MPLS Based Recovery Mechanisms

Master Thesis

Johan Martin Olof Petersson

UNIVERSITY OF OSLO
May 2005

Foreword

This thesis is part of my Candidatus Scientiarum studies in communication systems at the department of Informatics, Faculty of Mathematics and Natural Sciences, University of Oslo. These graduate studies are composed of one part of courses and one part of scientific research. This thesis is the result of my scientific research.

I would like to thank Christian Callegari for his help to make a working RSVP-TE module for the network simulator NS-2, and Boning Feng for his guidance during the work with this thesis.

Table of Contents

FOREWORD **3**

TABLE OF CONTENTS **5**

1. INTRODUCTION **7**

1.1 OVERVIEW 7
1.2 THESIS GOALS 8
1.3 STRUCTURE OF THIS THESIS 8

2. TRAFFIC ENGINEERING **9**

3. MPLS TECHNOLOGY OVERVIEW **11**

3.1 THE MPLS DOMAIN 12
3.2 FORWARDING EQUIVALENCE CLASS (FEC) 13
3.3 LABEL SWITCHED PATH (LSP) 13
3.4 MPLS HEADER 15
3.5 MPLS CONTROL PLANE AND FORWARDING PLANE 16
3.6 PATH DETERMINATION 16
3.6.1 Off-Line Path Calculation 16
3.6.2 Constraint-Based Routing 16
3.7 MPLS PATH SETUP 17
3.8 MPLS SIGNALING PROTOCOLS 18
3.8.1 LDP 18
3.8.2 CR-LDP 18
3.8.3 RSVP-TE 19
3.8.4 BGP-4 22
3.9 PATH PRIORITY 22
3.10 PENULTIMATE HOP POPPING 22
3.11 MAKE BEFORE BREAK 23
3.12 LOCAL AND GLOBAL LABEL SPACE 24
3.13 LABEL MERGING 24
3.14 HIERARCHY, TUNNELING. 25
3.15 MPLS AND DIFFSERV. 25
3.16 SUMMERY 27

4. NETWORK RECOVERY **29**

4.1 TYPES OF ERRORS IN A NETWORKS 30
4.2 FAILURE DETECTION 31
4.3 PROTECTION AND RECOVERY AT DIFFERENT LAYERS 33
4.3.1 Protection Switching. 33
4.4 TOPOLOGIES: 36
4.4.1 Mesh Topologies 36
4.4.2 Ring Topologies 37
4.4.3 Hybrid Cycle / Mesh 41
4.5 NETWORK LAYER RECOVERY 42
4.6 COMPARISON OF THE DIFFERENT LAYER MECHANISMS 43

5. MPLS RECOVERY **47**

5.1 FAILURE DETECTION / NOTIFICATION IN MPLS 48
5.1.1 RSVP-TE Hello Extension 48
5.1.2 RSVP-TE Softstate 49
5.1.3 LSP Ping/Traceroute 49
5.1.4 Graceful restart 50
5.1.5 Failure reports from other layers 50
5.1.6 Summary 51
5.2 MPLS RECOVERY MECHANISMS 52

5.2.1 Recovery Path Placement 52
5.2.2 Recovery Path Calculation 55
Rerouting 55
Protection Switching........ 57
5.2.3 Comparison of Rerouting and Protection Switching........ 58
5.2.4 Techniques to setup recovery paths 59
5.2.5 Path Mapping........ 59
5.2.6 Protection Resource Sharing 60
5.3 MPLS RECOVERY MODELS 63
5.3.1 Makam's Model........ 63
5.3.2 Reverse Notification Tree (RNT)........ 64
5.3.3 Reverse Backup (Haskin's model)........ 65
5.3.4 Hundessas Model 66
5.3.5 Fast Reroute........ 67
5.3.6 Fast Reroute one-to-one backup 68
5.3.7 Fast reroute Facility Backup 69
5.3.8 Modifications for shared backup protection. 70

6. SIMULATION AND ANALYSIS 73

6.1 SIMULATION ENVIRONMENTS 73
6.1.1 J-Sim 73
6.1.2 OMNeT++ 73
6.1.3 GLASS........ 74
6.1.4 NS-2 74
6.1.5 Installation of MNS with RSVP-TE for NS-2........ 75
6.1.6 Extending the RSVP-TE patch 76
6.2 A CLOSER LOOK AT THE RSVP-TE HELLO MECHANISM........ 77
6.3 SIMULATIONS OF MPLS RECOVERY 80
6.3.1 Global protection with rerouting (best effort protection) 82
6.3.2 Makam's model (Global recovery with protection switching)........ 83
6.3.3 Haskin's model........ 84
6.3.4 Local rerouting 85
6.3.5 Fast reroute with one-to-one backup 86
6.4 RESULTS........ 88
6.4.1 Packets dropped........ 88
6.4.2 Service disruption 89
6.4.3 Pre-reserved backup resources........ 90

7. CONCLUSION........ 93

7.1 TOPICS FOR FURTHER RESEARCH 97

8. REFERANCES........ 99

APPENDIX A 107

APPENDIX B 111

APPENDIX C 131

1. Introduction

1.1 Overview

Early computer networks carried continuous bit streams over physical links in a technique called circuit switching. This was well suited to transmit voice or real time data from a single sender to a single receiver (unicast communications). In this kind of network a single physical link failure had dramatic consequences, leading to the interruption of all communications that was using the failed link. The Internet today is a datagram packet switched network that fixes this drawback by cutting data into small chunks called packets. These packets are individually routed through the network, so two packets from the same communication are individually handled in the network. Therefore if a link fails, packets can be *rerouted* to avoid the failed link and communications are not interrupted. This feature of a datagram switched network is called *resilient* because it hides network failures from the end users. On the other hand it is more difficult to manage flows of data in a datagram packet switched network than in a circuit switched network because each packet is handled individually.

In the last years there have been an enormous growth in the use of Internet, and new real-time connection-oriented services like streaming technologies and mission-critical transaction-oriented services are in use and new ones are currently emerging. These new services are gaining more and more importance for companies, but they require more from the Internet then previous services.

A lot of research has been done in a field called Quality of Service (QoS), which aims to give priorities to traffic in a network. The idea is that not all traffic in a network needs to be treated in the same way. Some traffic needs less delay then other or more bandwidth (or other constraints). By classifying traffic into different classes, these traffic classes can be treated differently by routers and a kind of guarantee can be given to those different priority classes. In DiffServ which is one of the QoS proposals high priority traffic have priority in buffers in routers and are therefore forwarded before lower priority traffic. In this way real-time traffic can get a guarantee in delay of packet handling in routers. Another QoS scheme is Intserv that allows resources to be reserved along a path in the network. With the use of Intserv a minimum bandwidth can be reserved along a path in a network. These QoS techniques can be useful for traffic that needs some kind of high constraint to be useful at the receiver side, but they can not guarantee anything in the case of a failure in the network.

The IP protocol used today is designed to be robust and is therefore able to re-establish connectivity after almost any failure of network elements. Although connectivity can be re-establish, the time it takes to do so might not be in the limit for what is acceptable for high priority services. For services that can't allow interruptions in the order of a few hundred milliseconds like voice over IP this can not be met by today's Internet. If the network shall be resilient to failures for these kinds of services, there needs to be mechanisms in the network that can re-establish connectivity faster then the current rerouting techniques used in today's internet.

1.2 Thesis goals

In this thesis I will look at different techniques that can be used to reroute traffic faster then the current IP rerouting methods in the case of a failure in a network. I will concentrate on proposals for MPLS but also describe approaches at other layers in the network communications model. The aim of this thesis is to look at the different proposals from IETF that shall make traffic flows in an MPLS domain resilient to failures for real time services that demands that traffic can be rerouted in the network as fast as possible.

1.3 Structure of this thesis

Chapter 2 gives a short introduction to traffic engineering. This is done because traffic engineering is today one of the main reasons to use MPLS. It is therefore necessary that the reader of this thesis is familiar with why traffic engineering is used, and why MPLS is useful for traffic engineering.

Chapter 3 gives an introduction to MPLS, the basic functionality of MPLS will be explained and some of the functionalities in MPLS that can be used for traffic engineering and network recovery will be explained.

Chapter 4 contains an overview of common failures in networks, failure detection mechanisms are explained and some recovery techniques for other layers in the communications model than the MPLS layer will be explained. This is done because it is important to know about how recovery is performed without MPLS, this gives an insight into some strengths of MPLS recovery and a comparison ground for recovery in MPLS.

Chapter 5 explains the basic mechanisms that can be used in MPLS for recovery operations and some of the proposed models for MPLS recovery are explained.

Chapter 6 will give an overview of available simulator environments that can be used for MPLS simulations. The simulations I have done for this thesis are explained and the results from those simulations are analyzed.

Chapter 7 holds the concluding remarks for this thesis and suggestions for further research in the field.

Appendix A gives an example of an algorithm that can be used to find a shared recourse reservation.

Appendix B holds the tcl scripts used for my simulations.

Appendix C shows some of the code additions used to implement new functionality to the NS-2 network simulator.

2. Traffic engineering

MPLS is today mostly used for traffic engineering and I will therefore start by describing what traffic engineering is and why traffic engineering is needed.

In a sense, IP networks manage themselves. A host using the Transmission Control Protocol (TCP) adjusts its sending rate according to the available bandwidth on the path to the receiver. If the network topology should change, routers react to changes and calculate new paths to the destination. This has made the TCP/IP Internet a robust communication network. But robustness does not implicate that the network runs efficiently. The interior gateway protocols used today like OSPF and ISIS compute the shortest way to the destination and routers forward traffic according to the routing tables build from those calculations. This means that traffic from different sources passing through a router with the same destination will be aggregated and sent through the same path. Therefore a link may be congested despite the presence of under-utilized link in the network. And delay sensitive traffic like voice-over-IP calls may travel over a path with high propagation delay because this is the shortest path while a low latency path is available.

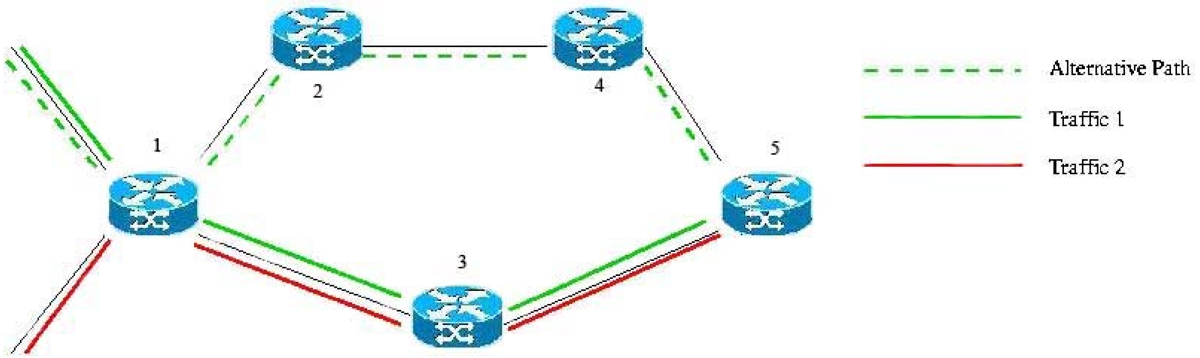


Figure 1 : Traffic Engineering

As illustrated in the above figure the shortest path from router 1 to 5 is the path (1-3-5). All traffic passing through router 1 with destination router 5 (or another router with router 5 in the shortest path) will travel through this shortest path if the shortest path algorithm is used for forwarding in this network. Although there is an alternative path (1-2-4-5) available that could be used to distribute traffic more evenly in the network.

Traffic engineering is the process of controlling how traffic flows through a network to optimize resource utilization and network performance [10].

Traffic engineering is basically concerned with two problems that occur from routing protocols that only use the shortest path as constraint when they construct a routing table.

The shortest paths from different sources overlap at some links, causing congestion on those links.

The traffic from a source to a destination exceeds the capacity of the shortest path, while a longer path between these two routers is under-utilized.

As we will se later MPLS can be used as a traffic engineering tool to direct traffic in a network in a more efficient way then original IP shortest path routing. MPLS can be used to control which paths traffic travels through the network and therefore a more efficient use of the network resources can be achieved. Paths in the network can be reserved for traffic that is sensitive, and links and router that is more secure and not known to fail can be used for this kind of traffic.

There are also other strategies then MPLS that can be used for traffic engineering like IP-over-ATM [37], constraint based routing [38] and others.

3. MPLS Technology Overview

In this chapter I will give an introduction to MPLS and describe some of the properties of MPLS that makes it useful for traffic engineering and resilience.

In conventional IP routing, each router in the network has to make independent routing decisions for each incoming packet. When a packet arrives at a router, the router has to consult its routing table to find the next hop for that packet based on the packets destination address in the packets IP header (longest match prefix lookup). To build routing tables each router runs IP routing protocols like BGP, OSPF or ISIS. When a packet traverses through the network, each router performs the same steps of finding the next hop for the packet. Because packets are forwarded solely on destination address, flows of packets to the same destination are aggregated. In a traffic engineering perspective this kind of routing can be highly sub optimal (though from a routers perspective it can be very desirable). This is because it would be easier to load balance the network if the traffic consisted of many small flows instead of a few large flows. To get reasonably close to an optimum use of resources in a network, additional mechanisms are needed to manage IP flows then conventional IP routing, like the ability to aggregate traffic into appropriate sized flows, and the ability to explicitly route those flows through the network.

The main issue with conventional routing protocols is that they do not take capacity constraints and traffic characteristics into account when routing decisions are made. The outcome is that some segments of a network can become congested while other segments along alternative routes become under utilized.

A common technique used among large ISPs is to use a layer 2 network (ATM or FR) to manage a network. In this approach often called the overlay solution, a complete mesh of virtual circuits connects the IP backbone. This serves to prevent the aggregation that occurs by hop-by-hop routing in an IP backbone with destination based routing. In this approach the flows can be individually routed through the layer 2 topology and traffic engineering can be achieved. But the drawback to this approach is the issue of scalability and that a single link failure can result in dozens of Virtual Circuits going down, forcing the IP routing protocols to reconvert. A solution for this problem can be coordination between the layer 2 network and the layer 3 IP network. This solution is MPLS, a set of procedures for combining the performance, QOS and traffic management of the Layer 2 label swapping paradigm with the scalability and flexibility of Layer 3 routing functionality. The basic of MPLS is to assign short fixed-length labels to packets at the edge of the MPLS domain and then use these pre assigned labels rather then the original packet headers to forward packets on pre-routed paths through the MPLS network.

In MPLS, the route the packet is forwarded through the MPLS domain is assigned only once i.e., when the packet enters the domain. The nodes along the path do not make any routing decisions for the specific packet, they use a label in the packet as an index into a table that specifies the packets next hop. Before a router forwards a packet it changes the label in the packet to a label that is used for forwarding by the next router in the path. This is the main architectural concept underlying MPLS, the separation of the control plane from the data forwarding plane in the switching elements. The control plane is concerned with network level coordination functions, such as routing and signalling to find and setup paths that can be used to move traffic across the network, while the data plane consists of forwarding components that performs simple label switching operations. This enables for explicit routing and differentiated treatment of packets while keeping the core routers simple.

MPLS is short for Multi Protocol Label Switching. The Multi Protocol indicates that MPLS is developed to work independent of what layer 2 and layer 3 protocols that are used in the network. While the Label Switching indicates the forwarding mechanism used by MPLS namely Label Switching.

3.1 The MPLS domain

In [1] the MPLS domain is described as *"a contiguous set of nodes which operate MPLS routing and forwarding"*. This domain is typically managed and controlled by one administration. The MPLS domain concept is therefore similar to the notion of an AS (autonomous system), as the term is used in conventional IP routing i.e. a set of related routers that are usually under one administrative and management control.

The MPLS domain can be divided into **MPLS core** and **MPLS edge**. The core consists of nodes neighbouring only to MPLS capable nodes, while the edge consists of nodes neighbouring both MPLS capable and incapable nodes. The nodes in the MPLS domain are often called LSRs (Label Switch Routers). The nodes in the core are called transit LSRs and the nodes in the MPLS edge are called LERs (Label Edge Routers). If a LER is the first node in the path for a packet travelling through the MPLS domain this node is called the **ingress LER**, if it is the last node in a path it's called the **egress LER**. Note that those terms are applied according to the direction of a flow in the network, one node can therefore be both ingress and egress LER depending on which flow is considered. The terms upstream and downstream routers are also often used to indicate in which order the routers are traversed. If a LSR is upstream from another LSR, traffic is passed through that LSR before the other (downstream). A schematic view of the MPLS domain is illustrated below.

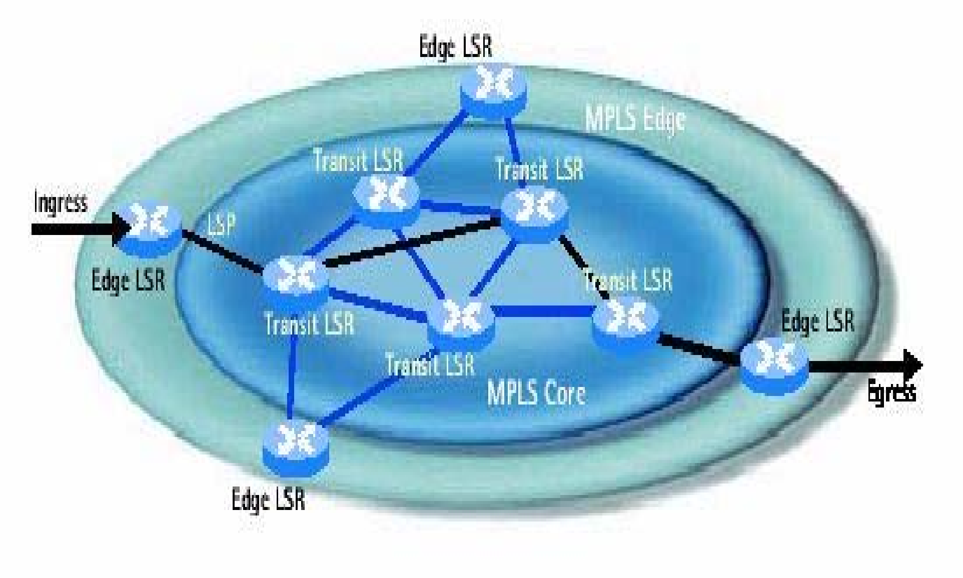


Figure 2 : The MPLS domain

3.2 Forwarding Equivalence Class (FEC)

The Forwarding Equivalence Class (FEC) is defined in RFC 3031 [1] as:

> *"A group of IP packets which are forwarded in the same manner (e.g., over the same path, with the same forwarding treatment)".*

In other words, all IP packets that are forwarded over the same path and treated in the same manner belong to the same FEC. Therefore traffic flows that are aggregated in MPLS are called an FEC. There should be a FEC to assign any unlabeled incoming packet into a group that will become MPLS labelled packets. MPLS FEC membership is not strictly based on SPF (shortest path first) destination address calculations as in IP, but can be determined based on other parameters such as packet source, DSCP (DiffServ code points) and other QOS parameters found in the network, transport and application headers.

The granularity of FECs within a router can vary from very coarse to extremely fine, depending on the level of information used to assign IP packets to a FEC. If for example the classification is based on the five-tuple (source and destination IP addresses, source and destination TCP or UDP ports, and a protocol number). This results in fine-granularity FECs. If the classification is based just on the destination IP address, then the resulting FECs are of medium-granularity. If the FEC classification is based solely on the egress LSR, this creates coarse-granularity FECs.

Because MPLS generally use control-driven label assignment, which label is applied to which incoming network layer packet is predetermined by setting up these FECs ahead of time, so that the FECs can be determined on administrative TE decisions and calculated off-line. This is opposed to the data-driven approach where traffic is aggregated and classified upon detection of a certain flow of data.

