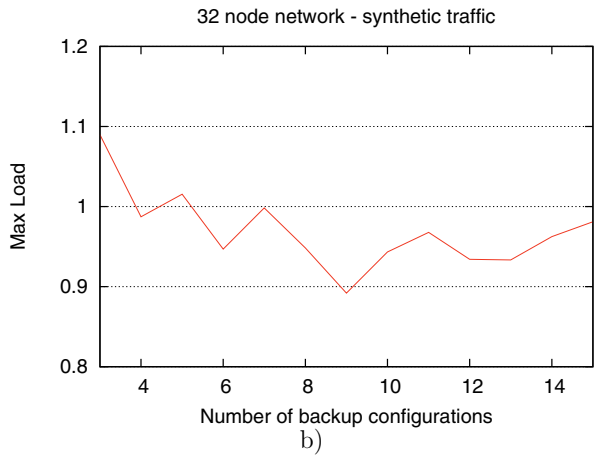
Comparing to IGP rerouting and standard MRC

In figure 6.5a), we show the worst case link loads for the load aware MRC, and after a full IGP re-convergence on the new topology. The links are sorted by the load in the failure-free case. Figure 6.5a) is directly comparable to figure 5.8. We see that the worst case load peaks for the optimized MRC are somewhat reduced compared to the standard MRC. The maximum link load after the worst case link failure has been reduced from 118% to 91%, which is better than what is achieved after a full IGP re-convergence. This is possible since the re-converged network will choose the shortest available path, while MRC in this case manages to route the recovered traffic over less utilized links.

The effect of the recovery load balancing introduced in this chapter is highlighted in figure 6.5b), where the optimized and standard MRC from chapter 5 are directly compared. Here, the links are sorted by their load after a worst case failure using standard MRC. We see how the optimized MRC often manages to route traffic over less utilized links after the failure of a heavily loaded link.



a)



b)

Figure 6.4: Cost Φ^a and maximum link load in the network after the worst case link failure. Φ^a is median over 20 runs, maximum load is mean over 20 runs.

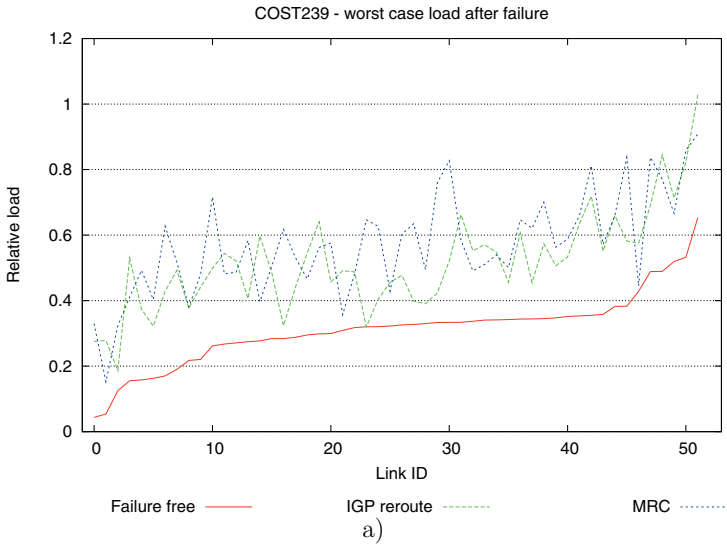


Figure 6.5: Load on all unidirectional links in the COST239 network after the worst case link failure. a)Optimized MRC vs complete IGP rerouting. b)Standard vs optimized MRC.

Table 6.1: Cost and maximum link load for selected real and synthetic network topologies

	Failure free		Proactive recovery MRC n=5			Proactive recovery MRC n=10			Reactive recovery S/G			Reactive recovery Normal SPF		
Network	Φ	l_{max}	Φ^a_{avg}	Φ^a_{max}	l_{max}	Φ^a_{avg}	Φ^a_{max}	l_{max}	Φ^a_{avg}	Φ^a_{max}	l_{max}	Φ^a_{avg}	Φ^a_{max}	l_{max}
German Tel	1.40	66%	1.91	4.85	102%	1.95	5.00	102%	1.63	2.05	81%	14.60	86.53	117%
Geant	1.36	68%	1.65	2.39	101%	1.69	4.94	108%	1.58	1.90	90%	2.54	31.91	120%
Sprint US	1.18	64%	1.40	6.05	110%	1.39	6.00	110%	1.40	5.58	110%	1.35	5.53	110%
Cost239	1.39	66%	1.57	2.62	99%	1.56	2.62	99%	1.51	1.94	79%	1.55	2.61	99%
T32-64-0	1.33	66%	1.48	2.20	103%	1.45	1.59	82%	1.42	1.60	87%	1.41	1.63	98%
T32-64-1	1.26	59%	1.39	1.73	95%	1.38	1.54	75%	1.36	1.54	94%	1.34	1.91	102%
T32-64-2	1.33	67%	1.48	2.21	100%	1.48	2.21	100%	1.42	1.52	89%	1.42	2.15	104%
T32-64-3	1.30	67%	1.46	2.65	105%	1.46	2.65	105%	1.61	3.04	109%	1.47	5.17	111%
T32-64-4	1.29	66%	1.42	1.91	96%	1.41	1.79	90%	1.35	2.04	102%	1.36	2.28	103%
T32-96-0	1.35	67%	1.43	1.99	104%	1.42	1.62	99%	1.39	1.47	92%	1.41	2.23	109%
T32-96-1	1.34	78%	1.46	3.60	110%	1.45	3.22	111%	1.39	1.85	101%	1.50	10.86	114%
T32-96-2	1.36	72%	1.59	7.60	117%	1.46	1.85	103%	1.43	3.05	111%	1.56	6.88	114%
T32-96-3	1.35	65%	1.44	2.27	108%	1.42	1.63	100%	1.41	1.57	98%	1.39	1.69	101%
T32-96-4	1.36	76%	1.48	5.05	113%	1.48	4.02	111%	1.40	1.53	97%	1.46	4.89	112%
T128-256-0	1.23	67%	1.27	1.34	91%	1.26	1.32	95%	1.25	1.28	86%	1.25	1.28	90%
T128-256-1	1.21	66%	1.24	1.30	83%	1.24	1.30	85%	1.23	1.25	73%	1.22	1.24	70%
T128-256-2	1.18	67%	1.21	1.31	92%	1.21	1.33	93%	1.19	1.22	72%	1.19	1.22	72%
T128-256-3	1.20	66%	1.23	1.31	84%	1.23	1.31	90%	1.22	1.23	82%	1.21	1.24	82%
T128-256-4	1.20	66%	1.23	1.31	84%	1.23	1.31	90%	1.21	1.24	76%	1.21	1.23	76%

Evaluation over different networks

We have evaluated the network cost and the maximum link load after the worst case link failure for a range of real-world and synthetically generated network topologies, as shown in table 6.1. Results are shown for MRC using 5 and 10 backup configurations, a normal full SPF re-convergence, and the method described in [73], denoted S/G. We have run experiments for 5 different topologies of each type of synthetic topologies. For each recovery method we show the average cost Φ^a_{avg} after each link failure, the cost Φ^a_{max} after the worst case link failure, and the load l_{max} of the most heavily loaded link after the worst case link failure. We also show the cost Φ and the maximum link load in the failure free case. The values shown in the table are median values over 3 runs with different seed.

The general trend is that MRC performs better than the normal shortest path rerouting after the worst case link failure, with respect to both cost and maximum link load. MRC performance is improved if we increase the number of backup configurations used. Using 10 backup configurations, MRC performance gets close to that of the S/G method, and for networks of moderate size and connectivity (T32-64), MRC performance is as good as that of S/G. The cost Φ in the failure free case is up to 20% higher with the S/G method than with MRC - in our experiments we typically saw values that were 3-15% higher.

Normal SPF re-convergence performs better for larger networks (T128-256) than for small networks. We believe this is partly a result of the traffic model used. With larger networks, the chance that a heavily loaded link is selected in a backup path decreases, and a normal shortest path re-convergence is closer to the optimal solution.

6.4 Summary

In this chapter, we have argued that the post-failure load distribution should be taken into account when designing a proactive recovery scheme for IP networks. We think this is imperative for the adoption of any such scheme. We presented an algorithm for creating backup configurations and a link weight assignment heuristic that reduces the chance of congestion after a link failure when MRC is used for recovery. The heuristic we propose is inspired by existing link weight heuristics used in single-topology scenarios,

but is adapted to that it works with our Multiple Routing Configurations. Our method does not compromise performance in the failure-free case, and it is strictly pre-configured; no calculations are necessary after the failure.

We have evaluated our method using both real and synthetic network topologies. Our results show that by using our scheme, MRC offers better post-failure load distribution in the network than what is achieved by a full global rerouting using the original link weights. In particular, our heuristic reduces the load on the most loaded links in the network after a worst-case link failure compared to a normal shortest path rerouting. The performance of our method is almost the same as that of the method described in [73], which is not designed to be used with a proactive IP recovery scheme and that reduces performance in the failure free case.