In MPLS, the assignment of a particular packet to a particular FEC is done just once, as the packet enters the network by the ingress LSR. This is a major difference to conventional IP routing where the packet header is analyzed at each hop.

3.3 Label Switched Path (LSP)

When an IP packet traverses through a MPLS domain, it follows a predetermined path depending on the FEC to which it was assigned by the ingress LER. The path the packet follows through the MPLS domain is called the Label Switched Path (LSP). LSPs are unidirectional so to build a duplex communication two LSPs are needed.

When various Layer 3 packets are entering the Ingress LSR, they are classified into a FEC as described earlier. Once the packets are classified, they are put into the corresponding LSP for this FEC. Or to say it in a different way: FEC acts as a filter defining which IP packets should be forwarded on particular LSPs. An LSP may carry more than one FEC. For example, several fine-granularity FECs may share the same egress LSR and therefore use the same LSP.

The FEC for a packet is assigned only once at the ingress LSR when the packet enters the MPLS domain. After the packet is assigned to a FEC, a MPLS header is inserted in the packet and the packet is forwarded to the next hop. The transit LSRs at the subsequent downstream hops does not need to analyze other header information in the packet then the MPLS header when they make forwarding decisions. The packet is forwarded solely based on the information in the MPLS header and the interface that the packet arrived on, which is used as an index in table lookups that specify the packet treatment. There are three basic types of operations that can be applied to a packet.

- Push the label stack.
- Swap the top label with a new label.
- Pop the label stack.

This way of decoupling forwarding from routing for the transient LSRs was originally one of the motivations for MPLS. At that time the difference of workload for routers in IP forwarding doing a longest match prefix lookup, and for the transient LSRs in MPLS forwarding packets by label switching, it was faster to do a lookup in the label forwarding table then to do a longest prefix match. But today longest prefix match in IP routers can be done at "wire speed" [4] so this is no longer a problem.

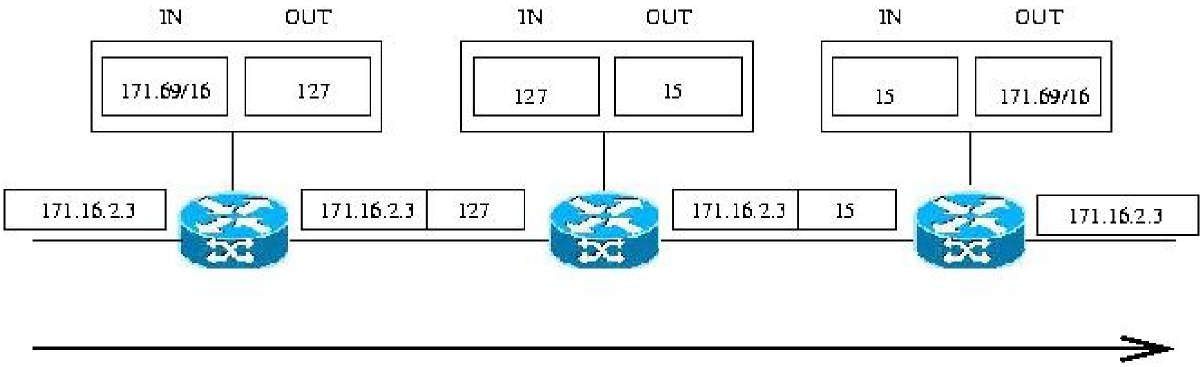


Figure 3: forwarding with labels in MPLS

The above picture illustrates how MPLS routers use the label in the MPLS header to forward traffic in an MPLS domain. The first router in the path is the ingress router in the domain and pushes an MPLS header on the packet. To do this it first has to decide which FEC this packet belongs to, this is done in what is called a FEC-To-Next Hop Label Forwarding Entry (FTN) mapping. The LSR also has a Next Hop Label Forwarding Entry (NHLFE) table that is used to decide how to handle incoming packets. It contains information on what actions to take on the label in the packet (push, pop or swap) and on which interface the packet shall be forwarded. The FTN is used to map an incoming unlabeled packet to a FEC and then map this FEC to a NHLFE. So after the FEC classification the label and the label action to perform are found in the NHLFE table. When the label is inserted and the outgoing interface found, the package is forwarded on the link to the next router.

The next router in the path uses the label in the packet to decide how it shall handle the packet. It swaps the label with a new one and forward it on the interface found from its Incoming Label Map (ILM). The ILM is used to match each incoming label to a NHLFE and is used when forwarding packets that arrive labelled. The last router finds that it is the egress router by consulting its ILM and NHLFE and therefore removes the MPLS header and forwards the packet as an IP packet.

3.4 MPLS Header

As described earlier forwarding in MPLS is done by using the label in the MPLS header. Therefore the MPLS header has to be inserted into packages that are to be routed in the MPLS domain. For data link layer switching technologies like ATM and FR, the MPLS header is inserted in the native label field for that protocol. For ATM the VPI/VCI field is used and for FR the DLCI field is used. In the case where the Layer 2 technology does not support a native label field, the MPLS header must be inserted between the Layer 2 and Layer 3 headers. This MPLS header is 32 bits long and is often called the "shim" header. The MPLS header contains four fields, shortly explained in the Table below.

Label	20 bits	The actual value of the MPLS label assigned to the packet.
Exp	3 bit	Experimental bits, used for differentiated services and providing classes of service.
S	1 bit	Stack field, indicates usage of label stacking.
TTL	8 bit	Time-To-Live, provides the same functionality as TTL of Ipv4 or Hop Limit of Ipv6.

Table 1: The MPLS Header.

One of the strengths of the MPLS architecture is the definition of the label stack. A label stack is the ability to have a sequence of MPLS headers in one packet. This stacking allows for creation of hierarchical LSPs that can be used for services like network management, VPNs and TE. When there is more then one header in a packet the S bit in the innermost label is set to 1, indicating that this is the last header in the stack.

The TTL field in the header works in the same way as the TTL in IPv4 and is copied from the IP header to the MPLS header as the packet enters the MPLS domain.

3.5 MPLS Control Plane and Forwarding Plane

Routers in a MPLS domain have two architectural planes: control plane and forwarding plane [2]. A LSR is able to process both conventionally routed IP packets and MPLS routed packets, so both the control plane and the forwarding plane has functionalities for both IP and MPLS. The control plane is the IP routing protocols and the label distribution protocol used. The forwarding plane is the conventional IP forwarding and the MPLS Forwarding.
The MPLS forwarding plane forwards packets according to the labels attached to packets. The MPLS forwarding can perform different label actions according to the instructions in the NHLFE (next hop label forwarding entry) and in the Incoming Label Map (ILM) table at each node.
If the LSR acts like an ingress router it must also have a FTN (FEC To NHLFE) table to accept an entering unlabeled packet and then determine which entry will be used to label and forward the packet as it enters the MPLS domain. The FTN maps each unlabeled incoming packet to an FEC.

3.6 Path Determination

The two main approaches to determine the desired granularity for FECs and determining the paths for the LSPs are offline path calculation and constraint based routing. There is also the option of data driven path setup. In data driven path setup the label bindings are created by each LSR by reacting to the forwarded data packets; e.g. when a flow of packets that belong to the same FEC are forwarded a LSR can create label bindings for the FEC and distribute those to the upstream neighbor LSR. But data driven binding is not relevant in practice because many of the benefits with MPLS cannot be attained with this approach, for example traffic engineering.

3.6.1 Off-Line Path Calculation

The LSPs and FECs can be determined with an off line tool without the LSRs directly participating in the process. The basic input to the tool is ingress and egress points, physical topology and traffic estimates. Based on the inputs the tool can be used to calculate a set of physical paths for LSPs that optimize the usage of the network resources. Then those routes can be explicitly setup in the MPLS domain. This way of doing path calculations can lead to optimal resource usage, predictable routing and stable network configurations.

3.6.2 Constraint-Based Routing

With constraint based routing, network parameters (constraints) are used to determine the best route a set of packets should take. Each LSR determines an explicit route for each traffic trunk (aggregation of traffic flows) originating from that LSR based on bandwidth and cost of the links and other topology state information. The LSP created is the route that satisfies the requirements of the traffic and the constraints that are set.

Calculating a path that satisfies these constraints requires that the information about whether the constraints can be met is available for each link, and this information must be distributed to all the nodes that perform path calculation. This means that the relevant link properties have to be advertised throughout the network. This is achieved by adding TE-specific extensions to the link-state protocols ISIS and OSPF that allow them to advertise not just the state (up/down) of the links, but also the link's administrative attributes and the bandwidth that is available. In this way, each node has knowledge of the current properties of all the links in the network. Once this information is available, a modified version of the shortest-path-first (SPF) algorithm, called constrained SPF (CSPF), can be used by the ingress node to calculate a path with the given constraints. CSPF operates in the same way as SPF, except it first prunes from the topology all links that do not satisfy the constraints.

Explicit Routing is a subset of the more general constraint-based routing where the constraint is an explicit route (ER). An explicit route is a LSP that is set up to send data on a specific path through the MPLS domain, this is as described earlier useful for traffic engineering.

A constraint-based routing framework can greatly reduce the level of manual configuration and intervention required to actualize Traffic Engineering policies [10].
In practice, the traffic engineer will specify the endpoints of a traffic trunk and assign a set of attributes to the trunk about the performance expectations and behavioural characteristics of the traffic trunk. The routing framework then finds a path that satisfies these expectations. The traffic engineer can then use explicit configured routes to perform fine-grained optimization.

3.7 MPLS path setup

There are two main categories of how to set up a LSP. Static LSP or Signaled LSP. Static LSP is a LSP that is manually configured via CLI or SNMP. Visiting each LSR and using network management to set the label and interface typically create this kind of LSP. This approach is roughly equivalent to using static routes in an IP network. As the networks change, get larger and more complex with increased dynamics, the static approach becomes increasingly complex and resource intensive to operate, if it can be maintained at all. Another more dynamic way to setup LSP is to use a signaling protocol instead. Then a signalled LSP is created.

Dynamic signaling protocols have been designed to allow single routers to request the establishment and label binding to FEC for an end-to-end path. The router that needs to setup an LSP simply determines the best path through the network according to the local constraints and requests the routers in the path to establish a LSP and distribute the label binding to FEC. Configuring a new LSP, over a domain that is MPLS and signaling enabled, does not require anything beyond the configuration in the instantiating router.

As described earlier explicit routes can be setup in MPLS. An explicit route can be set up either loose or strict. If a LSP is set up strict all the LSRs in the path is predetermined before the path is set up, and only those routers can be part of the resulting LSP. If a path is set up loose then only a subset of the LSRs are predetermined and the rest is found using path calculations from the underlying routing protocol. This means that a LSP can be setup with some nodes that must be included in the LSP and other intermediate nodes that can be determined by the routing protocol.

3.8 MPLS Signaling Protocols

Signaling is a way in which routers exchange relevant information. In an MPLS network, the type of information exchanged between routers depends on the signaling protocol being used. At a base level, labels must be distributed to all MPLS enabled routers that are expected to forward data for a specific FEC and LSPs created. The MPLS architecture does not assume any single signaling protocol [1] and so far four methods have been specified for label distribution.

- Label Distribution Protocol (LDP)
- Resource Reservation Protocol extension for MPLS (RSVP-TE)
- Constrained Routing with LDP (CR-LDP)
- Distributing labels with BGP-4

3.8.1 LDP

LDP [3] is designed by a working group at IETF from the ground up for the explicit purpose of distributing MPLS labels, thus setting up LSPs in the MPLS domain. LDP works closely with IGP routing protocols and is thus often called "hop-by-hop" forwarding. It always selects the same physical path that conventional IP routing would select. Thus LDP does not support TE. The motivation behind setting up an LSP that follows the same path as conventional IP instead of just using conventional IP routing was originally to speed up the forwarding in routers. In conventional IP routing the next hop for each packet is found by a longest match prefix lookup on the IP header in the routing table. These lookup could in some cases where the routing tables were large be time consuming and it was thought that data forwarding with label switching instead of IP lookups would speed up data forwarding. However, the forwarding speed of IP packets is not an issue anymore [4] when IP header analysis is done at "wire speed".

Because of the resent development in routing technology, LDP is not much used for label distribution today. There is however an extension to the original LDP protocol that brings new functionality for the LDP protocol called CR-LDP.

3.8.2 CR-LDP

CR-LDP [5] is an extension of LDP to support constraint based routed LSPs. The term constraint implies that in a network and for each set of nodes there exists a set of constraint that must be satisfied for the link or links between two nodes to be chosen for an LSP. An example of a constraint is to find a path that needs a specific amount of bandwidth. Another example of a constraint is a path that uses links that is considered to be secure.

LSRs that use CR-LDP to exchange label and FEC mapping information are called LDP peers, they exchange this information by forming a LDP session. There are four categories of LDP messages:

Discovery messages announce and maintain the presence of an LSR in an MPLS domain. This message is periodically sent as a Hello message through a UDP port with the multicast address of all routers on this subnet.

Session message is sent to establish, maintain and delete sessions between LDP peers.

Advertisement messages create, change and delete label mappings for FECs.

Notification Messages provides status, diagnostic and error information.

The last three message types are transported over TCP. CR-LDP makes hard state reservations which means that reserved resources has to be removed explicitly.

In [51] the MPLS working group at IETF stated that they would not continue to develop the CR-LDP standard. This decision was taken after an implementation survey was published in 2002 [52]. In this survey 21 implementers of 22 had implemented RSVP-TE and only 3 had implemented CR-LDP.

3.8.3 RSVP-TE

RSVP-TE [6] is an extension of RSVP [7] that utilizes the RSVP mechanisms to establish LSPs, distribute labels and perform other label-related duties that satisfies the requirements for traffic engineering. RSVP-TE supports both strict and loose routed LSPs that do not have to follow conventional IP routing, giving support also for traffic engineering [8]. RSVP-TE is soft state protocol, which means that when a path has been setup by RSVP-TE it has to be continually updated to keep the recourses reserved. RSVP-TE is a receiver-oriented protocol, which means that requests for reservation are made from the receiver end of the path. When RSVP-TE is used for LSP setup the ingress router starts by sending a PATH message on the path were a LSP shall be setup. Each transit router on that path has to check if it has the possibility to set up the requested LSP. If the requested LSP is rejected an error message is returned upstream until it reaches the ingress router. Otherwise the path message is sent to the next transit router in the path until it reaches the egress router. Then the egress router sends a RESV message back through the path that the PATH message travelled. In the RESV message downstream routers includes the label that they want the adjacent upstream router to use for the LSP that’s being setup. No reservations are made in the routers until the RESV message is returned.

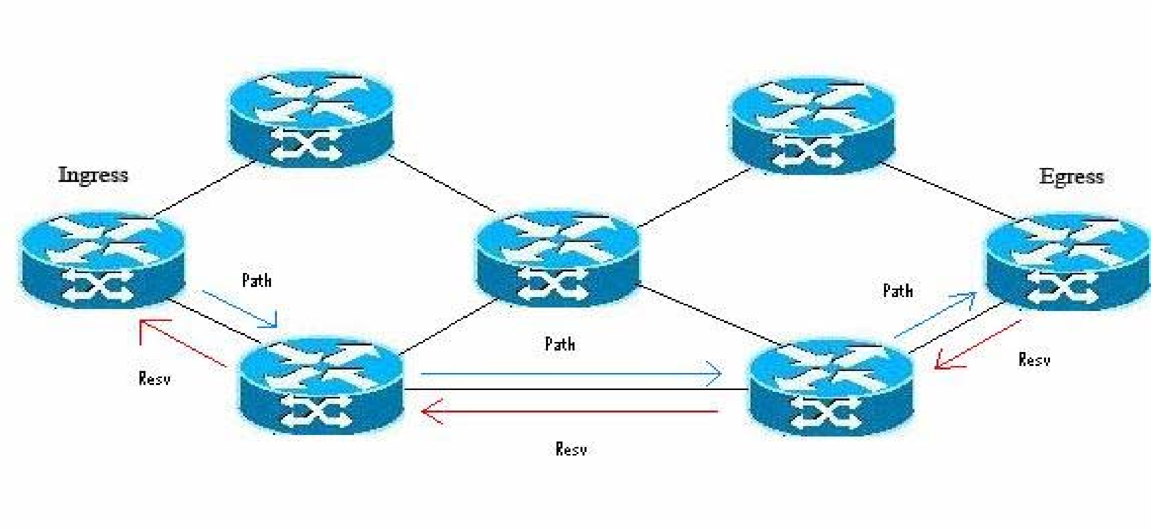


Figure 4: RSVP-TE Path Setup

To use traffic engineering with RSVP there have been some new objects defined for RSVP which resulted in RSVP-TE. To be able to define an explicit route to setup, an ERO-object (explicit route object) had to be included in the PATH message. If the ERO is included in the PATH message then the route follows the defined path in the ERO-object independent of the shortest path. The path can be either strictly or loosely defined. The ERO object is made up of tlv (type length value) objects and each LSR that is part of an explicitly routed path is coded into one tlv object.
A Record-Route-Object (RRO) is included in both the PATH and RESV messages. In the PATH message the RRO is used to record each LSR and in which order each LSR is visited. This list is then sent in the RESV message so that the each upstream router up to the ingress LSR receives this list.

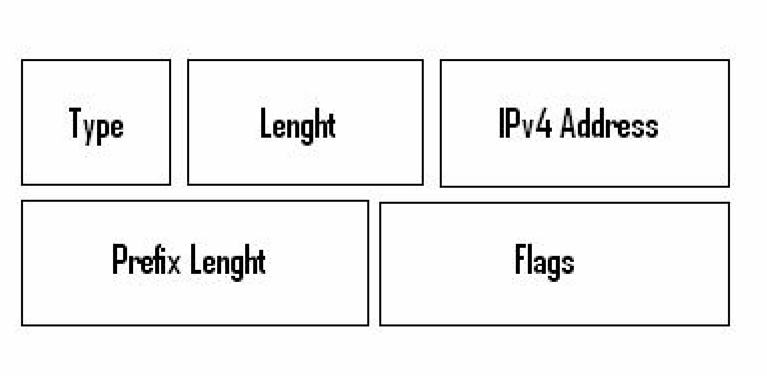
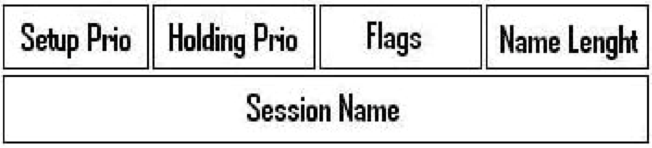


Figure 5: The Record-Route-Object

A SESSION-ATTRIBUTE-object is included in the PATH-message to identify sessions and to describe characteristics like the priority for the LSP.



Setup Prio	Holding Prio	Flags	Name Lenght
Session Name			

Figure 6: The Session-Attribute-Object

The Setup Priority field defines the priority to the LSP. The field is 4 bits so there are 8 possible priorities from 0 to 7, with 0 as the highest priority. Priority is used in pre-emption, where a new path can take down an already established path if that path holding priority is lower then the new paths setup priority.

The Flags field is 8 bits. Of these 3 are defined:

1. **Local protection desired:** This flag permits transit routers to use a local repair mechanism, which may result in violation of the explicit route object. This is used in recovery techniques explained in the chapter about recovery in MPLS.

2. **Label recording desired:** This flag indicates that label information should be included when doing a route record, this is useful in tunnelling that will be explained later in this chapter.

3: **SE Style desired:** This flag indicates that the tunnel ingress node may choose to reroute this tunnel without tearing it down. A tunnel egress node shall use the SE Style when responding with a RESV message. This is useful for make-before-break that will be explained later in this chapter.

Reservation Styles:

Each LSP can be reserved with a specific reservation style. There are three types of reservation styles defined for RSVP-TE [6].

Fixed Filter (FF): In fixed filter a distinct reservation is made for traffic from each sender. This reservation can not be shared by other senders.
Shared Explicit (SE): Allows a receiver to explicitly specify the senders to be included in a reservation. There is a single reservation on a link for all the senders listed. Because each sender is explicitly listed in the RESV message, different labels may be assigned to different senders, thereby creating separate LSPs.
Wildcard Filter (WF): With the Wildcard Filter (WF) reservation style, a single shared reservation is used for all senders to a session. The total reservation on a link remains the same regardless of the number of senders. A single multipoint-to-point label switched path is created for all senders to the session. On links that senders to the session share, a single label value is allocated to the session. If there is only one sender, the LSP looks like a normal point-to-point connection. When multiple senders are present, a multipoint-to-point LSP (a reversed tree) is created.

3.8.4 BGP-4

The Border Gateway Protocol (BGP) can also be used for label distribution. BGP is a routing protocol used between different autonomous systems to exchange routing information.
The update messages in BGP-4 [9] that are used to distribute BGP routes can additionally carry the appropriate MPLS labels that are mapped to the same BGP route. The label mapping information for a particular route is piggybacked in the same BGP update message that is used to distribute the route itself.

3.9 Path Priority

MPLS introduces the concept of LSP priorities. The purpose of priorities is to mark some LSPs as more important than others and allow them to use resources from less important LSPs (pre-empt the less important LSPs). This makes it possible for an important LSP to be established along the most optimal path for this LSP, regardless of existing reservations, if those reservations have a lower priority then this LSP. When a LSPs need to reroute, important LSPs have a better chance of finding an alternate path the lower priority LSPs. Best effort traffic that does not need the same treatment in the network, can be mapped to low priority LSPs and higher priority LSP can pre-empt those low priority LSPs if it becomes necessary.

There are eight priority levels, with 0 as the best and 7 as the worst value. An LSP has two priorities associated with it: a setup priority and a hold priority. The setup priority controls access to the resources at the time of LSP establishment, and the hold priority controls access to the resources for an LSP that is already established. At LSP setup time, if insufficient resources are available, the setup priority of the new LSP is compared to the hold priority of the LSPs using the resources, in order to determine if the new LSP can pre-empt any of the existing LSPs and take over their resources.

3.10 Penultimate Hop Popping

With the simple MPLS forwarding there is a clear drawback, the egress LSR must perform a double lookup. First it makes a table lookup in the MPLS forwarding table just to realize that the label must be popped. Then it makes a conventional Layer 3 lookup before the packet can be forwarded to the next hop. This double lookup can be avoided with a technique called penultimate hop popping (PHP). In PHP the egress LSR requests its upstream neighbor to pop the MPLS label before forwarding the packet. Then the egress LSR receives an unlabeled IP packet and can directly perform Layer 3 lookup. In this way the double lookup can be avoided.

3.11 Make Before Break

One of the requirements for Traffic Engineering is the capability to reroute an established LSP under a number of conditions, based on administrative policy. For example, in some contexts, an administrative policy may dictate that a given LSP is to be rerouted when a more "optimal" route becomes available. Another important context when a LSP reroute is usually required is upon failure of a resource along the path. Under some policies as in the case of failure, it may be necessary to return the LSP to its original path when the failed resource becomes available again.

In general, it is highly desirable not to disrupt traffic, or impact network operations while LSP rerouting is in progress. This means that is necessary to establish a new LSP and transfer traffic from the old LSP onto the new LSP before tearing down the old one. This concept is called "make-before-break". A problem can arise because the old and new LSP might compete with each other for resources on network segments that they have in common. Depending on availability of resources, this competition can cause admission control to prevent the new LSP from being established.

To support make-before-break in a smooth fashion, it is necessary that for the links common for the old and the new LSP, the resources used by the old LSP should not be released before traffic is transferred to the new LSP. As the reservations made on the common links is for the same LSP, these reservations should not be counted twice because this might cause admission control to reject the new LSP.

Another example of when make-before-break is needed is if an administrator wants to increase the bandwidth of a LSP. The new reservation needs the full amount of the links bandwidth, but the actual allocation needed is only the difference between the new and old bandwidth. The following example is used to make the concept of make-before-break clearer.

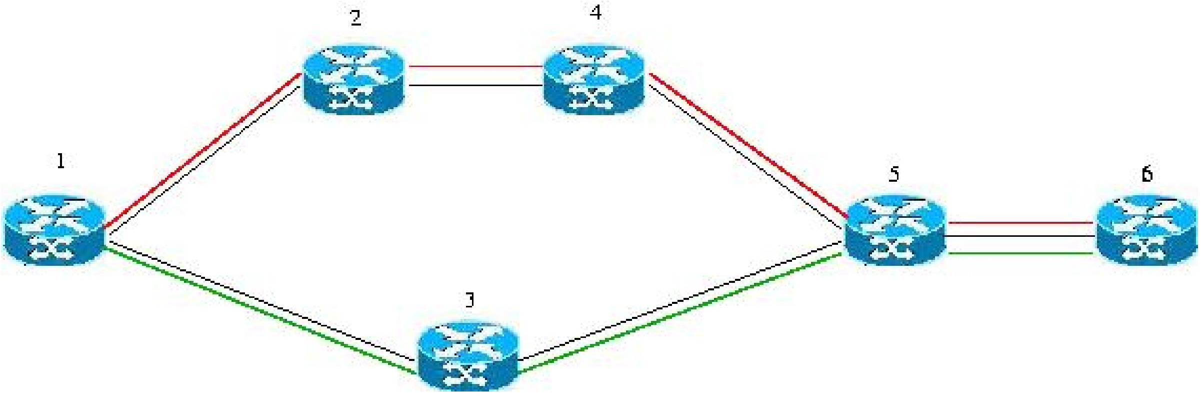


Figure 7 : Make before break

In the network in the picture above there is a LSP from (1,2,4,5,6). Suppose that this LSP has reserved all available bandwidth along the links in this path. If the LSP is being moved to the shorter path (1,3,5,6), then when the new LSP tries to make a reservation on the link between (5,6) this reservation is denied because the LSP that is being rerouted has already reserved the bandwidth. Instead of first tearing down the LSP and then setup the new one, if make-before-break is used, the reservation along the link can be shared for a short period because it is used by the same LSP. When traffic is switched over to the new LSP the old LSP can be torn down without as much interruption in the traffic flow.

3.12 Local and Global Label Space

Labels can be assigned between LSRs by one of two methods. In the first method called *per interface label space*. Labels are associated with a specific incoming interface on a LSR. If the LSR uses an interface value to keep track of the labels on each interface, a label value can be reused at each interface. The second method is *global label space*, here incoming labels are shared across all interfaces attached to the node. This means that a LSR must allocate the label space across all interfaces, so that the same label value can only be used once per router. The forwarding decision is then made only on the label value and not on the incoming interface and label as for local label space.

3.13 Label Merging

If multiple LSPs arriving at a LSR have different incoming labels but are to be forwarded the same path towards the egress router, then these LSPs may not need to be treated as separate LSPs for the rest of the path. If the FECs carried by these LSPs can be aggregated, this can be seen as an aggregation of traffic. Then those LSPs can be merged together and switched using a common label. This is known as label merging or aggregation of flows.

Aggregation can reduce the number of labels needed to handle a particular set of packets and can also reduce the amount of label distribution traffic needed.
An LSR is capable of label merging if it can receive two packets from different incoming interfaces, with different labels, and send both packets out the same outgoing interface with the same label without the use of the label stack. Once the packets are sent, the information that they arrived from different interfaces with different labels is lost and they are treated as one flow.

3.14 Hierarchy, Tunneling.

The possibility to use stacking of labels in the MPLS header can be used to setup tunnels in an MPLS domain. Consider the LSP through R1, R2, R3 and R4 in the network under. Where R1 is the ingress LSR and receives an unlabeled packet P, and pushes on its label stack the label to cause it to follow this LSP to the egress R4.
However, let us further suppose that R2 and R3 are not directly connected, but are virtually neighbours by a LSP tunnel set up between the two LSRs. This LSP goes through R2, R21, R23 and R3 and is used for all traffic from R2 to R3. Then the actual sequence of LSRs traversed by P is R1, R2, R21, R23, R3 and R4. This concept is called tunneling.

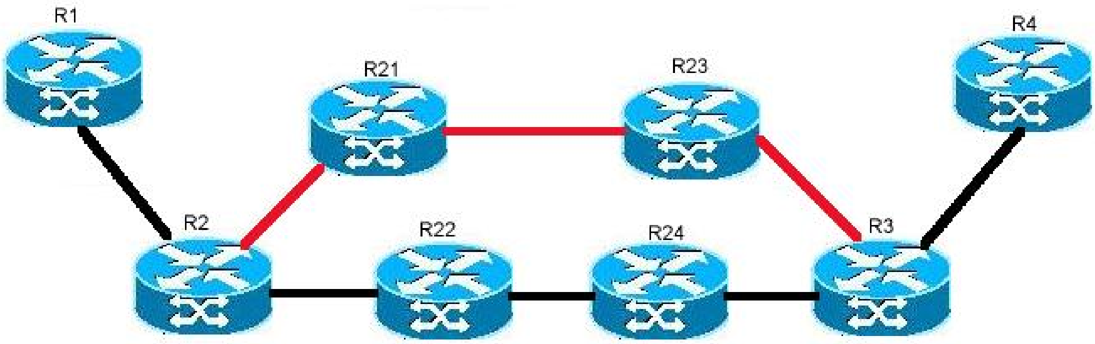


Figure 8: Tunnelling in MPLS

When P travels from R1 to R2, it will have a label stack of depth 1. R2, forwarding on the label in P, determines that P must enter the tunnel set up as the LSP from R2 to R3. R2 first replaces the incoming label with a label that is meaningful to R3. Then it pushes a new label on Ps label stack. This level 2 label has a value which is meaningful to R21. Forwarding is done on the level 2 label by R21, and R23. R23 which is the penultimate hop in the R2-R3 tunnel pops the label stack before forwarding the packet to R3. When R3 receives packet P, P has only a level 1 label. Since R3 is the penultimate hop in P's level 1 LSP, it pops the label stack, and R4 receives P unlabeled.

3.15 MPLS and Diffserv.

The Internet is currently requiring a means for providing different users with different service levels. Traffic with high requirements in for example delay, jitter and bandwidth has to be treated with a certain priority, while the traditional best effort services are still available. Diffserv [11] enables network traffic to be classified into different priority levels and then applies different scheduling and queuing mechanisms at the nodes according to the priority level. The TOS field in the IP header is used to mark a packet and is then used as an indicator of how the packet is forwarded.

Diffserv and MPLS have some things in common [12]. Both approaches push the complexity to the edge of the network. The egress router performs classification of flows and decides how the traffic flow will be treated in the network. In Diffserv the TOS field id is used to label a packet and this field determines the scheduling mechanisms and queuing management used in the transit routers. In MPLS the label applied by the egress router determines the path the packet travels.

It is desirable to combine features from both MPLS and Diffserv to achieve good QoS in a network. If MPLS and Diffserv are combined it is possible to specify the path the packet take and the behaviour in queues at the transit routers. The result is the ability to give strict Quality of Service (QoS) guarantees while optimizing use of network resources.

There are two ways to use Diffserv in MPLS [13].

In **E-LSP** (EXP–Inferred-PSC LSP) the TOS field in the IP packet is mapped into the 3 EXP bits in the MPLS shim header. In this way labels are used to make forwarding decisions and the EXP field is used to determine how to treat the packet in the routers buffer scheduling.

In **L-LSP** (Label-Only-Inferred-PSC LSP) a separate LSP can be established for the flow of traffic. In this case, the LSR can decide the path as well as treatment of the packet from the label of the packet. The EXP field encodes the drop precedence of the packets.

3.16 Summery

This chapter have described the basics of MPLS and explained some of the properties of MPLS that can be used for traffic engineering. The main property of MPLS that makes it useful for traffic engineering is the possibility to compute a path from source to destination that is subject to a set of constraints, and forward traffic along this path. Forwarding traffic along such a path is not possible with best effort IP routing, since the IP forwarding decision is made independently at each hop, and is based solely on the packet’s IP destination address. MPLS can easily achieve to forward traffic along an arbitrary path. The explicit routing capabilities of MPLS allow the creator of the LSP to do the path computation, establish MPLS forwarding state along the path, and map packets into that LSP. Once a packet is mapped onto an LSP, forwarding decisions are based on the label, and none of the intermediate hops makes any independent forwarding decisions based on the packet’s IP destination. This property of MPLS can be used to achieve performance objectives such as optimization of network resources and placement of traffic on particular links.

MPLS works by classifying traffic flows into forwarding equivalence classes (FECs) and then map these FECs onto pre-setup paths (LSPs) in the network. To setup a LSP a signaling protocol can be used. The RSVP-TE protocol has become the standard way to setup paths and distribute information in a MPLS domain. Forwarding in an MPLS domain is done by label switching. This means that a label inserted in the packet is used to determine the next hop for a packet.

In today’s market, service providers seek to offer new services such as voice and guaranteed bandwidth for business-critical applications. These services are often expressed as QoS classes and defined in Service Level Agreements (SLA). The SLAs define the service quality experienced by traffic transiting the network and are expressed in terms of latency, jitter, bandwidth guarantees and resilience.

To fulfil an SLA, DiffServ can be used in the network. This is done by assigning applications to different classes of service and marking the traffic appropriately. However, to receive strict scheduling guarantees, it is not enough to mark traffic appropriately as done in Diffserv. If the traffic follows a path with inadequate resources to meet performance characteristics such as jitter or latency requirements, the SLAs cannot be met. Service providers can solve this problem by using over provisioning to avoid congestion. But this solution of “throwing bandwidth at the problem” is wasteful with regard to resource utilization.

If MPLS is used it sets up LSPs along links with available resources, this ensures that bandwidth is always available for a particular flow to avoid congestion. Because LSPs are established only where resources are available, over provisioning is not necessary. If MPLS is used to combine DiffServ based classification with Traffic Engineering this can lead to better QoS in packet backbones. This can be very useful for services that demand a guarantee of how the network behaves. But to fully provide QoS in a network there also needs to be a guarantee for what happens with traffic in the case when congestion is caused by link and/or node failures.

4. Network recovery

In this chapter I will look at how recovery can be performed in a network. Common network failures and ways to detect failures will be described, and some current solutions for how network recovery can be performed by other layers then MPLS will be presented.

If a failure occurs in a network, recovery is achieved by moving traffic from the failed part of the network to another portion of the network. It is important that this recovery operation can be performed as fast as possible to prevent too many packets from getting dropped at the failure point. If this is achieved fast the failure can be unnoticeable (resilient) or minimal for end-users.

Recovery techniques can be used in both Circuit Switched and Packet switched networks. When a link or node in a network fails, traffic that was using the failed component must change the path used to reach the destination. This is done by the adjacent routers to the failure that updates their forwarding tables to forward packets on a different path that avoids the failing component. The path that the traffic was using before the failure is called the primary path or the working path and the new path is called the backup path. Often recovery techniques consist of four steps.

First, the network must be able to *detect* the failure. Second, nodes that detect the failure must *notify* certain nodes in the network of the failure. Which nodes that are notified of the failure depend on which recovery technique that is used. Third, a backup path must be computed. Forth, instead of sending traffic on the primary path a node called Path Switching Node must send traffic on the backup path instead. This step is called *switchover* and completes the repair of the network after a failure.

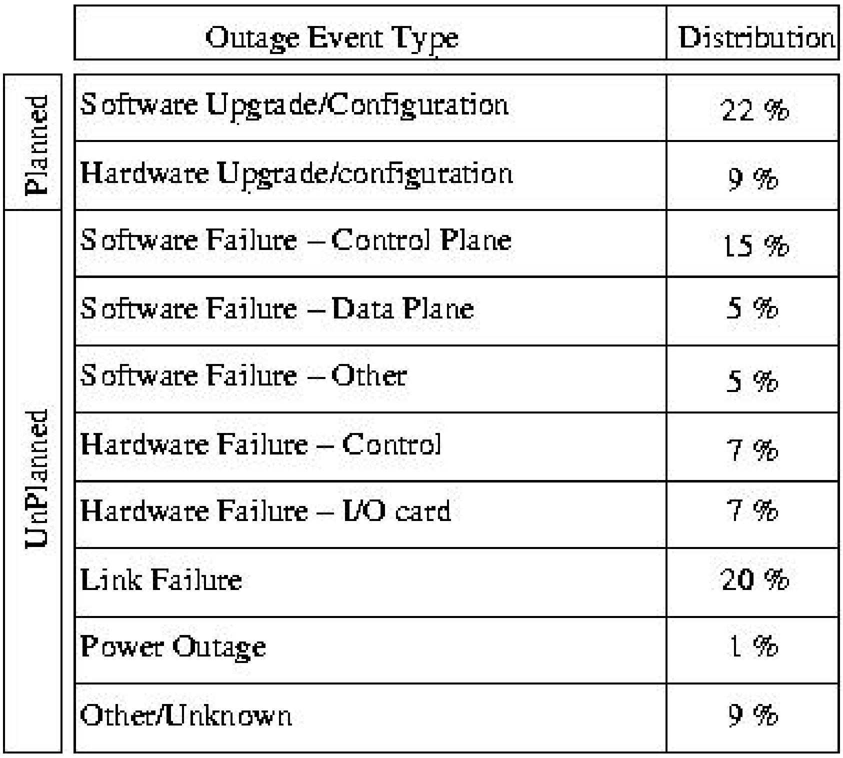
If we consider an unicast communication. When a link fails on the path between the sender and the receiver, users experience service disruption until these four steps are completed. The length of the service disruption is the time between the last bits was sent before the failure occurred is received, and the instance when the first bit of data that uses the backup path arrives at the receiver. The total time of service disruption is:

Service Disruption = Time to detect failure + Time to notify + Time to compute backup + Time to switchover.

4.1 Types of errors in a networks

Any of the resources within a network might fail. The traditional error in a network is a link failure caused by a link getting cut or unplugged by mistake. According to [53], long distance carriers experience between 1.2 and 1.9 cable cuts per 1000 km of cable per year, while local exchange carriers encounter about 3 cable cuts per 1000 km cable per year. And according to [54], cable cuts resulted in approx. 25% of telephone downtime during 1992-1994. So the human factor can very often be the cause of failure. Failures may also arise due to the aging process of equipment or its components or other hardware or software failures in router equipment.

In [49] and [50] a study looked at events that caused outage in a network. The following table shows the distribution of outage events in that study.



	Outage Event Type	Distribution
Planned	Software Upgrade/Configuration	22 %
	Hardware Upgrade/configuration	9 %
UnPlanned	Software Failure – Control Plane	15 %
	Software Failure – Data Plane	5 %
	Software Failure – Other	5 %
	Hardware Failure – Control	7 %
	Hardware Failure – I/O card	7 %
	Link Failure	20 %
	Power Outage	1 %
	Other/Unknown	9 %

Table 2: Failure types

Planned events can be upgrades or configuration of the router OS. This includes patches, new software releases, routing/policy changes etc. It can also be replacement or reconfiguration of hardware. These outages are planned in time so they do not need to result in any downtime for traffic, as traffic can be moved to another segment of the network before the outage occurs.

The unplanned events are the failures that do affect network traffic as they are not planned and therefore have to be reacted upon when they occur.

Software failures in the control plane are failures in any of the routing control plane protocols like the IP routing protocol. Data plane software failures are failures in the forwarding software. Hardware failures are categorized as failures that happen in any control or system component of a router, as router processor, switch fabric, fans etc, and in failure of a particular line card.

Link failures are failures to links such as link cuts and failure in transmission equipment.

No device can be more reliable than its power supply, it has therefore been common to install extra power supplies to eliminate the risk of power outage. And as can bee seen in the table, power outage is only 1% of the observed causes for outage in a network.

As we can se from this table, link failures and control plane failures are the most common kind of failures.