In the final stages of the work in this chapter, we discovered the technical report [104]. This report describes a technique inspired by the method described in chapter 5 for creating multiple topologies to achieve fast rerouting in IP networks, and a heuristic to set link weights in the topologies. They compare their post-failure load distribution to that achieved by using the not-via approach [61], and find that their multi-topology strategy performs better for the tested networks according to their metrics. Both the method for creating backup topologies and for setting link weights is substantially different from ours. As future work, we plan to compare the performance of this method to that of our own.

Chapter 7

Multi-Topology Load Balancing

So far, we have looked at mechanisms that give fast recovery from component failures in a network domain. In this chapter, we will utilize some of the concepts that we have discussed earlier to solve another kind of challenge in a network: how to maintain a good routing when the traffic demands are constantly changing.

7.1 Introduction

In connectionless intradomain routing protocols like OSPF or IS-IS, traffic engineering is done by carefully tuning the link weights that decide the shortest paths from each ingress to each egress node. Based on the network topology and the projected traffic demands, the link weights are set so as to minimize the cost of routing the demands through the network. The performance of such methods have been shown to be close to what can be achieved using connection oriented protocols like MPLS [69, 68]. The main problem with traffic engineering approaches based on optimizing link weights is that they rely heavily on the available estimate of the traffic demands. These estimates can be based on traffic measurements and projections of customers needs. However, the demands vary significantly over time, and it is difficult to get an accurate network-wide view of the situation [14].

With a changed traffic matrix, we would like to run the optimization heuristic again, and install the new optimized link weights in the network to maintain the desired load balancing properties. However, changing link weights in an operational network is a bad thing. Not only does it lead to a

period of routing instabilities as the routing protocol converges on the new topology [7], but it may also change the egress routers that are chosen in the BGP route-selection process, causing additional unwanted traffic shifts [105].

Several proposals have been made to mitigate the effects of traffic demand changes. In [15], a method is described that adapts the routing to the new traffic demands with as few weight changes as possible. This reduces the consequences, but it does not remove the problem. Other schemes try to find a link weight setting that performs well also in the presence of a link failure [71, 72, 73]. These proposals prepare for changes to the routing caused by failures, but do not handle natural changes in the traffic matrix caused by shift in user demands.

In this chapter, we propose a new method for IGP traffic engineering that avoids the problems associated with link weight changes. Our method is based on Multi-Topology (MT) routing, which is currently being defined by the IETF [94, 95]. MT routing allows the routers to maintain several independent logical topologies, with independent link weights, and hence independent routing, in each topology.

The main idea in our contribution is to construct the set of logical topologies in such a way that any congested link can be avoided in at least one topology. Traffic is then spread among the topologies in a way that gives good load balancing. We explore two different ways of utilizing this; one global method where ingress-egress flows are mapped to a topology at the ingress node, and one local method where traffic is dynamically moved to an alternate topology by the node experiencing congestion.

By looking at authentic traffic demands from the pan-European GEANT network, we show that the day to day variations of ingress-egress flows are significant. We then evaluate our two methods using simulations of this network, and compare the results to a well known method for traffic engineering based on IGP weight tuning [69]. We find that our local method significantly reduces the chances of packet loss in our simulated scenarios, while the global method performs as good as the weight tuning heuristic with a much simpler and more dynamic algorithm.

The rest of this chapter is organized as follows. In section 7.2 we illustrate that the temporal variations in traffic demands are indeed large. We then introduce MT routing and describe our algorithm for calculating logical topologies in section 7.3. In section 7.4 we explain the details of our global and local methods, before we evaluate our methods and compare them to an existing traffic engineering method in section 7.5. Finally in section 7.6, we

summarize this chapter.

7.2 Temporal variations in backbone traffic

Traffic demands vary in both daily and weekly patterns, but they also show significant stochastic day-to-day variations. In this work, we look at real intradomain traffic matrixes from the GEANT network [106]. GEANT is a high capacity network connecting the national research networks in most European countries[1]. The network consists of 23 nodes connected by 37 bidirectional links. Most of the links have capacities ranging from 2.4 Gbps to 10 Gbps, with a few links with lower capacities from 155 Mbps to 1.5 Gbps. The traffic matrixes in our dataset tell us how much data was sent from each ingress node to each egress node in every 15 minutes interval over a four month period.

The demands in the GEANT traffic trace typically have their daily peak around lunch hours. To illustrate the stochastic variations in the demands, we have computed a traffic matrix that consists of the average peak hour demands in the GEANT network over 7 consecutive days. We then look at the peak hour demands on the 8th day (a Friday), and measure how much each ingress-egress flow deviates from the previous week average.

The results are shown in figure 7.1. We see that the variations are significant. For this particular day, about 7% of the flows were reduced by more than 90% compared to the previous week average. Over 13% of the demand flows were more than doubled, and almost 5% more than 4 times as large. This illustrates how difficult it is to find a single demand matrix that can be used as input to a load balancing heuristic based on link weight manipulation.

7.3 Multi-Topology routing

Multi-Topology routing allows the routers in an AS to maintain several logical views of the network topology. The routers exchange topology-specific link state advertisements describing the properties of each link. Conceptually, the routers build a separate routing table for each topology, and create a separate forwarding entry for each destination in each topology. Data traffic

[1] A map of the GEANT topology is publicly available at http://www.geant.net

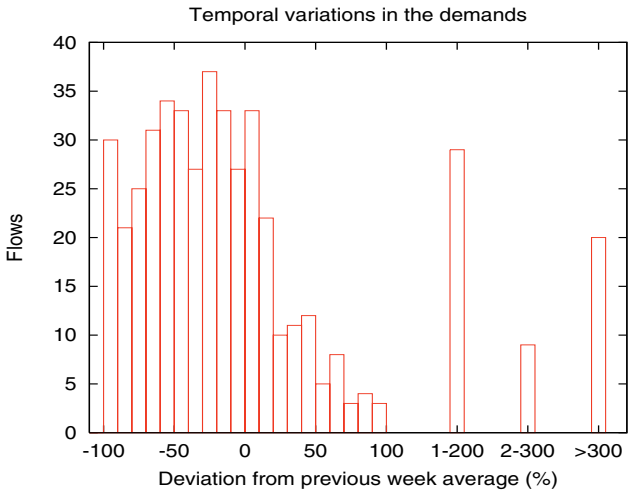


Figure 7.1: Flow deviations from previous week average

is associated with a specific topology, and is routed according to the corresponding routing table. This association might be in the form of a topology identifier in the packet header, or by using tunnelling in combination with a private address space [61].

7.3.1 Building logical topologies

We will now describe an algorithm for creating the alternate topologies. The algorithm is the same as the *Rich* algorithm introduced in section 4.3.2, where we proposed the use of multiple topologies for fast recovery from link failures. Here, we repeat the rules that control the topology creation.

We define a topology T as a set of nodes N and a set of links A. Given a network topology T_0, we build n additional topologies $T_1, \ldots, T_n$. We refer to T_0 as the *original* topology, and $T_1, \ldots, T_n$ as the *alternate* topologies. The alternate topologies are copies of the original topology T_0, with the difference that a subset of the links are removed in each alternate topology. Importantly, links are removed in such a way that each alternate topology

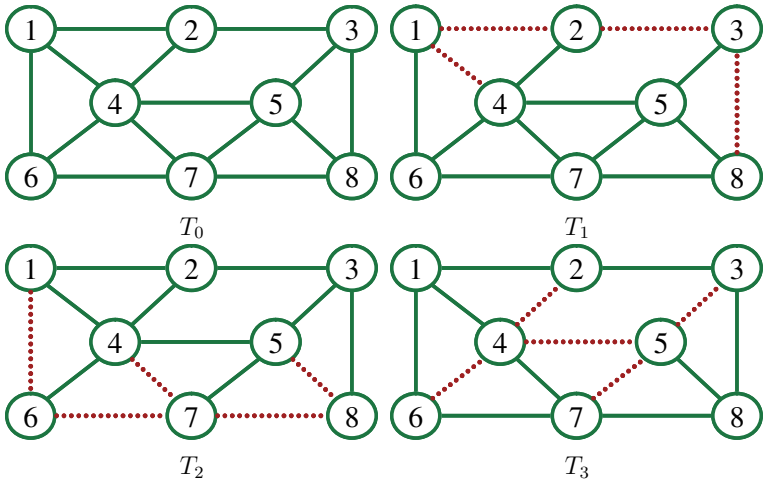


Figure 7.2: Example topologies

$T_1, \ldots, T_n$ is still connected, and thus all nodes are still reachable in all topologies. Figure 7.2 shows an example of how three alternate topologies can be built so that all links are removed in one of the topologies.