4.2 Failure Detection

If a failure occurs in a network there must be a way to detect that the failure has occurred so that the recovery operation can start. Failure detection depends on the type of failure and may be done by the failing node, at a node adjacent to the failure or at a configured point of repair in the network. Failure detection can involve multiple layer mechanisms and in this section an overview of failure detection techniques will be presented [31].

Hardware manager: Some devices have software and hardware that monitors the state of the controller cards and line cards and the software running on them. If a failure occurs in the device then it self can be aware that it is not working correctly.

Loss of electric connectivity: Failure of an electrical link (for example Ethernet cables) is detected by the line card and reported to the device driver. The failure is detected when the line card detects that there is no electrical connectivity on the link. Failure on an electrical link is detected at both the upstream and downstream ends of a link because electrical links are bidirectional.

Loss of light (LOL): On optical links a failure like a fiber cut is detected at the downstream node of the fault through loss of light. The upstream node is unaware of the failure on the link and can be informed of the failure through a link management protocol.

Link management protocols: Link Management protocols (LMP) [29] are used by a node in a network to monitor individual connections. The end device in a network sends status enquiry message to the network node every polling interval (heartbeat). The node responds with a status message that verifies the link integrity. If the node doesn't respond the downstream node can communicate upstream until the fault is isolated, and then report this to a local management component.

Signaling hellos and keep alive: Many signaling protocols include "hello message" processing where adjacent nodes poll each other periodically to check if a link between them is active. Signaling protocols can also use “keep alive” messages, as example RSVP uses this method to keep a path reserved in a network (soft state). Such soft state mechanisms are implemented by the use of timers and these timers are often set with high tolerance to limit the signaling traffic.

Signaling error notification: If a signaling protocol fails to set-up a path, upstream nodes will be notified by an error message sent by the node that failed to set up the path. This failure message can include crankback information whereby set-up failure information of a path set-up is returned to allow new set-up attempts to be made, avoiding the blocked resources [30].

IGP hellos: Interior Gateway Protocols (IGP) used to setup routing tables run “hello message” exchanges to check if a link is active. If a node does not receive a pre-set number of hello messages from its adjacent neighbors then a failure is detected.

IGP topology updates: An IGP protocol sends periodically topology update messages. If the topology has changed or a failure has occurred this will be detected.

In the case of hardware managers and loss of electronic connectivity, the node that detects the failure is a possible node to perform the network recovery operation. This failure detection is fast because it is done in hardware. If the node that detects the failure has a possibility to make a switchover then recovery can be achieved fast because the detection step is minimal and the notification step can be skipped. Some hardware component failures within a router can also be internally survivable. This can be realized for example if a router has a primary and a backup line card both connected to the same link using a link splitter. If the primary card fails then the backup card can take over and the router can recover transparent to the network. Redundant hardware is therefore the fastest way to have recovery in a network, but it is also expensive and can only protect for a failure on that component that has a backup component.

If traffic is going to be switched over to a backup path, then the node upstream of the failure has to be aware of the failure. Therefore in the event of loss of light the node upstream has to be informed that a failure has occurred. Failure detection in optical networks is therefore usually assisted by a layer 2 protocol that can perform node notification upstream. This requires bi-directional communication between the nodes. Such a solution exists for example in SONET networks or a LMP can be used. In these cases detection has to be assisted by notification, the recovery time is therefore larger then if the node that first detects the failure could perform the next recovery step itself.

Signaling protocols can also be used to detect failures in a network. Because these protocols operate at a higher layer then detection mechanisms like LOL and hardware managers the failure detection with these mechanisms takes more time. This is also the case with failure detection in routing protocols. This will be described later in this chapter.

Lower layer fault detection such as hardware managers, loss of electronic connectivity or loss-of-light (LOL) provide the possibility of fast failure detection and notification, however they cover only lower layer failures. On the other hand, detection and notification of failures through control and management plane can cover many more types of failures such as node failures. However the detection time would typically be much slower.

4.3 Protection and recovery at different layers

Very often the physical layer can have built-in inherent protection mechanism in terms of used technology or topology involved. The great advantage is that in many cases no signaling is required, which means that it is sufficient that the nodes next to the failure are aware that the failure has occurred [22]. Thus, the speed of the whole recovery operation is considerably improved. Within this section recovery mechanism present at the lowest layers are explained.

4.3.1 Protection Switching.

In the case of protection switching, an alternative connection is pre-established and persevered (pre-provisioned) [23]. Protection switching can be done in two different styles, dedicated protection or shared protection.

Dedicated protection:

In dedicated protection the resources used for protection is dedicated for backup traffic. There are two types of dedicated protection called 1+1 and 1:N protection switching.

In 1+1 protection, traffic is simultaneously transported on two different links from the source to the target. The receiver compares the two signals and chooses the better one as the working path and drops traffic received from the other path. When a failure occurs on the working path, the target switch traffic and starts to use traffic from the backup path instead. This full redundancy is very expensive, the recovery path is used for backup only and no other traffic is allowed on this path.

In 1:1 (one for one) protection switching the traffic is sent on the backup link only after a failure is detected on the working path. If a failure occurs the source starts sending traffic over the backup link instead. In some cases the target node has to inform the source about the failure before the source switch traffic over to the protection path.

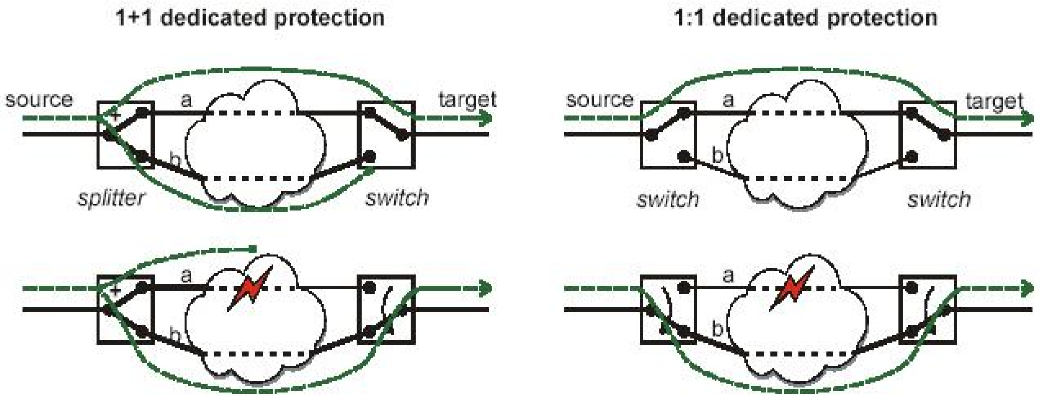


Figure 9 : Protection Switching

In 1: N protection switching which is the general term for 1:1 the backup resources are not dedicated for the recovery of a specific connection, but can be shared by N connections for different failure scenarios. This means that N working paths can be protected by a single protection path thereby a single link can provide protection for anyone of the N working paths. In most implementations the limit for N is set to 14. Because of this sharing of backup resources, 1:N protection is more resource efficient than 1:1 and 1+1 protection. This kind of protection also requires a signaling mechanism for the activation of the backup connection. This scheme can be further generalized to M:N where N working connections is protected by M backup connections.

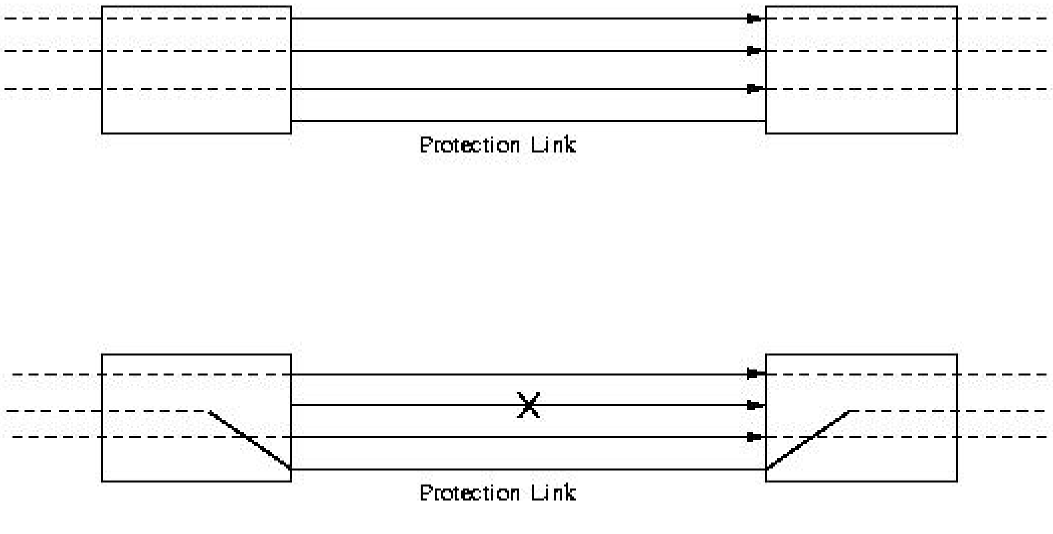


Figure 10 : 1:N protection switching

Shared Protection

As we saw in dedicated protection, different protection paths do not share common resources, which may be a physical transmission line or a WDM wavelength. The failure and activation of one backup path does not affect any other backup path. In shared protection, multiple protection paths may use the same resources. When a backup path is activated, other backup paths that share resources with it will have to be rerouted.

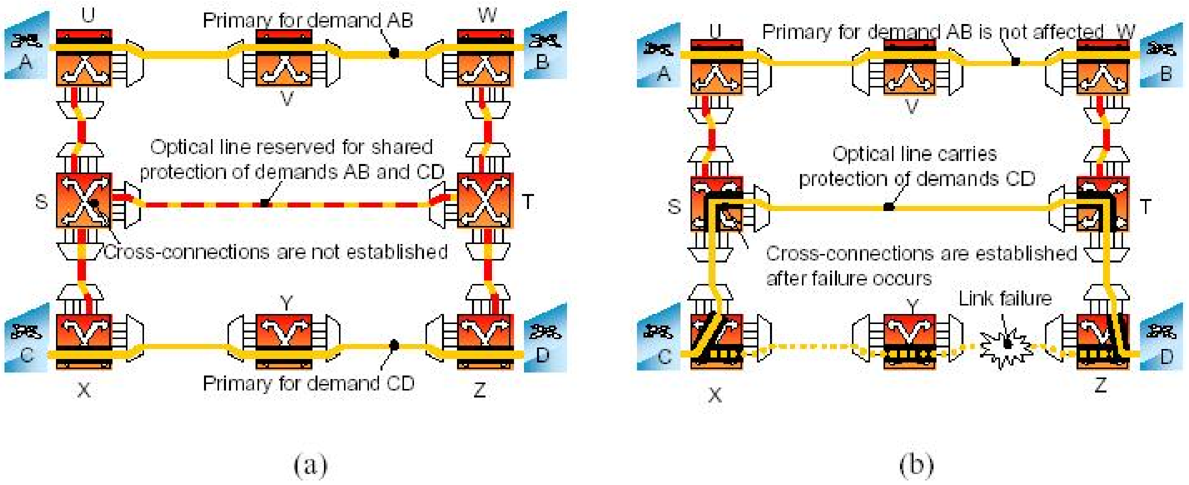

U
Primary for demand AB
W
A
B
V
Optical line reserved for shared protection of demands AB and CD
S
T
Cross-connections are not established
Y
C
D
X
Primary for demand CD
Z
(a)
U
Primary for demand AB is not affected
W
A
B
V
Optical line carries protection of demands CD
S
T
Cross-connections are established after failure occurs
Y
Link failure
C
D
X
Z
(b)

Figure 11 : Shared protection in optical WDM

The above picture shows an optical WDM network where optical cross connectors use shared protection [24].

In (a) we can see how the shared protection for the demands AB and CD is predefined but the cross-connections along ST is not created until a failure occurs. Since the capacity on ST is not reserved, this optical channel can be shared to protect multiple light paths. This means that in this example one less channel is used then in dedicated protection with pre-reserved resources. In (b) a link failure has occurred on the link between Y and Z and CD traffic now travels on the path between ST. The restoration involves signaling to establish the cross-connections along this path.

Shared protection is more complex to maintain then dedicated protection. However shared protection can offer higher network utilization. In the dedicated case each backup resource has to be reserved, in the shared case that reservation can be shared. For shared protection, since multiple backup paths share common recourses, the total number of backup resources can be potentially much lower.

4.4 Topologies:

In this section I will look at how the topology used in the network can affect how recovery can be performed.

4.4.1 Mesh Topologies

A mesh is a network topology in which devices are connected with many redundant interconnections between network nodes. There are two types of mesh topologies: full mesh and partial mesh.

Full mesh topology occurs when every node has a connection to every other node in a network. Full mesh is very expensive to implement but yields the greatest amount of redundancy, so in the event that one of those nodes fails, network traffic can be directed to any of the other nodes. Full mesh is usually reserved for backbone networks.

Partial mesh topology is less expensive to implement and yields less redundancy than full mesh topology. With partial mesh, some nodes are organized in a full mesh scheme, while other nodes are only connected to one or two other nodes in the network. Partial mesh topology is commonly found in peripheral networks connected to a full meshed backbone.

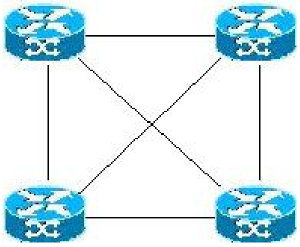


Figure 12: Fully mesh

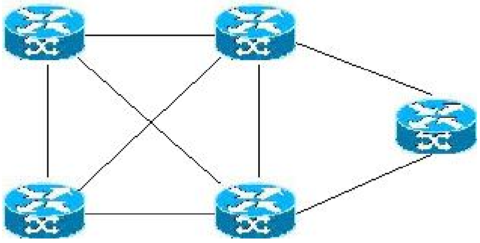


Figure 13 : Partial mesh

A mesh network is reliable and offers redundancy. If one node can no longer operate, all the rest can still communicate with each other, directly or through one or more intermediate nodes.

The chief drawback of the mesh topology is expense, because of the large number of cables and connections required.

4.4.2 Ring Topologies

A common way of recovery at the lower layer is to have a network topology in the form of a ring. These ring topology networks are often called self-healing rings. In a self-healing ring each node in the network are connected with two links to its upstream and downstream neighbors. In normal operation, traffic is sent from a source to a destination in one direction of the ring. If a link fails, then the traffic is sent on the other link in the reverse direction so that the failed link is avoided.
In this section I will look at three different physical layer rerouting mechanisms which all rely on ring topologies. SONET/SDH, FDDI and RPR.

SONET/SDH

SONET (Synchronous Optical NETwork) is a physical layer technology for synchronous data transmission over fiber optic networks. This technology is used for long-distance transmission of data over optical networks. SONET/SDH was designed as a means to deploy a global telecommunication system, so SONET/SDH is widely deployed by the world carriers. With a technique call Packet over SONET/SDH (PoS) IP datagrams can be mapped into the SONET/SDH frame payload using the Point-to-Point protocol (PPP).

SONET is the United States version of the standard, published by the American National Standards Institute (ANSI). SDH (Synchronous Digital Hierarchy) is the European version of the standard published by the International Telecommunications Union (ITU). In SONET/SDH, protection with self healing rings is called ring APS (Automatic Protection Switching). Ring APS comes in two versions called UPSR and BLSR. It is also possible to use protection switching in SONET/SDH networks, this is called Linear APS.

Recovery operations in SONET/SDH has set a standard for how fast recovery operations shall be performed, and often other models for recovery is compared with the times that is used for recovery in SONET/SDH networks. The switchover time in a SONET/SDH network is ~50ms and these networks are therefore well suited to transport traffic that is sensitive to delay if a failure should occur.

SONET UPSR (Unidirectional Path-Switched Ring):

In SONET UPSR [19] a source sends exactly the same traffic in reverse directions on both rings. The destination receives traffic on both rings but takes only traffic on one ring in account. If a fail occurs the receiver detects the absence of traffic on the main ring and then decides to take the traffic on the other ring in account. As explained before this kind of protection uses 1+1 protection switching. Traffic that is sent in the reversed directions is used to protect from node failures. If a node goes down, traffic will still reach the destination from the other direction. With this type of protection traffic can be restored very fast and meets the 50 ms goal set for voice traffic. But SONET UPRS requires a substantial amount of dedicated backup resources on the reverse direction link, as in this technique half of the links are used for path restoration purpose only.

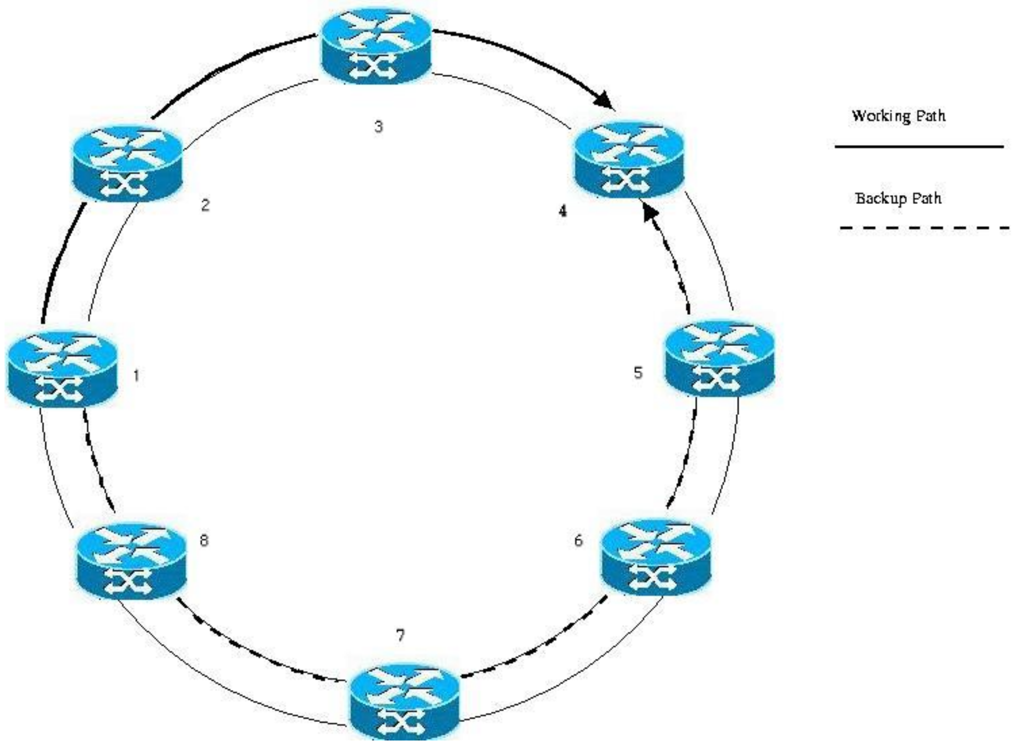


Figure 14 : SONET UPRS

In the picture above the UPRS functionality is illustrated. Router 1 sends traffic to router 4 through both directions of the ring. In normal mode the working path is the path (1,2,3,4) but if a failure occurs on this path, router 4 starts to use the traffic received in the other direction (1,8,7,6,5,4) from router 5.

SONET BLSR (Bidirectional Link-Switched Ring)

In SONET BLSR[20] every link can carry both regular traffic and backup traffic and thus no dedicated backup links are required. This type of protection mechanism is 1:1.
If a link failure occurs the node upstream of the failure wraps traffic from one ring to the other ring in the reverse direction. BLSR does not use as much resources as UPSR and there is no notation of dedicated primary and backup link.

In the below picture router 1 sends traffic to router 4 through the working path (1,2,3,4). When a failure occurs on the link between router 2 and 3, router 2 wraps the working path and starts to forward traffic from the working path onto the backup path in the reversed direction. Traffic from 1 to 4 then travel the way through routers (1,2,1,8,7,6,5,4).

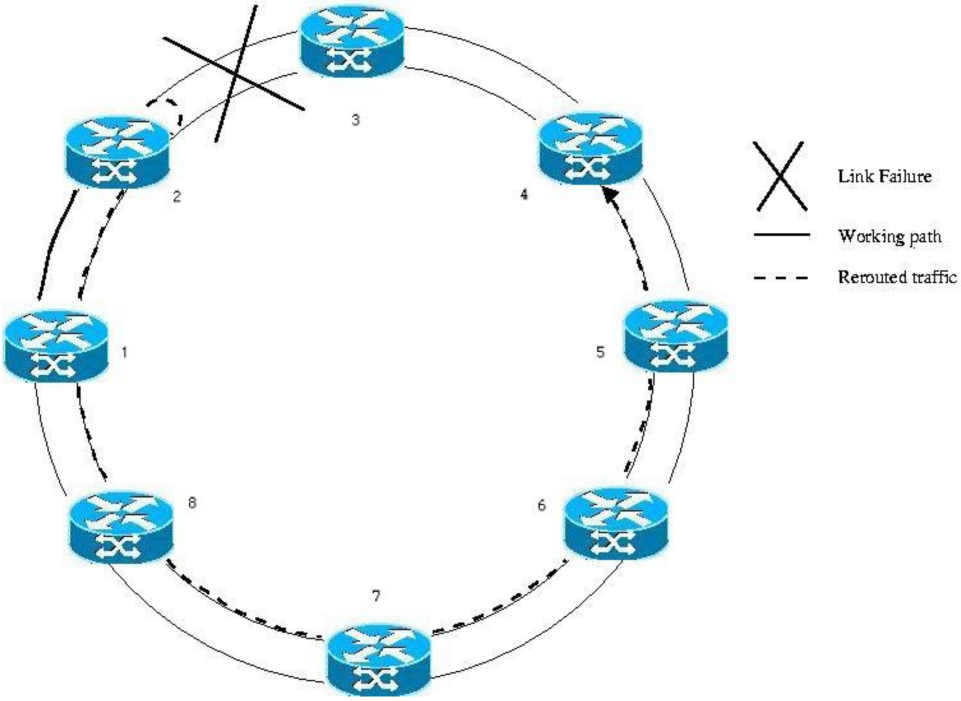


Figure 15 : SONET BLSR

FDDI (Fiber Distributed Data Interface)

FDDI implements at the MAC layer a protection mechanism that is similar to SONET BLSR. The FDDI standard was produced by the ANSI X3T9.5 standards committee in the mid-1980's. FDDI uses dual-ring architecture with traffic on each ring flowing in opposite directions. The dual rings consist of a primary and a secondary ring. During normal operation, the primary ring is used for data transmission, and the secondary ring remains idle. If a link failure occurs or a node goes down, the nodes adjacent to the failure wraps traffic on the backup ring. The switchover time in FDDI is larger then for SONET/SDH and often around 200ms.

RPR (Resilient Packet Ring)

In RPR [21], path protection is called IPS (Intelligent Protection Switching). IPS can be seen as an enhanced SONET BLSR mechanism. When a link failure is detected, traffic is first wrapped exactly like in SONET BLSR on the reverse direction of the ring. But in RPR the sending node is also notified of the failure and changes the ring it sends traffic on. The new path the traffic take is therefore shorter then the wrapped path, which result in a better use of resources then in SONET BLSR.

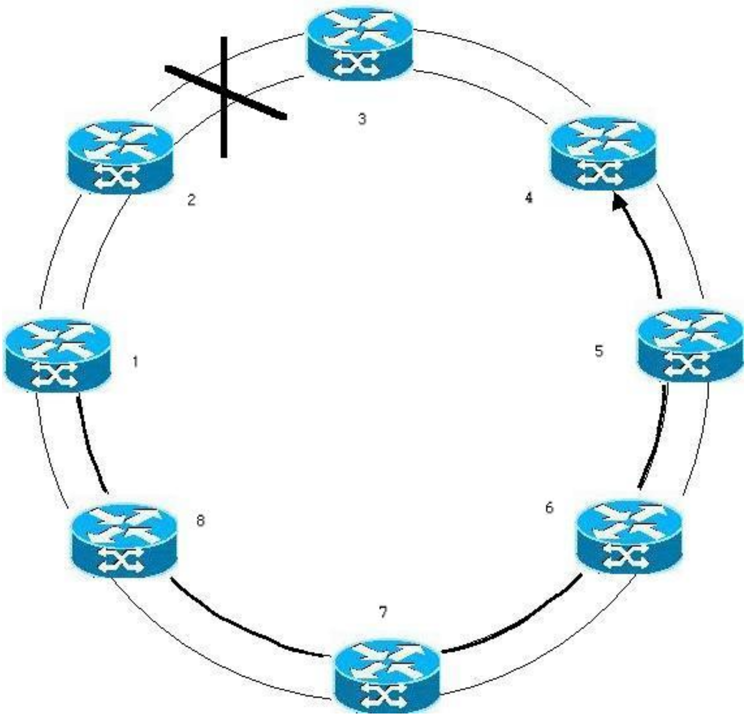


Figure 16 : Resilient Packet Ring

In the picture above router 1 sends to router 4 on the path (1,2,3,4). When a failure occurs on the links between router 2 and 3 traffic is first wrapped by router 2 onto the backup path as in SONET-BLSR. The sending router 1 is then informed about the failure on the working path and decides that is faster to send onto the backup path directly then through router 2. So router 1 starts to send traffic that would have passed the failed link in the reverse direction of the link instead. The new working path for traffic between router 1 and router 4 is therefore updated to (1,8,7,6,5,4). RPR claims to have the same recovery times as SONET/SDH and a switchover can be performed ~50ms.

4.4.3 Hybrid Cycle / Mesh

Recovery with self healing rings mechanisms offer very fast recovery times (around 50 ms), but the required spare to working resources ratio can be very high. For mesh protection mechanisms, the required spare to working ratio is typically lower for a well planed physical network [25]. Since recovery in mesh networks often involves a complex signaling, restoration in mesh networks is generally slower then in ring networks. To combine the best properties of from both ring and mesh a new concept called p-cycles has been introduced. The method of p-cycles is based on the formation of closed paths, formed in the spare capacity only of a mesh restorable network. They are formed in advance of any failure, out of the previously unconnected spare capacity units of a restorable network [26]. With the hybrid cycle/mesh approach, the p-cycle concept is able to benefit from the advantages of both worlds.

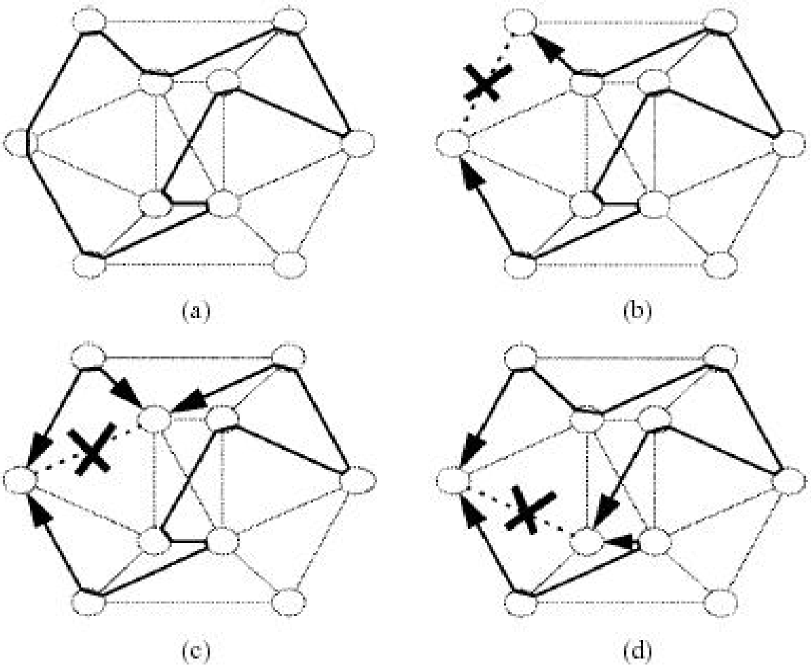


Figure 17 : P-Cycle

In the figure above we can see the p-cycle in (a). In (b) a link failure has occurred on the p-cycle and a restoration path for the link is available on the cycle. In (c) and (d) a link failure has occurred that is not on the p-cycle that results in two different restoration paths on the p-cycle.

In difference to conventional SONET/SDH rings where only links on the ring is protected and in the case of a link failure only one possible backup path is available, with p-cycles links off the ring can also be protected and different protection paths can be available if a failure occurs. In the graph above the 9 node ring can protect against 19 possible link failures in difference to 9 if the nodes was connected in a SONET/SDH ring [27]. In p-cycles the spare capacity design takes little or no more capacity than a corresponding mesh network and the recovery operations with p-cycles is essentially just that of BLSR. This means that p-cycles can offer the speed of rings with the efficiency of mesh [26].

4.5 Network layer recovery

In packet switched networks like the Internet the recovery mechanisms rely on the capabilities of the routing protocols. If a failure occurs then each router in the network has to be informed about the failure and the routing tables in each router have to be recomputed using a shortest path first algorithm. When routing tables has been recomputed and converted, all paths that were using a failed link or node are rerouted through other links.

To detect when a failure has occurred, routing protocols use timers. The routing protocol on a node periodically sends hello messages to its neighbor nodes. If a neighbor doesn't receive a preset number of hello messages in a time interval, then the routing protocol interprets this as a failure on the link to that node. When a network failure has been detected then all the nodes in the network has to be notified about the failure. This is done by the node that detected the failure, which sends a LSA (Link State Advertisement) to all other nodes in the network. Then each node has to perform a shortest path first calculation with the failed link pruned from the network. When all nodes have finished the calculation, new routing tables are updated and traffic can be rerouted in the network.

Routing protocols make the network able to survive one or multiple link or node failures, but they do not guarantee the recovery time. The recovery time strongly depends on the dimension of the network and on the routing protocol used. The IP layer relies on the physical layer that provides transport of IP packets between two points in the network. There is no standard mechanism for exchange of information about network status between the IP layer and the lower layers, therefore routing protocols in IP run transparent as regards of type and structure of the physical layer.

With the current default settings in routing protocols as OSPF, the network takes several tens of seconds before recovering from a failure. The main reason for this is the time it takes before a failure is detected using the hello protocol [16]. There have been some proposals [17] that the recovery time could be reduced by reducing the value of the hello message interval. But it has been showed that a reduction of this interval can cause problems. If the interval is to small then it will result in an increased chance of network congestion causing loss of several consecutive hellos, thus a node may think that a link is down that is only congested. It will then flood the network with LSAs causing all the routers to make shortest path first calculations. Then it will start to receive hello messages again and conclude that the link is up, thus flooding the network with new LSAs. This will not only lead to unnecessary routing changes but also increase the processing load on the nodes.

4.6 Comparison of the different layer mechanisms

Various technologies operating at different layers provide protection and restoration capabilities at different time scales, ranging from a few tens of milliseconds to minutes, at different bandwidth granularities (i.e., from packet-level to wavelength level), ranging from a few kilobits per second to hundreds of gigabits per second, and at different QoS granularities, ranging from aggregated traffic classes (e.g., DiffServ classes) to individual traffic flows [28].

In traditional IP networks, only one class of service was supported the best-effort class. Network survivability in such an environment involves the restoration of network connectivity, which is provided by layer 3 re-routing alone, since this is all that is needed for best effort traffic. When recovery rely on the routing algorithms alone the time that the routing algorithms take to converge and restore service can be significant, in the order of several seconds to minutes, causing a disruption of service. This may not be a concern with best-effort traffic, but it does become a significant concern when the aim is to provide applications requiring a highly reliable service, where the recovery time must be in the order of tens of milliseconds to not affect the service transported. If such traffic should be carried by the network, then layer 3 recovery is not sufficient. The IP layer provides a fine granularity of protection as at this layer a path selection algorithm to pick paths based on priority and other requirements of the traffic can be uses as in constraint-based routing. But a problem with layer 3 is that it can not detect physical layer faults, the IP layer may only be aware of the existence of a fault by the loss of hello or keep alive messages in the routing protocol, but it may not know where the fault is (link or node failure).

Ring topologies and linear APS can reroute traffic in less then 50ms, which is set to be the limit for recovery in SONET/SDH. This is much faster then in layer 3 recovery by routing table updates, and is in the limits for highly reliable services. This can be realized because failure detection can be done directly in hardware without the need for hello or keep alive messages, and the use of automatic protection switching to a dedicated backup path. But this requires the expensive hardware used in those systems. In SONET/SDH the topology is largely limited to rings, where spare capacity often remains idle, making the efficiency of bandwidth usage an issue. In some implementations up to half of the available bandwidth is unused as it is only dedicated for backup purposes. SONET/SDH networks can not distinguish between different priorities of traffic. It is therefore not possible to switch prioritized traffic classes like EF or AF traffic differently from best effort traffic, all traffic affected by a failure follows the same backup path in the ring. When recovery is done at this layer, it is not aware of errors on the higher layers, therefore a packet that has been corrupted at the ATM, MPLS or IP layer will not be detected as a failure.

In a mesh topology there are more alternatives for backup paths then in a ring topology. In ring topologies working traffic is sent in one direction of the ring and when a failure happens in the ring traffic is sent in the different direction. In a mesh topology different paths can be setup for backup, and shared protection can be used. But this often involves notification to other nodes and is therefore requires more time.

Lower layer recovery is fast in the cases of local recovery, where the node that detects the failure can perform the switchover step directly to a backup path that is pre setup. This means that a switchover can be done directly without the need to wait for other nodes to be notified about the failure, before traffic can be switched to an alternative path. As in 1+1 recovery where the end node can detect when the traffic on the working path is failing (by for example loss of light) and switch to receive traffic from the backup path instead. This has the capacity to restore a large number of higher layer flows affected by a single link failure and restore them simultaneously without any signaling. The total repair time is therefore reduced to the detection time of the failure, but this has limited range of granularity. It can not restore individual circuits, paths or types of traffic.

If shared protection is used, then some type of signaling must be used which adds to the time used for recovery. Lower layer protection is often used to recover from link failures, and can not in general detect nodes failures. It may therefore not be able to provide path protection at the higher layers as in MPLS LSPs or ATM virtual circuits.

The time-scale of the recovery operation is an important factor in determining which layer that shall perform network recovery. In a generic sense the closer to the fault, the faster the recovery. However, faults occur at different layers and not all layers have visibility to all faults at the different layers. If higher layers shall perform the recovery then more time is used for the recovery operation, but at higher layers we can have a finer granularity over which type of traffic that needs to recover fast and therefore less bandwidth could be kept pre reserved dedicated for recovery. If backup resources are not pre reserved, then some kind of signaling has to be used if recovery not is left to layer 3 with recalculations of routing tables.

As we have seen re-routing at the lower layers is fast but requires dedicated hardware. And in some techniques dedicated backup paths is used that does not use the available links in an optimal way. On the other hand, IP re-routing is slow but does not relay on any specific topology and is implemented on every router on the Internet. In MPLS, which is implemented between the IP layer and layer 2, we have a chance to implement recovery mechanisms that provide a trade-off between repair speed and deployment cost. There are some motivations why recovery should be implemented by MPLS and not left to other layers.

IP routing may not be able to provide bandwidth recovery, where the objective is to provide not only an alternative path for the traffic, but also a path where bandwidth is equivalent to that on the original path. This is necessary in QoS networks where users have paid for a high priority service.

MPLS recovery can be motivated by the fact that there are limitations to improving the recovery times of current routing algorithms at the IP layer. As we saw routing algorithms has to go to the steps of detection, notification and SPF calculations before traffic is rerouted. But in lower layers re-routing can be done directly after the failure is detected.

MPLS could enable IP traffic to be put directly over WDM optical channels and provide a recovery option without an intervening SONET/SDH layer.

MPLS has desirable attributes when applied to the purpose of recovery for connectionless networks. Specifically that an LSP is source routed and a forwarding path for recovery can be pinned and is not affected by transient instability in SPF routing brought on by failure scenarios.

Lower layer mechanisms may have no visibility into higher layer operations. Thus, while they may provide for example protection of one link, they cannot easily provide node protection or protection of traffic transported at layer 3. Further, this may prevent the lower layers from providing restoration based on the traffic's need. For example fast restoration for traffic that needs it like voice over IP traffic, and slower restoration for traffic that does not require fast restoration. In networks where the latter class of traffic is dominant, providing fast restoration to all classes of traffic may not be cost effective from a service provider's perspective.

5. MPLS recovery

In this chapter I will look at how recovery can be implemented in the MPLS domain. I will look at the basic mechanisms that can be used for recovery and some of the proposed models for recovery.

General recovery terminology in the MPLS domain:

In RFC 3469[32] there are some general terminology defined for recovery in the MPLS domain. This section contains explanations of the terms that will be used in this thesis.

Working Path
Working path is the name of the path that carries traffic before the occurrence of a fault. If there is a recovery mechanism for that path, the working path is said to be a protected working path. Synonyms for the working path are primary path and active path.

Recovery Path
Recovery path is the path to which traffic is rerouted after the occurrence of a fault. In other words, the path on which traffic is directed by the recovery mechanism. The recovery path can either be an equivalent recovery path and ensure no reduction in quality of service, or be a limited recovery path and thereby not guarantee the same quality of service as the working path. Synonyms for a recovery path are: backup path, alternative path and protection path.

Path Switch LSR (PSL)
An LSR upstream of the fault in the working path that is responsible for switching or replicating the traffic between the working path and the recovery path.

Path Merge LSR (PML)
An LSR that is responsible for receiving the recovery path traffic, and either merge the traffic back onto the working path, or if it is itself the destination, passing the traffic on to the higher layer protocols.

Point of Repair (POR)
A point of repair is an LSR that is configured to perform MPLS recovery. In other words, an LSR that is responsible for the repair of a LSP. The POR, for example, can be a PSL or a PML, depending on the type of recovery scheme employed.

Bypass Tunnel
A bypass tunnel is a path that serves to back up a working path or a set of working paths using the label stacking approach. The working path(s) and the bypass tunnel must share the same path switch LSR (PSL) and path merge LSR (PML).

MPLS Protection Domain
The protection domain is the set of LSRs over which a working path and its corresponding recovery path are routed.

Revertive Mode
Revertive mode is a recovery mode in which traffic is automatically switched back from the recovery path to the original working path when the working path is restored to a fault free condition. This assumes that a failed working path does not automatically surrender resources to the network.

Fault Indication Signal (FIS)
FIS is a message that indicates that a fault along a path has occurred. It is relayed by each intermediate LSR to its upstream or downstream neighbor, until it reaches an LSR that is setup to perform MPLS recovery (POR). The FIS is transmitted periodically by the node/nodes closest to the point of failure, for some configurable length of time or until the transmitting node receives an acknowledgement from its neighbor.

Fault Recovery Signal (FRS)
FRS is a message that indicates that a fault along a working path has been repaired. Again, like the FIS, it is relayed by each intermediate LSR to its upstream or downstream neighbor, until it reaches the LSR that performs recovery of the original path. The FRS is transmitted periodically by the node/nodes closest to the point of failure, for some configurable length of time or until the transmitting node receives an acknowledgement from its neighbor.

5.1 Failure Detection / Notification in MPLS

MPLS introduces a new layer in the network model that is independent of what other layers that is used in the network. It is therefore necessary that MPLS also has its own mechanisms for failure detection. Even if the lower layers have fast failure detection mechanisms, nothing can be assumed about those layers because MPLS is supposed to work with many different network types.