Input to our algorithm is the original topology T_0, and the number n of desired alternate topologies. The algorithm then iterates through all links and tries to remove each of them in one of the topologies T_i. A link can only be removed from a topology if doing so does not disconnect the topology. If a link cannot be removed in T_i, we try again in topology $T_{(imodn)+1}$ until all alternate topologies have been tried. For each link we want to remove, a new topology is chosen as the first T_i we try, so that the number of removed links is approximately equal over the different T_i.

The algorithm results in n alternate topologies $T_1, \ldots, T_n$ with two important properties.

1. All topologies are connected, so that in each topology, there is a valid routing path between each pair of nodes.

2. All links are removed in exactly one of the alternate topologies. We denote by $T(l)$ the topology where link l is removed.

In section 4, we showed that a surprisingly small number of alternate topologies are needed to fulfill these two properties. This way of generating the alternate topologies ensures that when a link becomes overloaded, there will always exist a topology where the congested link is not used to forward traffic.

7.4 Load balancing using Multi-Topology routing

We describe two different methods for load balancing based on MT routing. We refer to the two as the *global* and the *local* methods respectively.

7.4.1 Global load balancing

The global method takes as input the network graph, the link capacities, and the available traffic matrix estimate. Based on this, it assigns traffic flows to topologies in a way that tries to minimize the chances of congestion, by minimizing the maximum link utilization in the network. Note that we use the notion of a flow to describe the aggregate of traffic going from an ingress to an egress, and we to not distinguish traffic on a finer granularity. We foresee the global method being used in a centralized off-line tool that calculates the consequences of different routings.

The global method starts out with routing all traffic demands according to the original topology T_0, and then moves some selected flows over to the alternate topologies in order to minimize the utilization of the most loaded link in the network. The load distribution in the network in the initial stage of the algorithm is determined by the link weights used in T_0. Our method is agnostic to the setting of these weights.

Pseudo-code for the algorithm that assigns ingress-egress flows to alternate topologies is given in algorithm7.1. Each iteration in the outer while loop starts by identifying the most utilized link a_{max} in the network, and the flows $\mathcal{F}$ that are routed over this link. The inner for loop then iterates over $\mathcal{F}$ to evaluate the consequences of moving each flow. The flow that gives the lowest new max utilization is moved from T_0 to the topology $T(a_{max})$ where a_{max} is avoided. The algorithm terminates when no flow is found that gives a lower max utilization. The output of the algorithm is a mapping that is used by the ingress nodes to assign each flow to the correct topology.

Algorithm 7.1: Topology assignment algorithm

```
1  continue ⇐ true
2  while continue do
3      a_max ⇐ most loaded link in the network
4      u_max ⇐ utilization of a_max
5      F ⇐ set of flows routed through a_max in T_0
6      u' ⇐ ∞
7      forall f ∈ F do
8          Move f to T(a_max)
9          u_f ⇐ new max utilization in the network
10         if u_f < u' then
11             u' ⇐ u_f
12             f' ⇐ f
13         end
14         Move f to T_0
15     end
16     if u' < u_max then
17         Move f' to T(a_max)
18     else
19         continue ⇐ false
20     end
21 end
```

Since we operate only on coarse ingress-egress flows, the number of flows routed over any single link will be modest. Hence, we evaluate the consequences of moving each flow to $T(a_{max})$ before deciding which flow is actually moved. If we wanted to decrease the running time of our algorithm, this could be done by basing this selection on randomness or the size of the flows instead of testing for each flow.

Responding to changing traffic demands

Network operators use various tools to monitor the state of their network, and to detect changes in the traffic demands [107]. When such changes are detected, we can use the above algorithm to calculate a new mapping from flows to topology. A key point here is that only this mapping needs to be changed to respond to changes in the demands. This gives two important advantages over traffic engineering methods based on link weight tuning. First, the calculation of this mapping is fairly simple and takes considerably shorter time than a weight search heuristic. Second, and most important, changing the mapping does not trigger a new IGP convergence in the network, and it is transparent to the BGP route selection process.

Because our algorithm has low complexity and does not introduce instability in the routing, we argue that this recalculation can be done quite frequently. We believe it can be done on a time scale of hours, or perhaps even minutes, depending on how fast changes in the traffic demands can be measured.

7.4.2 Local load balancing

With the local method, no prior global calculations or setup is needed, except the construction of the alternate topologies. Instead, traffic is moved to an alternate topology that avoids the congested link by the upstream node local to the congestion. The local method does not require any knowledge of the traffic matrix, and the method responds immediately to unexpected bursts or changes in demands.

We propose a simple mechanism inspired by Random Early Detection [108], where packets are moved to the topology where the congested link is avoided with a probability that increases as the buffer occupancy gets closer to 100%. When there is no congestion in the network, all traffic is routed according to the original topology T_0. Packets are moved to the alternative

topology by an intermediate node with a probability p that is 0 if the buffer occupancy stays below a certain threshold t_{buf}, and then increases linearly to 1 when the buffer is full. To prevent looping, we never let a packet chance topology more than once.

This local moving of traffic from one topology to another increases the chances of packet reordering, with its adverse effects on TCP performance. Packet reordering can be reduced by a mechanism that tries to select packets from the same TCP flow when deciding what traffic should be moved to another topology, much like what is done today when traffic is split between several equal cost paths using hashing functions.

7.4.3 Discussion of the global and local methods

While the global method tries to *prevent* congestion by assigning flows to topologies that avoid the most heavily loaded links, the local method instead *resolves* congestion by routing the traffic on an alternative path from the point of congestion.

The difference between the two methods is clearly visible in figure 7.3. The figure shows the simulated load and the corresponding buffer occupancy over a 5 seconds period for the two 155 Mbps links leading to a node n in the GEANT network (further description of the simulation environment and setup will be given in section 7.5). Figure 7.3a) shows how the global method distributes the traffic towards the node relatively evenly between the two links. For most of the time, the link utilization is kept below the capacity, and the buffer occupancy stays close to 0, as seen in figure 7.3b). With the local method, the load on one of the incoming links is far greater than on the other, as seen in figure 7.3c). This results in a higher variation in the buffer occupancy (figure 7.3d). When the buffer occupancy approaches the capacity, the local method responds by moving traffic over to the topology that avoids the congested link. This traffic will follow the shortest path in the alternate topology. Traffic destined for node n will finally finds its way through the other link attached to n, where we can observe it as small spikes.

The main advantage of the local method is that no knowledge of the traffic demand is needed, and that it immediately adapts to shifts and short term fluctuations in the traffic. The price to pay for this flexibility is higher variation in buffer occupancy than with the global method, which might affect the delay experienced by the packets.

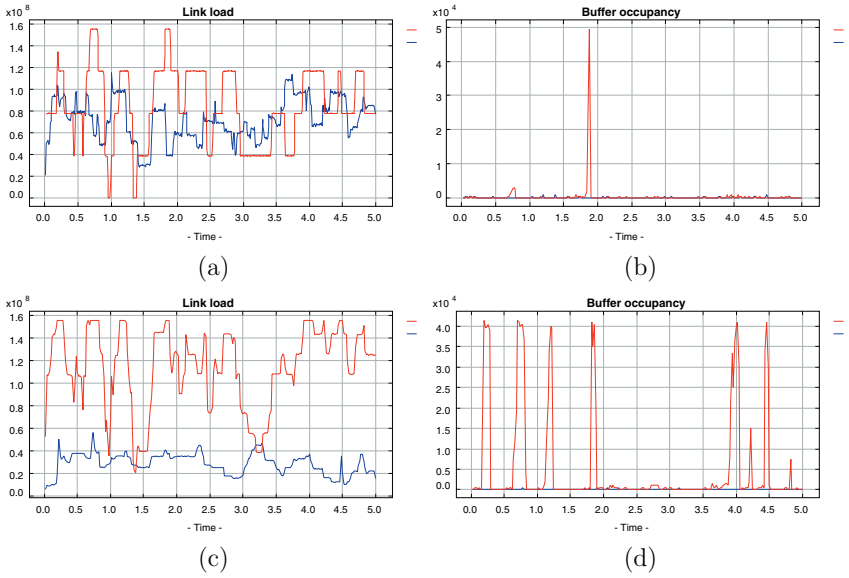


Figure 7.3: Link load and buffer occupancy of the two links into a node with the global (a,b) and the local (c,d) method

7.5 Evaluation

We have evaluated our proposed methods using simulations of the pan-European GEANT network, with real intradomain traffic matrixes [106]. We conduct simulations on the packet level, using the J-sim simulation framework [20]. Network traffic has been shown to have significant short term variations, with self-similar characteristics [109]. To capture these effects, we let each ingress-egress flow be modelled as a self-similar packet source at the ingress node, producing the desired average load to the egress. Our self-similar packet sources are based on multiplexing several heavy-tailed ON/OFF sources, and are set to produce traffic with Hurst parameter 0.9. This method for generating self-similar traffic has been shown to give good control of the traffic characteristics compared to other known methods [110]. Our traffic setup results in large short term variations in link utilization, as seen in figure 7.3. We use relatively small buffers (50 kBytes) in the routers. Combined with the large short term variations in link utilization, this setup gives a relatively high change of packet drops.