5.1.1 RSVP-TE Hello Extension

The RSVP Hello extension enables LSRs to detect when an adjacent node is not reachable. The extension is composed of a Hello message, a HELLO REQUEST object and a HELLO ACK object. Hello processing between two adjacent LSRs follows an independent configured failure detection interval where the default value is set to 5 ms. Failure detection is accomplished by collecting and storing an adjacent LSRs “instance” value. If a change in value is recorded the neighbor is presumed to have reset. Each LSR periodically generates Hello messages containing the HELLO REQUEST object with that LSRs “instance” value and the previously received “instance” value from its neighbor. When the neighbor node receives the message it compares the senders “instance” value to the last received value from that LSR. It also checks if the sender returns the last “instance” value used by the receiver. If

any of these values differs then the neighbor is presumed to have reset. The receiver of the HELLO REQUEST then generates an HELLO ACK message. If the received values were correct the ACK message includes the senders and receivers “instance” values, otherwise the sender of the ACK sets the receiver’s value to zero to indicate that communication has been lost. If no hello messages are received within a configured number of hello intervals then a node presumes that it can not communicate with its neighbor. The default for this number of hello intervals is set to 3.5 ms.
The hello protocol can therefore detect if a LSR is not reachable or if it has restarted. It is therefore used to check if a LSRs control plane is working and if a LSR is reachable, this mechanism can not be used to check the forwarding plane.

5.1.2 RSVP-TE Softstate

As explained earlier RSVP-TE is a soft state protocol. This means that LSPs signalled with RSVP-TE is constantly updated with PATH and RESV messages. If a PATH or RESV message fails to update the LSP a PathErr or a ResvErr message is returned from the point where reservation failed to the ingress node of the LSP. If crankback is used information about why the path update failed is included in the error message so that new attempts to setup the LSP can be done. The interval in which soft state updates are sent is often kept high to decrease the amount of control traffic sent by the signalling protocol. The default value for sending out refresh messages is often set to 30 sec. This makes failure detection by soft state refresh messages unsuitable for fast reroute mechanisms.

5.1.3 LSP Ping/Traceroute

In [47] a mechanism is introduced that is modeled by the Ping/Traceroute paradigm to check the forwarding plane of LSRs in a LSP. With this model it can be checked that packets that belong to a FEC actually end their LSP at the correct egress LSR. In Ping mode a packet encapsulate by an UDP packet is forwarded down a LSP like any other packet belonging to that FEC. This packet is called an echo request. When it arrives at the egress LSR it is checked against the control plane that this LSR really is the egress router for this FEC. Then a packet called echo reply is sent back with the result to the router that requested the ping. In Traceroute mode the packet is checked against the control plane on each LSR in the LSP to check if this router really is a transit router for that FEC. Trace route is therefore a fault isolation tool. These tools can be used to periodically ping a FEC, if that ping fails a traceroute can be initiated to determine where the fault is located. This mechanism can also be used to periodically traceroute FECs to verify that the forwarding plane matches the control plane. However, this places a computational burden of the transit LSRs.

In [48] a mechanism called Bi-directional Forwarding Detection (BFD) is presented that is a more light-weight testing of the forwarding plane, this mechanism is still in the early stages of development. Ping is designed to be used in en environment where fault detection messages are exchanged in the order of seconds. Therefore it is not appropriate for fast failure detection. BDF on the other hand is being designed for sub-second fault detection intervals.

Even though BDF aims to decrease the failure detection time of the forwarding plane, some precautions has to be concerned. If BDF shall work together with fast rerouting mechanisms for link or node failures in MPLS, the detection interval for BDF has to larger then the recovery switch over time. Otherwise false fault detection can occur. When a rerouting mechanism is started because a link or node failure, BDF packets will be dropped until traffic is switched on to a recovery path. Then BDF will act like a failure has occurred even though the LSP is being locally repaired. To avoid this, the BDF fault detection interval has to be greater then the recovery switchover time.

5.1.4 Graceful restart

As described earlier MPLS routers has two architectural planes, the control plane and the forwarding plane. These two planes are handled by two different processors. The dedicated route processor calculates routes based from information from neighbors and then downloads these routes to the forwarding table. Forwarding can either be centralized in one processor or distributed among multiple line cards. The routing functionality is referred to as the control plane and the forwarding functionality as the forwarding plane. This separation makes it possible for a router to still forward packets correct even if the control plane is temporarily down, because the forwarding table still contains valid routes regardless of the condition of the route processor.

But when the control plane in a router goes down and restarts the control channel between the router and its adjacent router restarts. Then all LSP’s traversing the router is terminated because the control channel is cut. To prevent this, a technique called graceful restart is used. In graceful restart for RSVP-TE a new object called a Restart_cap object is included in the hello message [43] [44]. This object includes two fields called Restart time and Recovery Time.

The Restart time field is set to the time it takes for a router to restart its control plane until it can start to send hello messages again. The Recovery time is set to the time a router needs to re-synchronize RSVP and MPLS forwarding state after the re-establishment of Hello synchronization. A value of zero indicates that the MPLS forwarding state was not preserved after the reboot. When a node detects that RSVP communication with a neighbor has been lost, the node waits the amount of time indicated by the Restart time before invoking procedures for communication loss. During this period the node preserves the RSVP and MPLS state for established LSPs that traverse the restarting node. The adjacent nodes behave as if they still receive RSVP messages from the restarting node. When the node has restarted it starts to send RSVP messages again and the adjacent nodes can then resynchronize the control channel again.

Graceful restart is also defined for LDP in [45].

5.1.5 Failure reports from other layers

Failures can also be detected by lower layer mechanisms and reported to the MPLS layer.

In this case it is important that the lower layer does not have a recovery mechanism implemented. In that case recovery will be performed at both the lower layer and at the MPLS layer. That means that MPLS will act as a failure is detected, but when the recovery operation is finished by MPLS, the lower layer recovery will have finished its own recovery mechanism and the connection is working again. MPLS will then see that the failing component is working again and that the LSP can be rerouted back to the original working path again. This makes it important that if recovery is implemented on lower layers, then failures that can be recovered at those layers shall not be reported to the MPLS layer. Failure detection mechanisms at lower layers are often faster then the failure detection mechanism associated with MPLS. It is therefore often better to use this kind of failure detection mechanism if available.

5.1.6 Summary

When RSVP-TE is the protocol used for label distribution, the soft-state properties of this protocol can be used for failure detection. But the values for the path refresh interval in RSVP-TE are often set high to decrease the amount of control traffic, which makes this failure detection method unsuitable for fast recovery operations.

Link failures can be detected fast by hardware and reported to the MPLS module or a hello mechanism as in RSVP-TE can be used. The RSVP-TE hello mechanism can also detect node failures and control plane failures. Control plane failures do not necessary lead to forwarding failures and therefore graceful restart can be used to prevent forwarding from stopping when a control failure occurs.

The polling used to find failures by the RSVP-TE hello extension is not as fast as failure detection mechanisms at lower layers, were failures can be detected quickly by hardware. But it is necessary that MPLS has a failure detection mechanism that is independent of the underlying layers, in cases were fast failure detection is not available by the lower layer or were the lower layer can not be used to detect node failures.

Data plane failures on the other hand can not be detected fast. To detect this kind of error Ping and Traceroute mechanisms are needed and those mechanisms are not suited for fast failure detection. The BDF proposal from IETF is an attempt to make failure detection in the data plane faster.

5.2 MPLS recovery mechanisms

As in recovery mechanism for other layers, MPLS recovery operates by forwarding traffic on a new path around the point of failure in the network. Where this path is placed, when it is calculated and how it is setup depends on the recovery mechanism used.

If no recovery mechanism is used in an MPLS domain, recovery is performed by the default action by the signaling protocol used to setup and maintain a LSP, this is called *best-effort restoration*. In the case of RSVP-TE, a failure can be detected by hello messages or by the RSVP-TE soft state mechanism. As described earlier the soft state mechanism in RSVP-TE keeps the LSP reserved by periodically sending PATH and RESV messages. If a failure occurs on the LSP, the soft state will fail to update the reservation and depending on if it were the PATH or RESV message that failed, a PathErr or ResvErr message will be sent back upstream to the ingress LSR of that LSP. If a failure is detected by the hello message state, then a PathErr will be sent back to the ingress LSR. When the failure is detected and notified to the ingress LSP it will try to setup the LSP again. If the LSP was loosely routed, the ingress can try to find a link and node disjoint path to the egress and try to setup the LSP on the new path instead. Or it can try to setup the same LSP again and let the node adjacent to the failure compute a new path from the point of failure to the egress. Once the path is established the traffic can start to flow on the new LSP. If the LSP was strictly routed the ingress will periodically try to setup the path again, until it becomes available. Restoration time of this scheme is typically large and is only acceptable for best effort type of traffic.

If recovery shall be performed faster other MPLS mechanisms need to be used that can try to minimize the failure notification time or backup path computation time.

The path recovery mechanisms in MPLS can be classified according to two criteria: recovery by rerouting vs. protection switching and local repair vs. global repair [33].

5.2.1 Recovery Path Placement

There are two main categories of where a recovery path is placed. These are called global repair and local repair.

Global Repair

The intent of global repair is to protect against any link or node failure on a path or on a segment of a path, with the obvious exception of the fault occurring at the ingress or egress LSR of the protected path segment. In global repair, the POR is usually distant from the failure and needs to be notified by a FIS. In global repair end-to-end path recovery can be applied, where the working path is completely link and node disjoint from the protection path. This has the advantage that all links and nodes on the working path are protected by a single recovery path. But it has the disadvantage that a FIS has to be propagated all the way back to the ingress LSR before recovery can start.

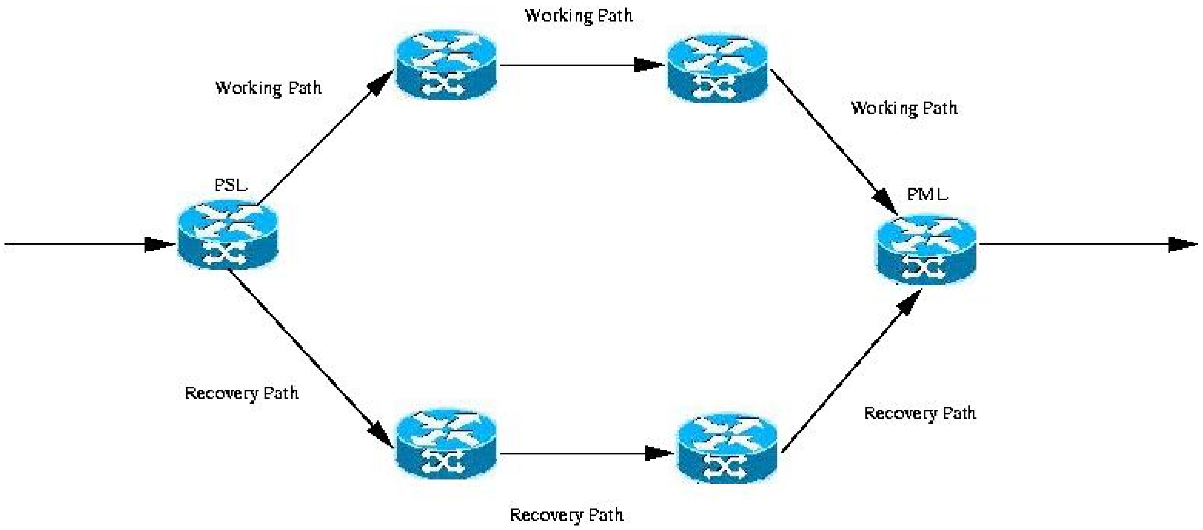


Figure 18 : Global Recovery Path

Local Repair:

The intent of local repair is to protect against a link failure or neighbour node failure and to minimize the amount of time required for failure propagation. If a repair can be performed local to the device that detects the failure, restoration can be achieved faster. In local repair (also known as local recovery), the node immediately upstream of the failure is the node that initiates the recovery operation (PSL). Local repair can be setup in two different ways.

Link Recovery

In link recovery the goal is to protect a link in the working path from failure. If a failure occurs on the protected link in the working path then the protection path connects the PSL and PML with a path disjoint of the working path with regard to the failed link.

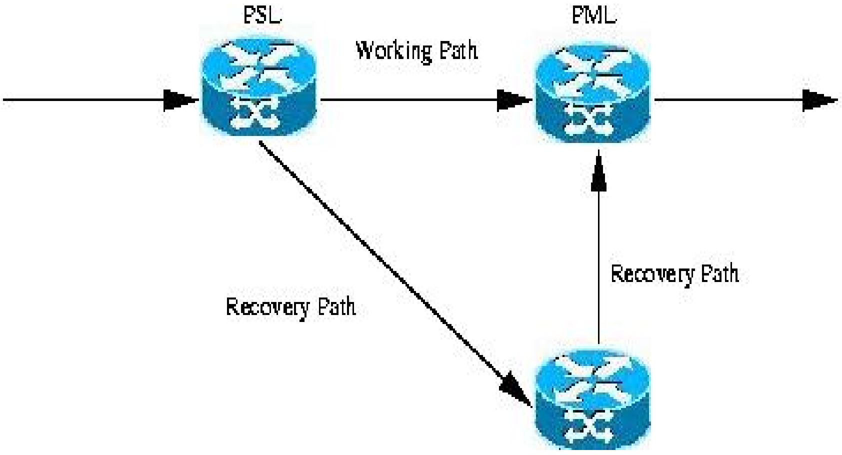


Figure 19 : Link Recovery

Node Recovery

In node recovery the goal is to protect a node in the working path from failure. In node protection the recovery path is disjoint from the working path in regard to the protected node and the links from the protected node, in difference from link protection where only the link between two adjacent nodes is protected. In node protection there will be one or more hops between PSL and PML.

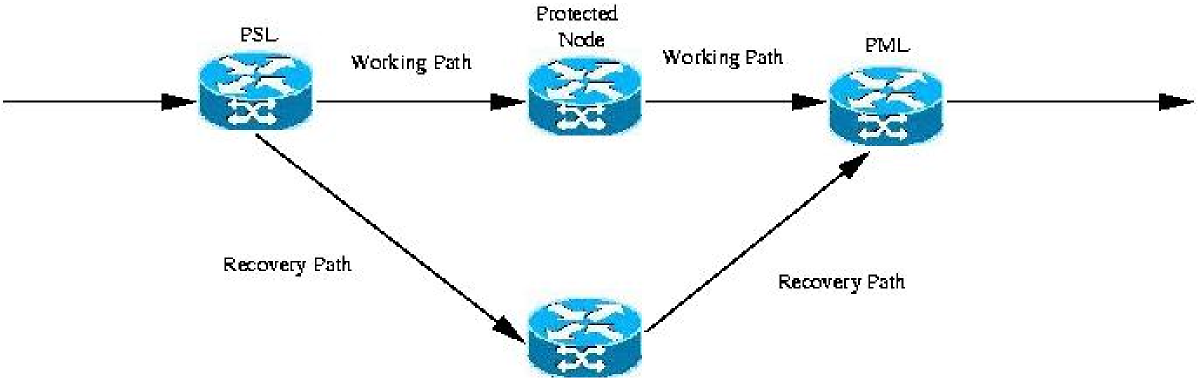


Figure 20 : Node Protection

Local protection has the advantage that the FIS don't have to be propagated all the way back to the ingress router before recovery can be started. As soon as a failure is detected, recovery can start by the LSP that detects the failure. It is therefore a faster alternative then global protection, but the disadvantage is that only segments of the working path are protected.

Segment Protection

An MPLS domain may be partitioned into multiple protection domains whereby a failure in a protection domain is corrected within that domain. In cases where an LSP traverses multiple protection domains, a protection mechanism within a domain only needs to protect the segment of the LSP that is within that domain. Segment protection will generally be faster, because recovery can start closer to the fault. If a working path traverses multiple protections domain, and each domain uses global protection in that domain. Then when a failure is detected the FIS only need to be propagated to the ingress LSP of that domain instead of the ingress LSP of the whole working path.

5.2.2 Recovery Path Calculation

A recovery path can either be calculated and setup at the time that a failure is detected, or it can be pre calculated and setup before the failure has occurred. This leads to two different way of recovery. These are called recovery by rerouting and recovery by protection switching.

Rerouting

In **recovery by rerouting** a recovery path is established on demand after a fault occur. The recovery path can be based upon fault information, network routing policies and network topology information. When a failure is detected in the network, an LSR upstream of the failure, setup as a POR can attempt to signal a recovery path that bypass the failure on the working path and merge that recovery path somewhere downstream of the failure with the working path. This recovery path can either be pre-computed or computed on demand. If the recovery path is selected on demand it is computed by the PSL when it has detected a failure, or when it has received a FIS that indicates that a failure has occurred in the working path. The PSL uses the current shortest path tree and prunes the failed link or node from the tree to find a recovery path. In this way it does not need to wait before link updates about the failure has been distributed in the network and a new shortest path tree has been computed. It then tries to signal the newly computed recovery path. When the recovery path is setup the PSL can start to forward traffic on the recovery path.

If a pre-computed recovery path is used no time for path calculation is used when the failure is detected, because the recovery path is computed and selected before the failure has occurred. But the recovery path is not signaled until the POR is aware that a failure has occurred. Therefore rerouting with a pre-computed recovery path is faster then when a recovery path is computed on demand, but if the path is pre-computed then the recovery path may not be the optimal recovery path at the time the failure occurs [35].

An advantage with recovery by rerouting is that it does not take up any backup resources in the network before the recovery path is signaled. But this may have the disadvantage that resources may not be available at the time the recovery path shall be set-up. There is a possibility that the set-up of the recovery path will fail, and there might be a need to recalculate a new recovery path, especially if the recovery path is pre-computed at the time when the working path is setup.

When the recovery path is pre-computed at setup time, it may not be the optimal recovery path when a failure occurs as network traffic flows change over time. Therefore a scheme was introduced in [35] where each LSP in a working path calculates a recovery path to its adjacent downstream LSP at set-up time. The recovery path is then re-calculated every time a LSP in the working path receives a routing update message. In this way a more updated and optimal version of the recovery path is available at the time of failure. This was an improvement over the existing pre-computed recovery path method, but it required additional resources from each LSP such as memory and CPU and this scheme only protected against link failures.

In [36] another approach was introduced that used node fail recovery instead of link fail recovery. If a recovery path shall be able to recover traffic from a node failure on the working path, then the node upstream of the failed node has to be aware of which nodes are downstream of the failed node, and not only which node that is the next downstream node. Therefore each node setup as POR in the working path has to remember its downstream nodes all the way to the egress LSR. This is done by recording the values in the ER object of the RSVP RESV message when the working path is setup. These nodes are kept as candidate nodes to be PML. When a failure is detected the LSP that detects the failure calculates a recovery path to each candidate PML and choose the one with the least cost that matches the conditions of the working path. In this way the optimal recovery path will be found. In the figure below, when LSR 5 has a failure LSR 2 will try to find recovery paths to LSR 8 and LSR 10. If hop count is the only condition for the recovery path the path LSR 2 -> LSR 4 -> LSR 7 -> LSR 10 will be selected as the recovery path. And the path LSR 2 -> LSR 3 -> LSR 6 -> LSR 9 -> LSR 8 will not be used as it has one more hop then the other path.