We use destination-based shortest path hop-by-hop forwarding, but the paths taken by packets will depend on which logical topology the packet belongs to. We use $n = 5$ alternate topologies, and we use original GEANT IGP link weights in all topologies $T_0, \ldots, T_n$. These link weights are mainly based on the inverse of the link capacities, with some manual tuning [106]. For our local method, we use a buffer threshold t_{buf} of 0.75.

To evaluate the performance of our methods, we compare them to the load balancing method proposed by Fortz and Thorup [69]. They define a cost function that penalizes routing traffic over heavily utilized links, and use a heuristic to tune the link weights in order to minimize this cost function. We will refer to this method as FT.

7.5.1 Robustness to increase in demand

Figure 7.4 shows the packet loss with our global and local methods, and with the FT heuristic. The simulations are based on three different 15-minutes traffic matrix captured between 12:45 and 13:00 from our dataset. The offered load is increased along the x-axis by multiplying all the flows with a scaling factor. The plotted values are average values from 20 simulations with different seeds. As can be seen in figure 7.4, the day-to-day variations are significant. The three plots are made from days with quite different traffic

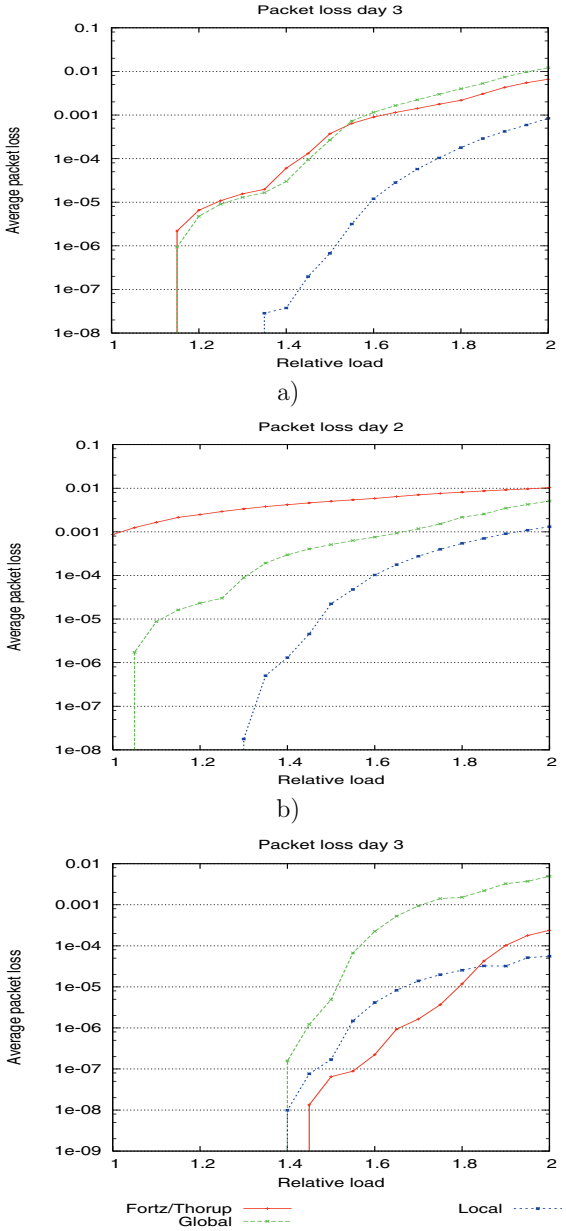


Figure 7.4: Packet loss with increasing load

loads in the network.

Figure 7.4a) shows the situation on a quite representative weekday, with average traffic loads. We have run simulations for several other days with similar results. On this day, we see how the local method gives substantially less packet loss than the other methods, the improvement is of a factor 10-100 compared to the FT heuristic. The global method performs slightly better than FT when the load factor is relatively low, and slightly worse for higher loads. This effect is seen because our global method only takes the load on the most loaded link into account, while the FT heuristic also looks at other highly loaded links. Hence the global method performs better as long as packet loss is confined to a single link.

As illustrated in figure 7.4b), there are some days when the FT method never overtakes the global method even for a high load factor. This is a day with high traffic load, and we experience packet loss even for a low setting of the scaling parameter. Again we see that the local method performs significantly better than the other methods, and that the FT heuristic performs relatively better at high loads.

Figure 7.4c) shows the situation on a Saturday, when the load in the network is lower than average. On this day, the traffic demands must be scaled higher before we experience any packet loss. We see that for this day, the FT heuristic performs better than both the global and local methods, except for very high loads. We have included this plot to illustrate that the relative performance of the different methods has significant variations. However, the situation on this day was not typical for weekend days.

7.5.2 Robustness to changes in demand over time

We test how the different load balancing methods handle the day to day variations described in section 7.2. Figure 7.5 shows the packet loss for 7 consecutive days. The FT method and our (static) global method are optimized for the average demands from the previous week. The dynamic global method is optimized for each day. In order to stress the network and get interesting results, the original GEANT traffic demands are multiplied with a factor 1.3. Again, the results shown are averages over 20 simulations with different seeds.

The traffic demands, and hence also the packet loss, varies significantly between weeks. Therefore, we plot results from two different weeks. Figure 7.5a) shows a week with relatively low demands, while figure 7.5b) shows

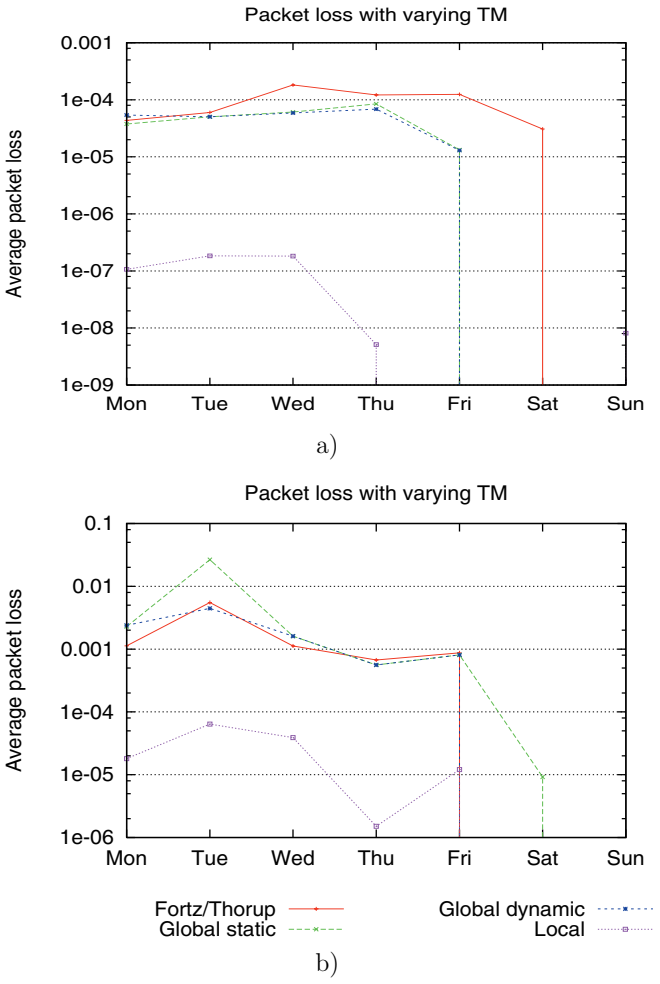


Figure 7.5: Packet loss with demands that change over time.

a week with higher demands. Again, the main tendency is the same as in figure 7.4; the local method gives significantly less packet loss than the other methods. Also, the performance of the global methods relative to the FT method is better when the network is not severely overloaded. A final point to notice is the reduced traffic in the weekends (days 6 and 7), when we see almost no packet loss.

7.6 Summary

In this chapter, we have argued that current traffic engineering methods based on link weight tuning are not flexible enough when facing variations in the traffic demands. Using real traffic demands from the GEANT network, we have demonstrated that such variations are indeed significant.

We have introduced the use of Multi-Topology routing to improve the load balancing in intradomain IP networks. This is a much more flexible approach than link weigh tuning, since it avoids the IGP re-convergence and the adverse traffic effects that are triggered by link weight changes. We have described two different mechanisms based on using MT routing. The *global* method uses the available estimates of the traffic demands to assign ingress-egress flows to different topologies, in order to avoid the most heavily loaded link. The *local* method needs no prior knowledge of the traffic demands, and responds to congestion by locally moving traffic to an alternate topology that avoids the congested link. Our evaluations show that our global method performs as good as previous methods based on link weight tuning with respect to packet loss, while our local method performs substantially better.

Chapter 8

Conclusions

In this work, we have argued that the increased use of the Internet as a transport medium for interactive and real-time applications makes current recovery mechanisms insufficient. We have introduced and discussed several mechanisms aimed at giving very fast response to challenges to normal operation in computer networks. A main theme has been fast recovery from component failures.

8.1 Resilient Packet Ring

Our first contribution was in the context of a recent link layer technology called Resilient Packet Ring (RPR). Fast (sub 50 ms) recovery from link or node failures has been a central motivation behind the development of RPR. RPR is designed to work in a dual ring topology, and takes advantage of this specialized topology by letting one ringlet serve as a backup when the other fails.

To give recovery from link and node failures, RPR offers two distinct protection mechanisms called wrapping and steering. In this work, we have analyzed the RPR resilience mechanisms with a particular emphasis on the mechanism used to prevent packet reordering. We have shown that the stabilization timer used for this purpose is the main contributor to the recovery time, and that it often prevents RPR from giving sub 50 ms recovery.

We have suggested three different methods that reduce the recovery time in RPR while maintaining the in-order delivery guarantee. The first method is based on finding a minimum safe value for the topology stabilization timer,

and requires no changes to the forwarding procedure used in RPR. The two other methods moves the responsibility for avoiding packet reordering to the receiver nodes, and require somewhat larger changes to the RPR standard.

We have shown that our proposed mechanisms significantly reduce the amount of packet loss and the experienced disruption in packet delivery. In our simulated scenarios packet loss was reduced by 60-90% compared to the RPR standard, and the recovery times were well below 50 ms. The choice between our proposed methods becomes a tradeoff between recovery time and the amount of packet loss on the one hand, and the amount of changes that are introduced in the RPR standard on the other hand.

8.2 Resilient Routing Layers

The protection mechanisms in RPR take advantage of the special ring topology to respond rapidly to a failure. Several existing recovery schemes for general mesh networks imitate this by imposing logical sub-topologies (such as trees [30] or rings [33]) on the network and using these to form recovery paths when a failure occurs. This is also the basic idea behind Resilient Routing Layers (RRL) introduced in this work.

RRL is a local protection scheme that guarantees fast and loop-free recovery from any single link failure in arbitrary biconnected networks. It does not affect the routing during normal failure-free operation. Since the rerouting decisions are taken by the detecting node based on pre-calculated information without any signalling to neighboring nodes, RRL can respond very rapidly to a failure. The RRL scheme can be used in connection-oriented networks, but we believe that the most favorable application domain for our approach is in connectionless networks with destination-based hop-by-hop forwarding.

In RRL, we build a small number of *backup configurations* that are used to forward traffic when a link failure is detected. Each backup configuration is a connected sub-topology that contains all the nodes and a subset of the links in the original topology. The backup configurations are built so that there is a valid routing path between all node pairs in the network, and all links are excluded in at least one of the backup configurations. RRL guarantees recovery from a single link failure in arbitrary biconnected topologies.

We have presented three different algorithms for creating RRL backup configurations, with different performance objectives. The different algorithms present tradeoffs between the amount of state information that is

stored in the routers, the recovery path lengths, and the ability to resist more than one concurrent failure. We have shown that RRL can be realized with a very limited number of backup configurations, and that the number of backup configurations needed grows very slowly with increasing network size. We have also shown that RRL gives backup paths that are not much longer than the optimal, and that the probability of recovering traffic from two concurrent link failures is high.

8.3 Multiple Routing Configurations

Based on the principles introduced in RRL, we have presented Multiple Routing Configurations (MRC). MRC is an extension of RRL in two ways. First it is more general, since it guarantees recovery from both link and node failures. Second it is more specific, as it is designed only for connectionless networks with shortest path routing.

Like RRL, MRC is based on the idea of using backup configurations to give loop-free recovery paths to all destinations after a failure. In MRC, no links are removed in the backup configurations. Instead, routing is restricted in parts of the network in the backup configurations by assigning very high links weights to selected links. Careful assignment of link weights allows MRC to handle both link and node failures with the same mechanism, and without knowing whether the connectivity loss is caused by a failed link or a failed node.

When a proactive recovery scheme like MRC is deployed in a network, the normal rerouting process can be delayed. By suppressing the advertisement of a failure for a given period of time, the global rerouting process will only be triggered by more permanent failures. Since a significant fraction of all failures are short lived, this increases the stability of the network. Delaying the re-convergence process also makes it easier to use techniques like fast hellos for rapid failure detection without compromising stability.

We have given a formal description of MRC, and we have presented an algorithm that creates backup configurations in accordance with the defined requirements. The performance of our method is evaluated with respect to state requirements, backup path lengths and load distribution. We have shown that MRC can be realized with a very limited number of backup configurations, requiring a moderate amount of additional state to be stored in the routers. The recovery paths in the backup configurations are not much

longer than the optimal achievable by a local recovery scheme. However, our evaluations show that the backup configurations created by our algorithm can sometimes lead to increased chances of congestion after a failure because recovered traffic is routed over already highly utilized links.

8.4 MRC routing performance

Motivated by the evaluations of the post-failure load distribution in MRC, we have described methods that can reduce the chances of congestion after a link failure. This problem has not previously been addressed in the context of a proactive recovery scheme for traditional IP routing protocols. We have argued that such mechanisms are important, since much of the purpose of fast protection mechanisms at the IP layer is to drop as few packets as possible. Our method does not compromise performance in the failure-free case, and it is strictly pre-configured; no calculations are necessary after the failure.

We have presented an algorithm that takes the estimated traffic matrix into account when the MRC backup configurations are created. The algorithm creates backup configurations so that the chances of spreading the traffic in the network are best for the most severe link failures. Further, we have presented a heuristic for finding link weights in the backup configurations that distributes the recovered traffic in a beneficial way.

Our mechanism has been evaluated using simulations of several real and synthetically generated networks. Our evaluations show that the post failure load distribution has been significantly improved compared to the MRC algorithm that does not take the traffic matrix into account. In fact, we achieve a load distribution that is better than after a full IGP re-convergence in most cases.

8.5 Multi Topology load balancing

Our work with the traffic-aware MRC methods demonstrated how the use of several logical topologies in an AS can give significant freedom with respect to distributing traffic over the available links in the network. On this background, we have proposed the use of Multi Topology routing as a traffic engineering tool in connectionless IP networks.

The main advantage of our Multi Topology approach is that it is more

flexible than methods based on tuning link weights when faced with unexpected changes in the traffic patterns. By looking at realistic traffic matrixes from a pan-European research network, we have demonstrated that such traffic variations are significant.

In our approach we use one of the algorithms proposed in RRL to create alternate topologies so that all links in the network are avoided in one topology. We have proposed two different ways to use these alternate topologies to avoid congested links in heavily loaded networks. The *global* method moves some source-destination flows over in alternate topologies in order to avoid congested links. When the traffic demands change, only this mapping from source-destination pairs to alternate topology needs to be re-calculated. Thus, we avoid the global re-convergence and the associated instability that follows from changing link weights to comply with the new traffic situation. In the *local* method, traffic is moved to an alternate topology locally by the node that experiences congestion. This way, the local method can respond rapidly to traffic changes, without any knowledge of the underlying traffic matrix.

Our simulations show that our proposed methods perform well with respect to reducing the chances of packet loss. The global method performs as well as a well known load balancing method based on link weight optimization, while the local method performs significantly better in our simulated scenarios.

8.6 Concluding remarks

For many years, it has been clear that the single shortest path routing used in the most common intradomain protocols (OSPF and IS-IS) is an obstacle to improving resilience and performance in IP networks. A basic challenge is to find a routing process that gives the routers more options with respect to where packets should be forwarded, and allows them to dynamically choose between the available options.

One step in this direction is the deployment of Equal Cost Multi Path (ECMP) forwarding, which allows a router to split traffic towards a destination equally over several available equal cost paths. This mechanism can thus give a better distribution of the load, and improved resilience by using the remaining next-hops as backup if one of them fails. A more far-reaching approach is the introduction of MPLS, which gives much more flexibility in

the routing. With MPLS, the routing path from each ingress router to each egress router can be explicitly set up in advance. An offline tool can be used to calculate these paths in ways that take different traffic engineering goals into consideration, and the resilience can be improved by setting up explicit backup paths that are ready for use when a failure occurs.

However, MPLS introduces significant changes in both the control and forwarding logic in the routers, and not all operators can or want to use MPLS in their network. In these cases, we believe that the use of virtualization in the form of multiple logical topologies in a network is a very promising approach for achieving both resilience and improved control over the routing. In this work, RRL and MRC are examples of how a multi-topology approach can be used to give fast recovery. We have also illustrated how Multi-Topology routing can be used as an effective traffic engineering tool.

To facilitate such uses of multiple logical topologies in a network, we think that the existing Multi-Topology routing mechanisms developed by the IETF should be further developed. First, mechanisms for switching data packets from one topology to another in-flight should be clearly defined. Rules for when topology-transitions can or cannot be done are needed in order to avoid possible forwarding loops. Second, different ways of explicitly marking data packets as belonging to a given topology should be standardized. In particular, overloading of some bits in the ToS field of the IP packet header as a topology identifier should be investigated.

8.7 Future research directions

There are several possibilities for further research based on the methods presented in this work.