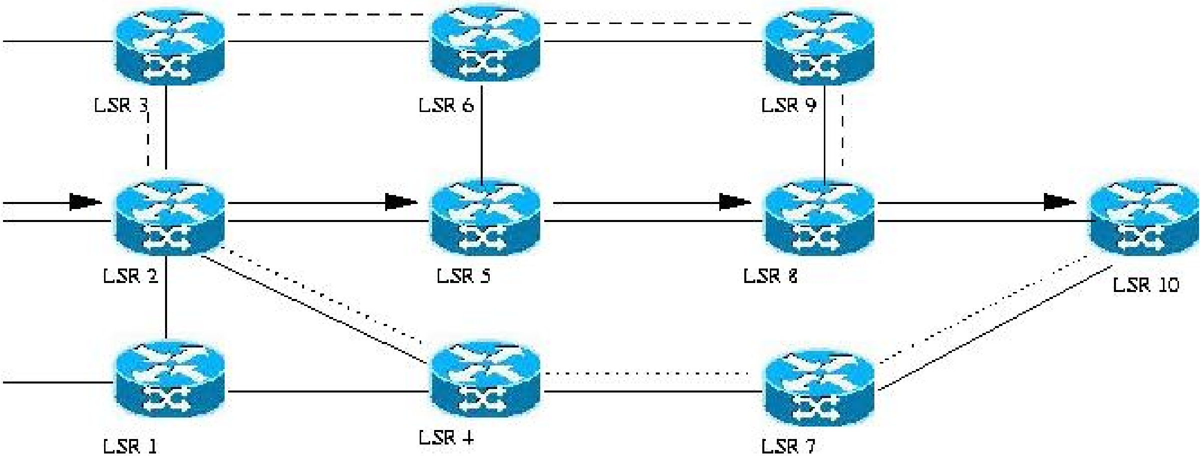


Figure 21 : Rerouting

In rerouting there is no guarantee that a recovery path will be found. If resources needed for the recovery path are used up in the network and the recovery path has lower priority then the traffic currently using the resources , then a recovery path might not be found.

Protection Switching

In protection switching a recovery path is pre-computed and pre-established before a failure occurs on the working path. The PSL is setup to switch traffic to the recovery path when it detects a failure or when it aware of a failure on the working path by reception of a FIS. Because the recovery path is pre-established no signaling is needed to setup the recovery path at the time when the failure is detected, this makes protection switching faster then recovery by rerouting.

Subtypes of Protection Switching

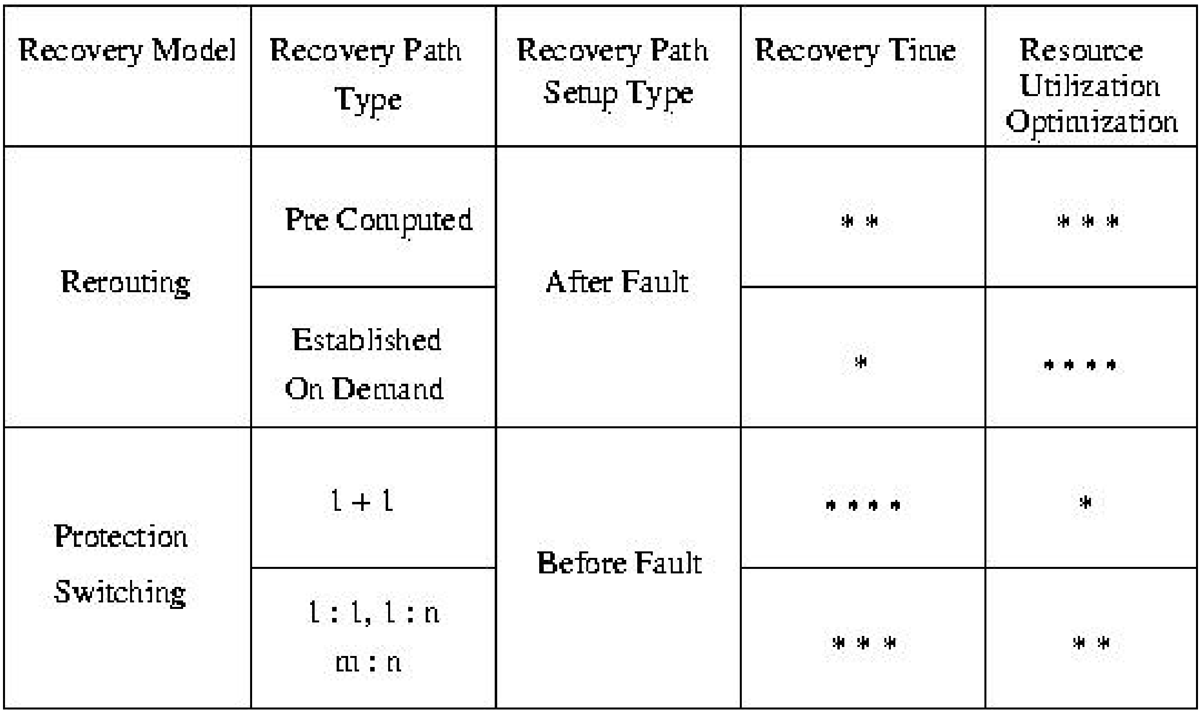
When the recovery path is pre-established some resources (bandwidth, buffers, processing) along the recovery path is kept reserved incase a switchover should occur.
These resources on the recovery path can be used to carry either a copy of the working path traffic or extra traffic. Extra traffic is traffic that has a lower priority then the traffic on the working path, that doesn't need to have a hard reservation. In the case of a switchover extra traffic can be dropped if not enough resources are available as explained in section 3.9. This leads to two subtypes of protection switching.

1+1 ("one plus one") protection, the resources (bandwidth, buffers, processing capacity) on the recovery path are fully reserved, and carry the same traffic as the working path. Recovery then only requires that the PML reads data from the recovery path, rather then the working path. This changeover is caused by the PML receiving a FIS from the point of failure. The PSL always sends the same traffic on both paths and has therefore not the same need of fast failure notification when a failure occurs. This is a fast recovery mechanism but its expensive in terms of resources used as the recovery path is fully reserved.

1:1 ("one for one") protection, the resources allocated on the recovery path are fully available to low priority traffic except when the recovery path is in use due to a fault on the working path. The recovery path is unused by working path traffic until the PSL receives a FIS, then traffic is switched over to the recovery path and lower priority traffic is no longer allowed to use the reserved resources on the recovery path. This concept can be extended to 1:N (one for N) and M:N (M for N) protection.

5.2.3 Comparison of Rerouting and Protection Switching

There are two main methods of recovery, protection switching and rerouting. Protection switching provides fast restoration of service but lacks efficient use of network resources as the recovery path is setup and resources reserved before a failure occurs. Rerouting has the advantage of optimizing the recovery path, and does not reserve any resources in the network before a failure occurs. The disadvantage is slower restoration of service. The different recovery models are compared in the table below.



Recovery Model	Recovery Path Type	Recovery Path Setup Type	Recovery Time	Resource Utilization Optimization
Rerouting	Pre Computed	After Fault	* *	* * *
	Established On Demand		*	* * * *
Protection Switching	1 + 1	Before Fault	* * * *	*
	1 : 1, 1 : n m : n		* * *	* *

Table 3 : Comparison of Rerouting and Protection Switching

If we compare these MPLS mechanism with non MPLS based recovery mechanism, protection switching in MPLS is much like lower layer recovery in the mechanism used. The difference is the granularity of what traffic that is being switched. In lower layers all traffic that passes through a point of failure is switched to the backup path when a failure occurs. This means that the backup has to have resources reserved for all traffic that passes through the point of failure. In MPLS the backup can be reserved per LSP or for a set of LSPs. This is preferable if not all traffic that passes by a point of failure has the same need for protection. In this way only enough resources for the selected traffic needs to be reserved and less recourses are kept reserved in the network.

For rerouting this is much like rerouting at the IP layer. The difference here is that it is not necessary to wait for LSA updates to be distributed in the network before traffic is rerouted. The switchover is therefore faster then IP rerouting.

5.2.4 Techniques to setup recovery paths

There are two different techniques to set-up recovery paths in MPLS [15]. These are called splicing and stacking. In both techniques a recovery path is set-up from the PSL to the PML, but the difference is in how labels are used by these. In splicing, the PSL change its forwarding table when the recovery path shall be used, to forward packets on the recovery path instead of the failed working path. A new outgoing interface and a new label are used by the PSL to forward packets on the recovery path. When the PML receives packets from the recovery path that uses a new incoming label and a different interface then the working path, it is setup to merge the recovery path traffic with the old working path.

In stacking the PSL also update its forwarding table to use a new outgoing label and a new outgoing interface for the affected LSP. But in stacking the new label for the recovery path is pushed on top of the label that would have been used for the failed working path. Then the next-to-last LSR in the recovery path pops the label stack revealing the old working path label before it forwards the packet to the PML. Then when the PML receives the packet it is unaware about the failure because it has the same label as a packet from the working path. There are two conditions that have to be fulfilled for stacking to work. The PML must use global label space so that it doesn't associate the incoming label with a specific interface, because traffic from the recovery path will have the same label as the working path but arrive at a different interface. The second requirement is that the PSL has to know the label it would have used on the working path for the PML. In link recovery this is easy because the label is the same as the one it would have used to forward traffic on the working path, but if the recovery path shall bypass one or more downstream nodes the PSL has to know the label that would have been used on the working path by the first upstream node from the PML. If RSVP-TE is used for label distribution the Record Route Object (RRO) and Session Attribute object can be used by the PSL to find out what labels the downstream routers use. If the stacking approach is used the recovery path is called a bypass tunnel. This is because it can be used for recovery by one or more LSPs as the PML does not have a separate incoming label for traffic received from the tunnel. Splicing can only be used to setup recovery for a single LSP as the PML merge the recovery path with the working path.

5.2.5 Path Mapping

Path mapping refers to the methods of mapping traffic from a faulty working path onto a recovery path. There are three main path mapping alternatives:

1-to-1 Protection: In 1-to-1 protection the working path has a designated recovery path that is only to be used to recover that specific working path.
n-to-1 Protection: In n-to-1 protection, up to n working paths are protected using only one recovery path.
n-to-M Protection: In n-to-m protection, up to n working paths are protected using m recovery paths.

As described earlier a backup path can be setup as a tunnel with the stacking method. In this way it is possible to use that tunnel for protection of more then one working path that needs protection for the same segment of the working path.

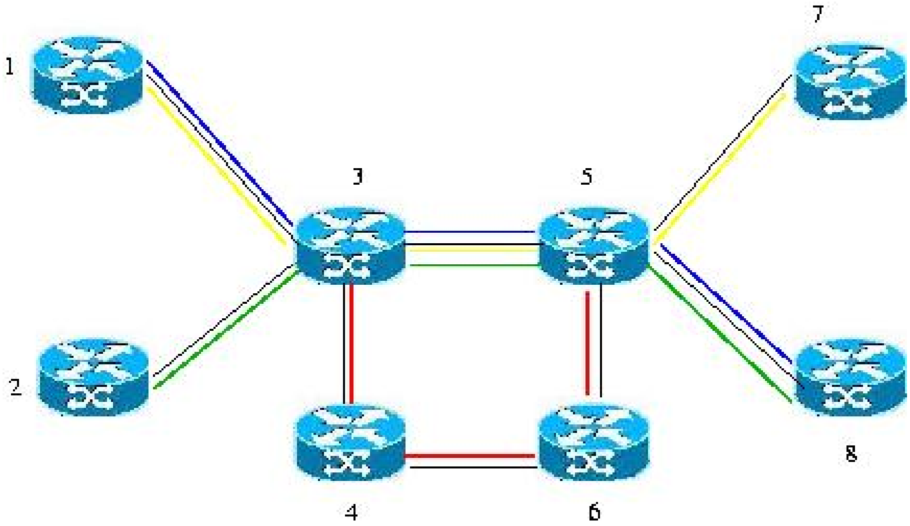


Figure 22 : Path Mapping

The above picture shows a network with three LSPs. (1,3,5,7), (2,3,5,8) and (1,3,5,8). If the link (3,5) needed protection a backup tunnel through (3,4,6,5) could be setup. Since all three LSPs pass through the same link they could use the same backup tunnel if they need protection on that link. This would decrease the amount of control traffic needed to keep a backup path reserved. For example: when 100 recovery paths protects 100 LSPs, each router along the backup path must maintain state for an additional 100 connections. But when one backup tunnel protects100 LSPs, each router along the backup path maintains one additional connection.

5.2.6 Protection Resource Sharing

Protection switching can keep a lot of resources reserved for backup purposes. Although those resources can be used by low priority extra traffic, it is desirable to keep as little resources as possible reserved for backup traffic.

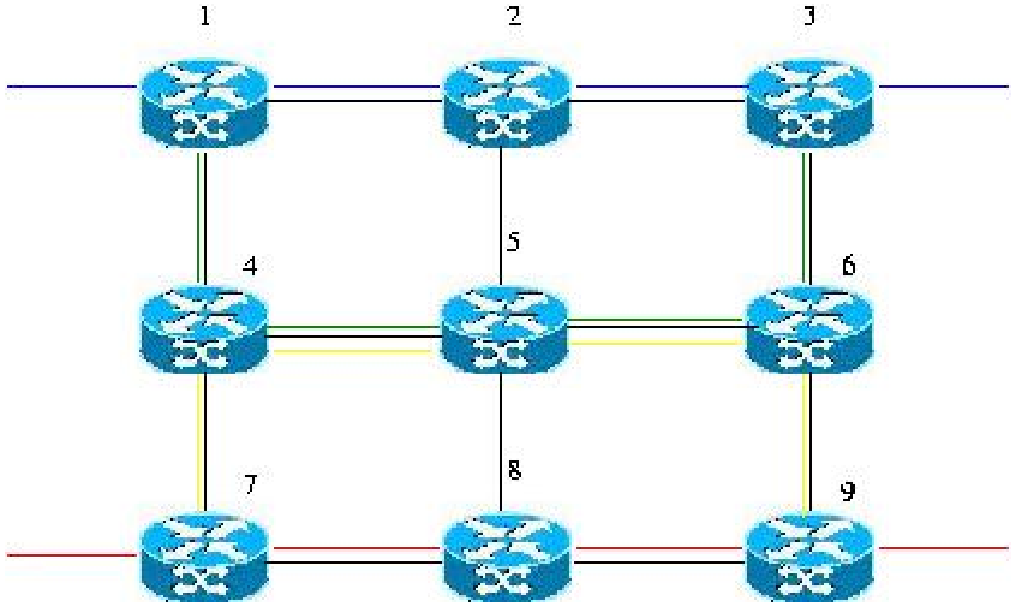


Figure 23 : Protection Sharing

The picture above shows a network with two working paths (1,2,3) and (7,8,9). The working path marked with blue lines has a recovery path setup with node protection for LSR 2 marked with green lines (1,4,5,6,3). The working path marked with red lines has setup a recovery path with node protection for LSR8, this recovery path is marked with yellow lines (7,4,5,6,9).

The two working paths use the same segments along links (4,5) and (5,6) for recovery. If we consider the possibility for multiple failure to occur at the same time in the network to be small, then the resources reserved along (4,5,6) can be shared between the two recovery paths. This is because these reservations protect different path segments, a failure on one working path would not affect the other working path. The single failure constraint is a reasonable assumption, because a backup path is likely to be used only for a short period of time until the failure is repaired or a new working path is established. Furthermore, most transport service providers do not plan restoration from multiple, simultaneous link or node failures because it would be expensive to protect against events that generally have a very low probability of occurrence.

The key observation for protection sharing is that the need for new backup recourse reservation depends on whether there is already sufficient protection capacity set aside to protect other connections, which does not belong to the same shared risk group (SRG) as a new working path. A SRG is a set of network elements that are collectively impacted by a specific fault or fault type. For example if a router goes down, all links connected to that router also goes down, so a router and all the links connected to this router are in the same SRG. Another example of a SRG is if two or more links use the same physical path, as if fibers are bundled together, then those links are in the same SRG for a cable cut.

The backup capacity reservations for a backup tunnel are often reserved at the same time that a working path is setup and activated. To share a segment of this recovery path for protection of another working path, two requirements must be met:

First, the links and nodes that the recovery path shall protect in the second working path can not be in the same SRG as the links and nodes that the shared segment protects in the first working path.

Second, either the bandwidth for backup that had previously been reserved along the recovery path is already sufficient to guarantee recovery from any single link or node failure along the protected segment for the second working path. Or there must be enough available bandwidth along the backup path to make an additional backup reservation.

If both these requirements are met, then shared protection can be used and the recourses reserved for recovery can be reduced in the network.

5.3 MPLS recovery models

In this section I will look at different proposed models for end-to-end recovery of LSPs in MPLS domains. End-to-end recovery means that a working path is fully protected for any link or node failure on that path.

5.3.1 Makam's Model

One of the first proposed models for MPLS recovery was presented by Makam in the draft [41]. The proposed model is often referred to as Makam's model. The model provides end-to-end protection for a LSP by setting up a global recovery path between the ingress and egress LSR. This recovery path is totally link and node disjoint with the working path. When a failure is detected anywhere along the working path, a fault indication signal (FIS) is used to convey information about the occurrence of the failure to the PSL. The PSL is then responsible for switching traffic over to the recovery path.

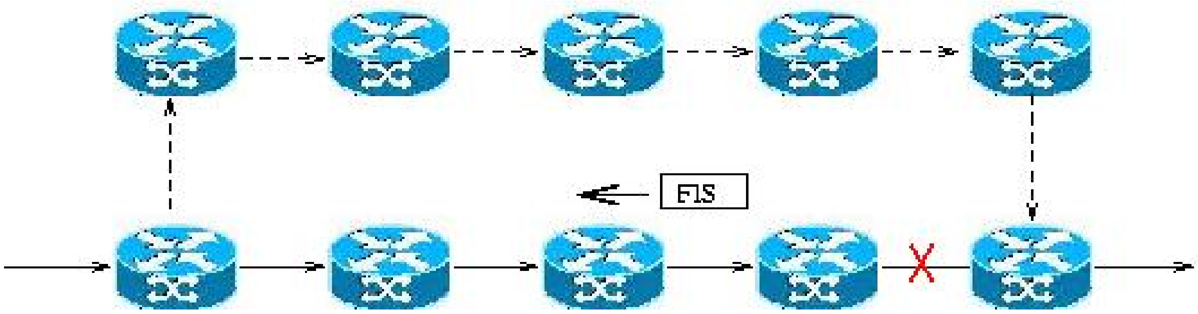


Figure 24 : Makam's Model

The above figure shows how the FIS is sent in the upstream direction of the working path when a failure is detected on the working path.

This model has proposals for both a pre setup (protection switched) recovery path and a dynamically established (rerouted) recovery path. As the dynamically established recovery path will add more time to the recovery operation the pre established path is the proposals mostly used with Makam's model.

As explained in the third chapter, a LSP is setup unidirectional and merging can be used so that more then one LSPs can be aggregated and therefore use the same segment of a LSP. When a LSR in the working path detects a failure, it must know which upstream LSR the FIS shall be sent to. To solve this problem a new concept called Reversed Notification Tree (RNT) was introduced.

5.3.2 Reverse Notification Tree (RNT)

Fault notification, once a failure is detected, can be local or global. In the case of local recovery the node that detects a failure is the node responsible for the recovery operation (PSL). But in the case of global protection, the node that detects the failure has to communicate from the point of failure to the PSL upstream in the working path. As LSP are setup unidirectional there has to be information of how the FIS shall be sent upstream. If merging of LSPs has been used then more then one PSL might need to be informed about the failure.

The Reverse Notification Tree is a proposal introduced in [14] to develop a notification tree in a global or segment protected environment using 1:1 protection. The reverse notification tree is a point to multipoint tree rooted at the PML along which a FIS can be sent to the PSLs affected by a failure. The RNT is a tree because with label merging, multiple working paths may have merged to form a multipoint to point tree, with the PSLs as the leaves.