In the context of MRC, several practical deployment issues can be addressed. One challenge is how to best organize the routing and forwarding tables in order to minimize the state overhead and make the lookup procedure as simple as possible. MRC requires one logical forwarding table per configuration, but we believe that these can be structured so that the actual amount of state information is reduced significantly compared to an n-fold increase when n configurations are used. Some early thoughts on this are presented in [104].

Another issue is the use of MRC in networks where some destination addresses can be reached through several egress routers. In such scenarios, it

would be beneficial if traffic is rerouted to the alternate egress router if one of them fails. Guaranteeing that this will be the case would demand some changes to MRC in its present form.

An interesting question is also how the MRC backup configurations can be re-constructed when there is a permanent change to the network topology. With MRC as described in this work, this requires new configurations to be constructed from scratch based on the altered topology. An incremental algorithm that takes the existing backup configurations as a starting point and only makes the necessary changes would make MRC more robust when faced with such network dynamics.

Fast protection mechanisms are primarily useful for real-time applications like IP-TV with strict demands on packet loss and availability. We believe that such traffic increasingly will be sent using some kind of multicast protocol. An interesting research topic is therefore whether MRC or other similar mechanisms can be used to protect also multicast traffic.

We believe that multiple parallel logical topologies can be a useful tool also for dealing with other challenges to a network. One possible area might be security and protection against outsider attacks. Work on robust routing with byzantine robustness rely at its core on a mechanism that allows a source to identify and select between several possible routes to the destination [111, 112]. By splitting information and sending it over several logical topologies, or by switching between sending traffic in different topologies, it can be made much harder to intercept or eavesdrop on a connection. If the logical topologies are created in a similar fashion to what we have proposed for MRC in this work, there will exist a topology that excludes any single node in a network from its routing paths. This way, mechanisms could be created so that several routers must cooperate in order to disrupt communication between any two nodes in the network.

There are also several open questions in the context of using Multi-Topology routing as a traffic engineering tool. The evaluations in this work indicate that the local method where traffic is switched to an alternate topology at the point of congestion performs very well compared to existing techniques. However, this method can in some cases lead to reordering of packets, which again has an adverse impact on TCP performance. As future work, it would be interesting to investigate how severe this packet reordering is with the local method, and how it can be minimized.

Bibliography

[1] D. D. Clark, "The design philosophy of the DARPA internet protocols," *SIGCOMM, Computer Communications Review*, vol. 18, no. 4, pp. 106–114, Aug. 1988.

[2] D. Hutchison and J. Sterbenz, "Resilinets: Multi-level resilient and survivable networking initiative," http://www.comp.lancs.ac.uk/resilinets/, Aug. 2006.

[3] "Resilinets wiki," http://wiki.ittc.ku.edu/resilinets_wiki/.

[4] A. Fumagalli and L. Valcarenghi, "IP restoration vs. WDM protection: is there an optimal choice?" *IEEE Network*, vol. 14, no. 6, pp. 34–41, Nov. 2000.

[5] L. Sahasrabuddhe, S. Ramamurthy, and B. Mukherjee, "Fault management in IP-over-WDM networks: WDM protection vs IP restoration," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 1, pp. 21–33, Jan. 2002.

[6] G. Iannacone, C.-N. Chuah, S. Bhattacharyya, and C. Diot, "Feasibility of IP restoration in a tier 1 backbone," *IEEE Network*, pp. 13–19, Mar. 2004.

[7] A. Basu and J. G. Riecke, "Stability issues in OSPF routing," in *Proceedings of SIGCOMM*, San Diego, California, USA, Aug. 2001, pp. 225–236.

[8] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian, "Delayed Internet Routing Convergence," *IEEE/ACM Transactions on Networking*, vol. 9, no. 3, pp. 293–306, June 2001.

[9] C. Boutremans, G. Iannaccone, and C. Diot, "Impact of link failures on VoIP performance," in *Proceedings of International Workshop on Network and Operating System Support for Digital Audio and Video*, 2002, pp. 63–71.

[10] D. Watson, F. Jahanian, and C. Labovitz, "Experiences with monitoring OSPF on a regional service provider network," in *ICDCS '03: Proceedings of the 23rd International Conference on Distributed Computing Systems*. Washington, DC, USA: IEEE Computer Society, 2003, pp. 204–213.

[11] P. Francois, C. Filsfils, J. Evans, and O. Bonaventure, "Achieving subsecond IGP convergence in large IP networks," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 2, pp. 35 – 44, July 2005.

[12] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C.-N. Chuah, and C. Diot, "Characterization of failures in an IP backbone network," in *Proceedings INFOCOM*, Mar. 2004.

[13] S. Iyer, S. Bhattacharyya, N. Taft, and C. Diot, "An approach to alleviate link overload as observed on an IP backbone," in *Proceedings INFOCOM*, Mar. 2003, pp. 406–416.

[14] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True, "Deriving traffic demands for operational IP networks: methodology and experience," *IEEE/ACM Transactions on Networking*, vol. 9, no. 3, pp. 265–280, June 2001.

[15] B. Fortz and M. Thorup, "Optimizing OSPF/IS-IS weights in a changing world," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 4, pp. 756 – 767, May 2002.

[16] W. J. Rapaport, "Philosophy of computer science: An introductory course," *Teaching Philosophy*, vol. 28, no. 4, pp. 319–341, 2005.

[17] P. J. Denning, D. E. Comer, D. Gries, M. C. Mulder, A. Tucker, A. J. Turner, and P. R. Young, "Computing as a discipline," *Communications of the ACM*, vol. 32, no. 1, pp. 9–23, Feb. 1989.

[18] R. Jain, *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation and modeling.* New York: John Wiley & Sons, 1991.

[19] S. Floyd and V. Paxon, "Difficulties in simulating the internet," *IEEE/ACM Transactions on Networking*, vol. 9, no. 4, pp. 392–403, Aug. 2001.

[20] H.-Y. Tyan, "Design, Realization and Evaluation of a Component-Based Compositional Software Architecture for Network Simulation," Ph.D. dissertation, Ohio State University, 2002.

[21] E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol label switching architecture," in *IETF*, RFC 3031, Jan. 2001.

[22] V. Sharma and F. Hellstrand, "Framework for multi-protocol label switching (MPLS)-based recovery," RFC3469, Feb. 2003.

[23] P. Pan and G. Swallow, "Fast reroute extensions to RSVP-TE for LSP tunnels," RFC4090, May 2005.

[24] R. Martin and M. Menth, "Backup capacity requirements for MPLS fast reroute," in *7. ITG Fachtagung Photonische Netze (VDE)*, Leipzig, Germany, Apr. 2006.

[25] *Synchronous Optical Network (SONET) Automatic Protection Switching.*

[26] A. Fumagalli, I. Cerutti, M. Tacca, F. Masetti, R. Jagannathan, and S. Alagar, "Survivable networks based on optimal routing and WDM self-healing rings," in *Proceedings INFOCOM*, vol. 2, New York, USA, Mar. 1999, pp. 726–733.

[27] O. Gerstel and R. Ramaswami, "Optical layer survivability - an implementation perspective," *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 10, pp. 1885–1899, Oct. 2000.

[28] S. Ramamurthy, L. Sahasrabuddhe, and B. Mukherjee, "Survivable WDM mesh networks," *Journal of Lightwave Technology*, vol. 21, no. 4, pp. 870–883, Apr. 2003.

[29] A. Itai and M. Rodeh, "The multi-tree approach to reliability in distributed networks," in *Foundations of Computer Science*, Oct. 1984, pp. 137–147.

[30] S. G. Finn, M. Medard, and R. A. Barry, "A novel approach to automatic protection switching using trees," in *Proc. ICC*, vol. 1, June 1997, pp. 272–276.

[31] M. Medard, S. G. Finn, and R. A. Barry, "Redundant trees for preplanned recovery in arbitrary vertex-redundant or edge-redundant graphs," *IEEE/ACM Transactions on Networking*, vol. 7, no. 5, pp. 641–652, Oct. 1999.

[32] R. Bartos and M. Raman, "A heuristic approach to service restoration in MPLS networks," in *Proc. ICC*, June 2001, pp. 117–121.

[33] W. D. Grover and D. Stamatelakis, "Cycle-oriented distributed preconfiguration: Ring-like speed with mesh-like capacity for self-planning network restoration," in *Proceedings IEEE International Conference on Communications (ICC)*, Atlanta, Georgia, USA, June 1998, pp. 537–543.

[34] J. Doucette, P. A. Giese, and W. D. Grover, "Combined node and span protection strategies with node-encircling p-cycles," in *Proceedings DRCN*, Oct. 2005.

[35] H. Huang and J. Copeland, "A series of hamiltonian cycle-based solutions to provide simple and scalable mesh optical network resilience," *IEEE Communications Magazine*, vol. 40, pp. 46 –51, Nov. 2002.

[36] D. A. Schupke, "Multiple failure survivability in WDM networks with p-cycles," in *IEEE ISCAS*, vol. 3, May 2003, pp. 866–869.

[37] D. Stamatelakis and W. D. Grover, "IP layer restoration and network planning based on virtual protection cycles," *IEEE Journal on selected areas in communications*, vol. 18, no. 10, Oct. 2000.

[38] *802.1D-2004, Standard for Local and Metropolitan Area Networks - Media Access Control (MAC) Bridges*, IEEE, 2004.

[39] S. Shah and M. Yip, "Ethernet automatic protection switching (EAPS)," RFC3619, Oct. 2003.

[40] *802.1s-2002, Standard for Local and Metropolitan Area Networks - Multiple Spanning Trees*, IEEE, 2002.

[41] J. Farkas, C. Antal, G. Tóth, and L. Westberg, "Distributed resilient architecture for ethernet networks," in *Proceedings DRCN*, Oct. 2005, pp. 515–522.

[42] *ANSI/IEEE Std. 802.5, 1989 Edition*, ANSI/IEEE, 1989, iEEE standard for Token Ring.

[43] F. E. Ross, "An overview of fddi: the fiber distributed data interface," *IEEE Journal on Selected Areas in Communications*, vol. 7, no. 7, pp. 1043–1051, Sept. 1989.

[44] J. Moy, "OSPF version 2," RFC2328, Apr. 1998.

[45] D. Oran *et al.*, "OSI IS-IS intra-domain routing protocol," RFC1142, Feb. 1990.

[46] G. Malkin, "RIP version 2," RFC 2453, Nov. 1998.

[47] B. Albrightson, J. Garcia-Luna-Aceves, and J. Boyle, "EIGRP - a fast routing protocol based on distance vectors," in *Proceedings Networld/Interop*, May 1994.

[48] D. L. Mills, "Exterior gateway protocol formal specification," RFC904, Apr. 1984.

[49] Y. Rekhter, T. Li, and S. Hares, "A border gateway protocol 4 (BGP-4)," RFC4271, Jan. 2006.

[50] P. Narvaez, K.-Y. Siu, and H.-Y. Tzeng, "Local restoration algorithm for link-state routing protocols," in *Proceedings ICCCN*, Boston, MA, USA, Oct. 1999, pp. 352–357.

[51] Y. Liu and N. Reddy, "A fast rerouting scheme for OSPF/IS-IS networks," in *Proceedings of ICCCN 2004*, Chicago, Illinois, USA, Oct. 2004, pp. 47–52.

[52] S. Rai, B. Mukherjee, and O. Deshpande, "IP resilience within an autonomous system: Current approaches, challenges, and future directions," *IEEE Communications Magazine*, vol. 43, no. 10, pp. 142–149, Oct. 2005.

[53] G. Schollmeier, J. Charzinski, A. Kirstädter, C. Reichert, K. J. Schrodi, Y. Glickman, and C. Winkler, "Improving the resilience in IP networks," in *Proceedings of HPSR*, Torino, Italy, June 2003, pp. 91–96.

[54] C. Reichert, Y. Glickmann, and T. Magedanz, "Two routing algorithms for failure protection in IP networks," in *Proceedings of the 10th IEEE Symposium on Computers and Communications (ISCC)*, June 2005, pp. 97–102.

[55] S. Nelakuditi *et al.*, "Failure insensitive routing for ensuring service availability," in *IWQoS'03 Lecture Notes in Computer Science 2707*, June 2003.

[56] S. Lee, Y. Yu, S. Nelakuditi, Z.-L. Zhang, and C.-N. Chuah, "Proactive vs. reactive approaches to failure resilient routing," in *Proceedings INFOCOM*, Mar. 2004.

[57] S. Nelakuditi, S. Lee, Y. Yu, Z.-L. Zhang, , and C.-N. Chuah, "Fast local rerouting for handling transient link failures," *Accepted for publication in IEEE/ACM Transactions on Networking*, 2007.

[58] Z. Zhong, S. Nelakuditi, Y. Yu, S. Lee, J. Wang, and C.-N. Chuah, "Failure inferencing based fast rerouting for handling transient link and node failures," in *Proceedings of IEEE Global Internet*, Mar. 2005.

[59] J. Wang, Z. Zhong, and S. Nelakuditi, "Handling multiple network failures through interface specific forwarding," in *Proceedings GLOBECOM*, Nov. 2006.

[60] M. Shand and S. Bryant, "IP fast reroute framework," IETF Internet Draft (work in progress), Oct. 2006, draft-ietf-rtgwg-ipfrr-framework-06.txt.

[61] S. Bryant, M. Shand, and S. Previdi, "IP fast reroute using not-via addresses," Internet Draft (work in progress), Oct. 2006, draft-bryant-shand-IPFRR-notvia-addresses-03.txt.

[62] S. Bryant and M. Shand, "A framework for loop-free convergence," Internet Draft (work in progress), Dec. 2006, draft-ietf-rtgwg-lf-conv-frmwk-00.txt.

[63] D. G. Cantor and M. Gerla, "Optimal routing in a packet-switched computer network," *IEEE Transactions on Computers*, vol. C-23, no. 10, pp. 1062–1069, Oct. 1974.

[64] D. Applegate and E. Cohen, "Making routing robust to changing traffic demands: Algorithms and evaluation," *IEEE Transactions on Networking*, vol. 14, no. 6, pp. 1193–1206, Dec. 2006.

[65] D. O. Awduche, "MPLS and Traffic Engineering in IP Networks," *IEEE Communications Magazine*, vol. 37, no. 12, pp. 42–47, Dec. 1999.

[66] D. O. Awduche and B. Jabbari, "Internet traffic engineering using multi-protocol label switching (MPLS)," *Computer Networks: The International Journal of Computer and Telecommunications Networking*, vol. 40, no. 1, pp. 111–129, Sept. 2002.

[67] Y. Wang, Z. Wang, and L. Zhang, "Internet traffic engineering without full mesh overlaying," in *Proceedings INFOCOM*, Apr. 2001, pp. 565–571.

[68] A. Sridharan, R. Guerin, and C. Diot, "Achieving near-optimal traffic engineering solutions for current OSPF/IS-IS networks," *IEEE/ACM Transactions on Networking*, vol. 13, no. 2, pp. 234–247, Apr. 2005.

[69] B. Fortz and M. Thorup, "Internet traffic engineering by optimizing OSPF weights." in *Proceedings INFOCOM*, 2000, pp. 519–528.

[70] C. Zhang, Z. Ge, J. Kurose, Y. Liu, and D. Towsley, "Optimal routing with multiple traffic matrices tradeoff between average and worst case performance," in *Proceedings ICNP*, Nov. 2005.

[71] A. Nucci, B. Schroeder, S. Bhattacharyya, N. Taft, and C. Diot, "IGP Link Weight Assignment for Transient Link Failures," in *18th International Teletraffic Congress*, Berlin, Germany, Aug. 2003.

[72] B. Fortz and M. Thorup, "Robust optimization of OSPF/IS-IS weights," in *INOC*, Oct. 2003, pp. 225–230.

[73] A. Sridharan and R. Guerin, "Making IGP routing robust to link failures," in *Proceedings of Networking*, Waterloo, Canada, 2005.

[74] F. Davik, M. Yilmaz, S. Gjessing, and N. Uzun, "IEEE 802.17 Resilient Packet Ring tutorial," *IEEE Communications Magazine*, vol. 42, no. 3, pp. 112–118, June 2004.

[75] *IEEE Standard 802.17-2004*, IEEE, June 2004.

[76] A. Kvalbein and S. Gjessing, "Analysis and improved performance of RPR protection," in *Proceedings 12th IEEE International Conference on Networks (ICON 2004).*, vol. 1, Singapore, Nov. 2004, pp. 119–124.

[77] ——, "Protection of RPR strict order traffic," in *Proceedings 14th IEEE Workshop on Local and Metropolitan Area Networks (LANMAN 2005).*, Chania, Crete, Sept. 2005.

[78] F. Davik, A. Kvalbein, and S. Gjessing, "An analytical bound for convergence of the resilient packet ring aggressive mode fairness algorithm," in *Proceedings 40th IEEE International Conference on Communications (ICC 2005)*, Seoul, Korea, May 2005.

[79] ——, "Improvement of resilient packet ring fairness," in *Proceedings 48th IEEE Global Telecommunications Conference (GLOBECOM 2005)*, St.Louis, Missouri, USA, Nov. 2005.

[80] W. Stevens, "TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms," in *IETF*, RFC 2001, Jan. 1997.

[81] E. Blanton and M. Allman, "On making TCP more robust to packet reordering," in *ACM Computer Communication Review*, vol. 32, no. 1, Jan. 2002.

[82] W. Willinger, M. Taqqu, R. Sherman, and D. Wilson, "Self-similarity through high-variability: Statistical analysis of Ethernet LAN traffic at the source level," *IEEE/ACM Transactions on Networking*, vol. 5, no. 1, pp. 71–86, Feb. 1997.

[83] A. Kvalbein, A. F. Hansen, T. Čičić, S. Gjessing, and O. Lysne, "Fast recovery from link failures using resilient routing layers," in *Proceedings 10th IEEE Symposium on Computers and Communications (ISCC)*, June 2005.

[84] I. Theiss and O. Lysne, "FRoots, a fault tolerant and topology agnostic routing technique,," *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 10, pp. 1136–1150, Oct. 2006.

[85] A. F. Hansen, T. Čičić, S. Gjessing, A. Kvalbein, and O. Lysne, "Resilient routing layers for recovery in packet networks," in *Proceedings of International Conference on Dependable Systems and Networks (DSN)*, June 2005.

[86] M. A. Weiss, *Data Structures and Algorithm Analysis*. Benjamin/Cummings, 1992.

[87] A. F. Hansen, O. Lysne, T. Cicic, and S. Gjessing, "Fast proactive recovery from concurrent failures," in *Proceedings 42nd IEEE International Conference on Communications (ICC 2007)*, Glasgow, UK, June 2007.

[88] "Rocketfuel topology mapping," WWW, http://www.cs.washington.edu.

[89] A. Medina, A. Lakhina, I. Matta, and J. Byers, "BRITE: An approach to universal topology generation," in *Proceedings of IEEE MASCOTS*, Aug. 2001, pp. 346–353.

[90] T. Bu and D. Towsley, "On distinguishing between internet power law topology generators," in *Proceedings of IEEE INFOCOM*, New York, June 2002, pp. 638–647.

[91] L. Li *et al.*, "A first-principles approach to understanding the internet's router-level topology," in *ACM SIGCOMM*, Portland, OR, Aug. 2004.

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

[93] A. Kvalbein, A. F. Hansen, T. Čičić, S. Gjessing, and O. Lysne, "Fast IP network recovery using multiple routing configurations," in *Proceedings INFOCOM*, Apr. 2006.

[94] P. Psenak, S. Mirtorabi, A. Roy, L. Nguen, and P. Pillay-Esnault, "MT-OSPF: Multi topology (MT) routing in OSPF," IETF Internet Draft (work in progress), Nov. 2006, draft-ietf-ospf-mt-07.txt.

[95] T. Przygienda, N. Shen, and N. Sheth, "M-ISIS: Multi topology (MT) routing in IS-IS," Internet Draft (work in progress), Oct. 2005, draft-ietf-isis-wg-multi-topology-11.txt.

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

[97] M. Menth and R. Martin, "Network resilience through multi-topology routing," in *Proceedings of the 5th International Workshop on Design of Reliable Communication Networks (DRCN)*, Oct. 2005.

[98] M. J. O'Mahony, "Results from the COST 239 project. Ultra-high capacity optical transmission networks," in *Proceedings of the 22nd European Conference on Optical Communication (ECOC'96)*, Sept. 1996, pp. 11–14.

[99] D. S. Johnson, "Approximation algorithms for combinatorial problems," in *Proc. of the 5. annual ACM symp. on Theory of computing*, 1973, pp. 38–49.

[100] A. Kvalbein, T. Čičić, and S. Gjessing, "Post-failure routing performance with multiple routing configurations," in *Proceedings INFOCOM*, May 2007.

[101] G. Ramalingam and T. Reps, "An incremental algorithm for a generalization of the shortest-path problem," *J. Algorithms*, vol. 21, no. 2, pp. 267–305, 1996.

[102] A. Nucci, A. Sridharan, and N. Taft, "The problem of synthetically generating IP traffic matrices: Initial recommendations," *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 3, pp. 19–32, July 2005.

[103] M. Roughan, "Simplifying the synthesis of internet traffic matrices," *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 5, pp. 93–96, Oct. 2005.

[104] G. Apostolopolous, "Using multiple topologies for IP-only protection against network failures: A routing performance perspective," ICS FORTH, Crete, Greece, Tech. Rep., Apr. 2006.

[105] R. Teixeira, A. Shaikh, T. Griffin, and G. M. Voelker, "Network sensitivity to hot-potato disruptions," in *Proceedings of SIGCOMM*, Portland, Oregon, USA, Aug. 2004, pp. 231–244.

[106] S. Uhlig, B. Quoitin, J. Lepropre, and S. Balon, "Providing public intradomain traffic matrices to the research community," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, pp. 83–86, Jan. 2006.

[107] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, and J. Rexford, "Netscope: Traffic engineering for IP networks," *IEEE Network Magazine*, vol. 14, no. 2, pp. 11–19, Apr. 2000.

[108] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, Aug. 1993.

[109] V. Paxson and S. Floyd, "Wide-area traffic: The failure of poisson modeling," *IEEE/ACM Transactions on Networking*, vol. 3, no. 3, pp. 226–244, June 1995.

[110] G. Horn, A. Kvalbein, J. Blomskøld, and E. Nilsen, "An empirical comparison of generators for self similar simulated traffic," *Elsevier Performance Evaluation*, vol. 64, no. 2, pp. 162–190, Feb. 2007.

[111] R. Perlman, "Network layer protocols with byzantine robustness," MIT, Tech. Rep. MIT-LCS-TR-429, Oct. 1988.

[112] ——, "Routing with byzantine robustness," Sun Labs, Tech. Rep. TR-2005-146, Aug. 2005.

Appendix A

Publication list

Most of the material in this work has previously been published in the proceedings of various international conferences. Here we provide a list of the publications that corresponds to each chapter, and a discussion of the contributions from the various authors.

Chapter 3

Title: Analysis and improved performance of RPR protection
Authors: Amund Kvalbein and Stein Gjessing
Published: 12th IEEE International Conference on Networks (ICON). Pages 119-124, vol.1. Singapore, Nov. 16-19, 2004.
Author contributions: All writing, analysis and simulations are performed by Amund Kvalbein. The possibility of letting the receiver discard out-of-order packets was first mentioned by Olav Lysne. The role of Stein Gjessing was mainly related to improving the structure and consistency of the presentation.

Title: Protection of RPR strict order traffic
Authors: Amund Kvalbein and Stein Gjessing
Published: 14th IEEE Workshop on Local and Metropolitan Area Networks (LANMAN) Chania, Crete, Sept. 18-21, 2005.
Author contributions: All writing, analysis and simulations are performed by Amund Kvalbein. The possibility of letting the receiver discard out-of-order packets was first mentioned by Olav Lysne. The proposed mechanisms were developed jointly by Amund Kvalbein and Stein Gjessing.

Chapter 4

Title: Fast recovery from link failures using Resilient Routing Layers
Authors: Amund Kvalbein, Audun Fosselie Hansen, Tarik Čičić, Stein Gjessing and Olav Lysne
Published: 10th IEEE Symposium on Computers and Communications (ISCC). Pages 554-560. Cartagena, Spain, Jun. 27-30, 2005.
Author contributions: Most of the writing was done by Amund Kvalbein. The algorithms presented in the paper were developed by Amund Kvalbein, but all authors contributed to the underlying ideas. The simulations were performed by Amund Kvalbein and Audun Fosselie Hansen.

Chapter 5

Title: Fast IP Network Recovery using Multiple Routing Configurations
Authors: Amund Kvalbein, Audun Fosselie Hansen, Tarik Čičić, Stein Gjessing and Olav Lysne
Published: 25th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM). Barcelona, Spain, Apr. 23-29, 2006.
Author contributions: Most of the writing was done by Amund Kvalbein, but all authors took part in the writing. The algorithm presented in the paper was developed by Amund Kvalbein, but all authors contributed to the underlying ideas. The simulations were performed by Amund Kvalbein and Audun Fosselie Hansen.

An extended version of this paper, which also contains some of the work in chapter 6, might appear in:

Title: Multiple Routing Configurations for Fast IP Network Recovery
Authors: Amund Kvalbein, Audun Fosselie Hansen, Tarik Čičić, Stein Gjessing and Olav Lysne
Published: Submitted to IEEE Transactions on Networking, 2006.

Chapter 6

Title: Post-Failure Routing Performance with Multiple Routing Configura-

tions
Authors: Amund Kvalbein, Tarik Čičić and Stein Gjessing
Published: To appear at 26th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM). Anchorage, Alaska, May 6-12, 2007.
Author contributions: The underlying idea, the writing, the algorithms and all simulations are by Amund Kvalbein. Tarik Čičić contributed with input to the simulation setup and improvement of the text. The role of Stein Gjessing was mainly related to improving the structure and consistency of the presentation.

Chapter 7

Title: Robust Load Balancing using Multi-Topology Routing
Authors: Amund Kvalbein and Olav Lysne
Published: Not yet published
Author contributions: The underlying idea has appeared through discussions among Amund Kvalbein, Audun Fosselie Hansen, Tarik Čičić, Stein Gjessing and Olav Lysne. The algorithms, writing and simulations presented are by Amund Kvalbein. Olav Lysne contributed in the discussion of the simulation scenarios and results, and through improving the structure and consistency of the presentation.