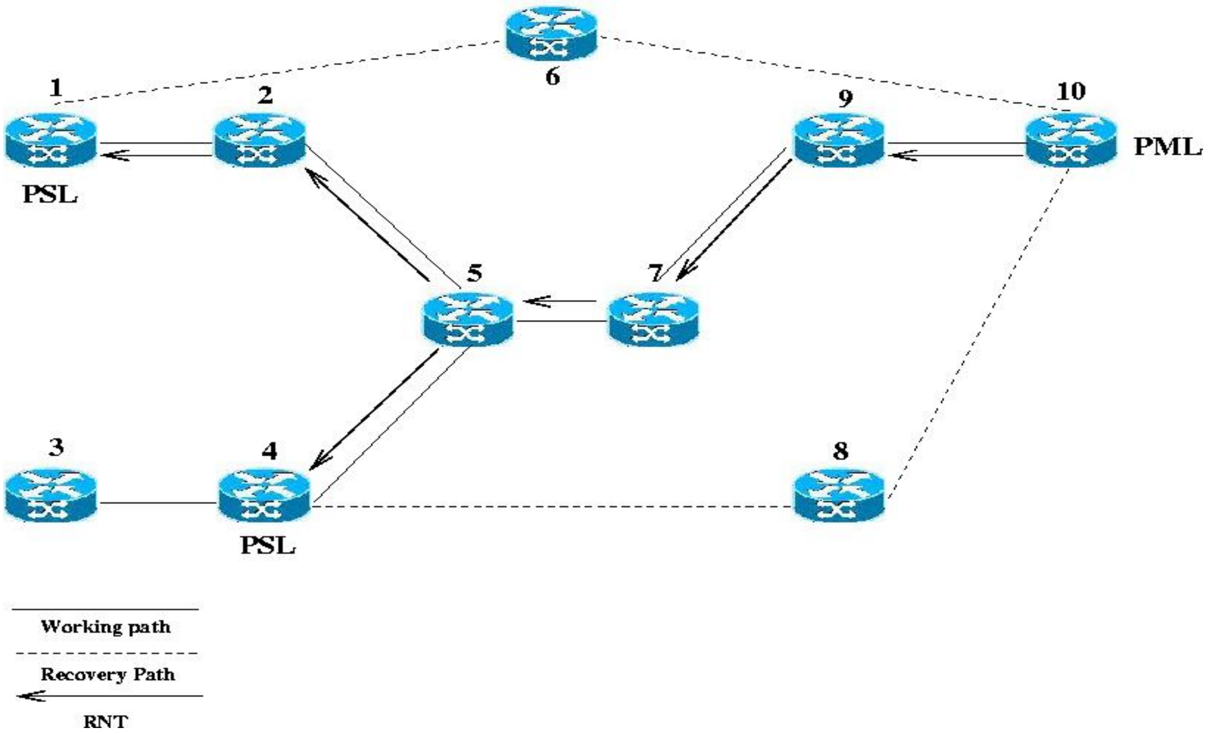


Figure 25 : Reverse Notification Tree

In the above figure the LSP 1 (1,2,5,7,9,10) is merged with the LSP 2 (3,4,5,7,9,10) at the path (5,7,9,10). PSL in LSP1 is LSR 1 and PSL in LSP 2 is LSR 4. The recovery path for LSP1 is set to (1,6,10) and the recovery path for LSP2 is set to (4,8,10). For both paths LSR 10 is set as PML.

If a failure occurs anywhere on the shared path from LSR 5 to LSR 10 the RNT shows how a FIS should be sent upstream on the paths. When LSR 5 receives a FIS from LSR 7 it has to send out two FIS, one to LSR 4 and one to LSR 2.

The RNT is an reversed image of the working paths along which a FIS can be sent, therefore signaling overhead can be reduced as only one FIS has to be sent on the shared segment of the working path where multiple working paths has been merged. The RNT is established in association with the working path, by making each LSR along a working path remember its upstream neighbor (or collection of upstream neighbors whose working path merge at the LSR). There is therefore no multicast routing required to set up a RNT.

Note that the RNT is not setup as a LSP, it setups forwarding information for how an incoming FIS should be forwarded in the reversed direction of an LSP.

5.3.3 Reverse Backup (Haskin's model)

When global recovery is used as in Makam's model, the PSL has to be informed about a failure in the working path before traffic can be switched over to the recovery path. When this is done with a FIS, traffic will be sent down the failed working path until this FIS has been received by the PSL. This will result in dropped packages at the LSR that is upstream of the failure, as this node does not have any forwarding information for these packages since the downstream node is not reachable. If the failure is situated far away from the point of repair and the transmission rate is high, the number of packets dropped can be very high.

The idea of reverse backup is to reverse traffic at the point of failure in the working path, back to the PSL (ingress LSP). As soon as a LSR detects a failure on the working path, it redirects the incoming traffic on to an alternative LSP that is setup in the reverse direction of the working path. When the reversed traffic reaches the PSL, it forwards this traffic on to a global protection path. Both the reverse path and the global protection path are pre reserved.
This scheme was introduced by Haskin in the draft [34] and is therefore often called Haskin's model.

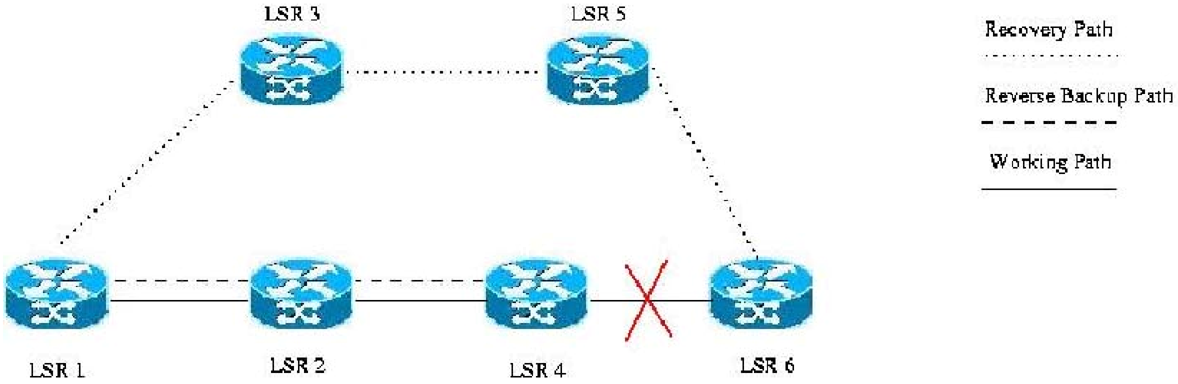


Figure 26 : Haskin's Model

The above figure shows an example of reverse backup. The working path and recovery path are established as in global protection, in addition there is a reverse path from LSR 4 to the ingress node LSR 1. When a failure is detected by LSR 4, the traffic is switched back to the ingress LSR 1 through the reverse backup LSP and then switched onto the global recovery path. As we can see this model protects against failures anywhere on the working path between LSR1 to LSR 6. This is much like the recovery ideas in SONET/SDH BLSR as described earlier in this thesis. The similarity is that a ring is constructed in the network and traffic is forwarded in the reversed direction of the ring if a failure occurs.

There is also the option to make the PSL (LSR 1) forward incoming traffic directly onto the global protection path as soon as it notices that the reverse backup path is in use. In this way the reversed traffic is used as an FIS, when the first package is received from the reversed path the PSL starts to forward traffic directly onto the global backup path instead of first down the failed working path. In this way the backup path gets shorter, because there seems to be no need to keep on forwarding traffic down a broken working path, if it is switched back to the sender and then on to the recovery path. This optimization has further similarities to a model explained earlier in the chapter about network recovery in other layers, and makes the mechanism look more like the techniques in RPR.

This optimization has a disadvantage, until the PSL receives any of the reversed traffic, packages will be forwarded on the broken working path. When the PSL receives traffic from the reversed path, it will start to forward incoming traffic onto the global backup path. For a short period the incoming traffic will be mixed with the reversed traffic as it is forwarded on the recovery path. This reordering of packets can cause problem with higher layer protocols that needs packets to arrive in order.

A different concern for Haskin's model is the less efficient use of resources, as the total length of the recovery path gets longer then the original working path. The positive side of this model is that the number of packages dropped when a failure occurs, can be decreased as no FIS is needed when traffic from the reverse backup path acts as FIS for the PSL. Traffic can be switched onto an alternative path by protection switching directly when a failure is detected.

In this model both 1:1 protection and 1:N protection can be achieved.

5.3.4 Hundessa's Model

As explained earlier, Haskin's model can recover from a failure with less packet loss then Makam's model. With the optimization of Haskins model, some unnecessary recourse usage can be freed when traffic is forwarded directly onto the global recovery path when reversed traffic is discovered by the PSL. The two way delay for packets that are sent along the broken working path and then returned to the PSL before the PSL is aware of the failure, will then results in reordering of packets. This is because traffic from the reversed path and new traffic received from the source are mixed when it is forwarded on the global recovery path in the first part of the optimization.

In [42] Hundessa presents a model to improve the optimization process in Haskin's model. In this proposal, when a failure is detected by a LSR, the packets that would have been forwarded on the failed path are returned to the PSL via a reversed backup path as in Haskin's model. But when the first packet, acting as a FIS arrives in the reverse direction at an upstream LSR, that LSR tags the next packet it sends out on the working path. The next packets it receives that belong to the same working path are buffered. The packets are kept in the buffer until the tagged packet is received from the reversed backup path, and then forwarded on the reverse backup path. When the last packet that was sent out by the PSL on the failed working path is received from the reversed path, all packages are forwarded directly onto the global recovery path. In this way reordering of packets are prevented and there is less unnecessary forwarding of packets along the broken working path.

The last packet that an LSR sends out on the broken working path is tagged by setting a bit in the EXP field in the MPLS header.

5.3.5 Fast Reroute

One of the largest motivations for MPLS recovery is the ability to ensure recovery from a link or node failure with minimal disruption to the data traffic. Often the limit of 50 ms is considered because this has been set as the longest acceptable disruption to voice traffic and is used as a limit for recovery time in SONET/SDH networks.
If recovery by re-routing is used we have seen that the recovery time depend on the time to discover the fault, the time to notify the PSL of the failure, the time to calculate a recovery path (if it is not pre-calculated) and the time for a new recovery path to be set up. This can be slow and can take up to several seconds which is unacceptable for many real time applications.
If recovery is done by protection switching, the recovery time can be decreased because recovery path calculations are not needed. When the recovery path is pre-established there is no need to signal the recovery path and recovery time then only depends on fault detection time and the time for the FIS to travel to the PSL.

The technique to have pre-set-up recovery paths that uses local repair is often called fast re-route. The idea is that recovery time will be fast if there is no need for signaling and if the PSL is the first node to detect a failure. This means that fast reroute is used as a local repair mechanism that uses link or node protection by establishing a recovery path that uses protection switching to detour the failed link or node. If fast reroute shall be used end-to-end then recovery paths needs to be pre-setup for each link or node in the working path. In [46] extensions to the RSVP-TE protocol are defined to establish an LSP with end-to-end fast reroute backup tunnels. Two techniques are described in the draft the one-to-one backup model and the facility backup model.

5.3.6 Fast Reroute one-to-one backup

In the fast reroute one-to-one technique a separate backup LSP, called a detour LSP is computed for each LSR in a protected path. These detour LSPs are set up to use node recovery if possible otherwise link recovery. To fully protect an LSP that traverses N nodes, there could be as many as N-1 detours.

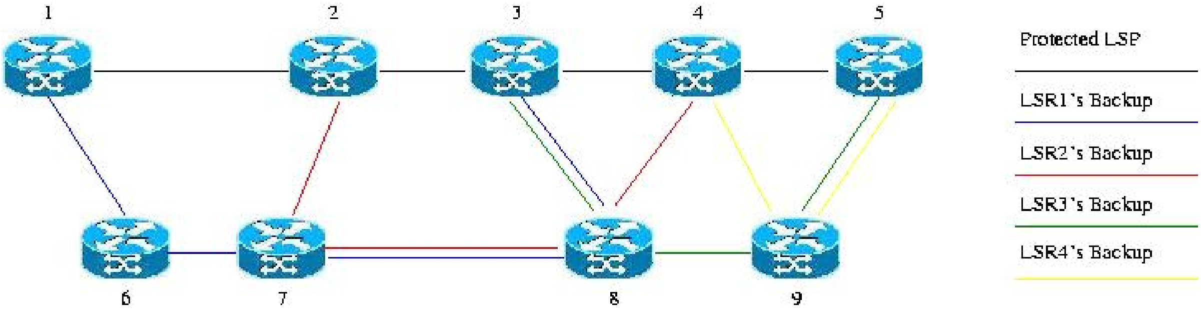


Figure 27 : Fast Reroute one-to-one

In the figure above the protected LSP (1,2,3,4,5) has 4 detour LSPs. The detours 1-3 use node protection, while the 4:th detour use link protection. If a failure occurs anywhere along the protected path, the LSP that detects the failure can always switch traffic onto a local detour without the need to send a FIS upstream before recovery can start. Therefore the recovery operation becomes a local decision for the LSP that detects the failure. The detours are setup to use splicing so the depth of the label stack does not increase when a detour is used. The use of splicing makes the recovery path only available for the specific LSP that it is setup to protect. In this model there are some extensions to the RSVP-TE protocol suggested, that makes it possible for those detours to be found and setup automatically at the setup time of the working path.

As can be seen in the above picture, the use of local protection for each LSR in the working path keeps a lot of resources reserved for backup purposes. Although there is only one working path in this example network, there are some links where two reservations are made for backup purposes. This is not necessary as there is only one working path and both of those reservations can not be used at the same time if a single failure occurs on the working path. This concerns the two reservations LSR7 to LSR 8 and LSR9 to LSR 5, and not the reservation between LSR3 and LSR 8 as these reservations are made in opposite directions. To prevent this double reservation, merging can be used to share those reservations. Merging can only be done if the two ore more reservations are backup reservations for the same working path with the same next downstream hop in the detour path. The following rules are used for merging of reservations:

For all reservations that share the same outgoing interface and next-hop LSR, the reservation shall only be done on the path that makes that detour the shortest path to the egress of the protected LSP. If the paths have the same hop count, the path that traverses nodes that others want to avoid should be eliminated. If there still is more then one detour available it is a local decision which one that will be chosen.

In the above example this means that on the link from LSR7 to LSR 8, the LSR1’s backup will be merged with LSR2’s at LSR7. This is because the second backup has a shorter path to the egress LSR (8,4,5 instead of 8,3,4,5).
LSR9 will also merge the reservations from LSR3 and LSR4 on the link from LSR8 to LSR 9, in this case the hop count to the egress is 1 for both detours. The detour from LSR3 is setup to bypass LSR4, so the detour from LSR3 will make the final reservation. In this case the reservations made are illustrated in the picture below.

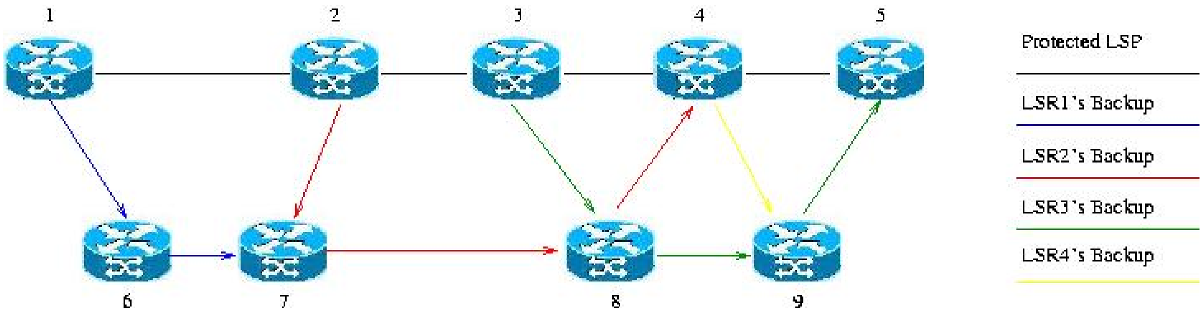


Figure 28 : Fast Reroute one-to-one with merging

The path from LSR7 to LSR4 is now used by both LSR1’s and LSR2’s backup and the path from LSR9 to LSR5 is used by both LSR3’s and LSR4’s backup.

With this merging technique we can se that the original twelve reservations have been reduced to nine.

5.3.7 Fast reroute Facility Backup

In the fast reroute facility backup model a single LSP is created which is used to backup a set of LSPs instead of using a separate detour for every protected LSP. This constrains the set of LSPs being protected via the backup tunnel to those that pass through a common downstream LSR. All LSPs that pass through a point of repair and through this common LSR can be protected by the bypass tunnel. This means that the facility backup model use n-to-1 path mapping as described earlier in this thesis (section 5.2.5).
The tunnel uses label stacking, because when multiple LSPs are protected by one backup tunnel there has to be a way to separate the different LSPs again when traffic from the backup tunnel arrives at the PML.

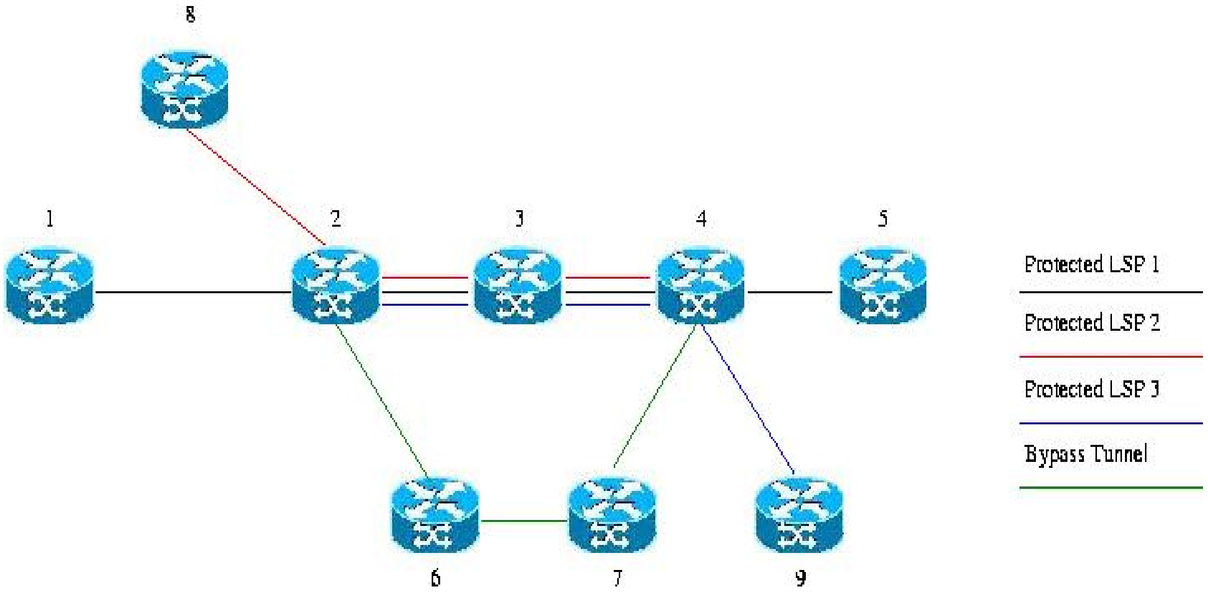


Figure 29 : Fast Reroute Facility Model

In the figure above there are three LSPs that are protected by one backup tunnel. As we se in this picture the facility backup tunnel, use local recovery with node protection. In this case LSR 3 is node protected for the three protected LSPs. If the one-to-one technique should have been used, there would have been three detours set up to node protect LSR 3, one for each protected LSP. Observe that the facility technique not is used for protection sharing. Even though multiple LSPs can share the same bypass tunnel, resources are not shared. Each protected LSP has to make an additional reservation if it shall use the tunnel. The facility technique is therefore used to mineralize the control traffic.

5.3.8 Modifications for shared backup protection.

To use shared protection as explained in section 5.2.6, some modifications have to be made to the signaling protocol that establishes the recovery path. As explained earlier sharing can be done if a previously established recovery path and a new recovery path share a segment of a path that is used for recovery, and the LSRs and links that this path protects in the two different working paths are not in the same SRG.

This means that when a recovery path is signaled a check needs to be performed at each LSR in the recovery path, to check if any previously reserved resources can be shared.
To be able to do this check, information about which working path a recovery path is setup to protect needs to be saved at each LSR in the recovery path.

When a LSP is setup with RSVP-TE the router ids in this LSP are collected by the Record Route Object (RRO) carried by the PATH and RESV messages. When the recovery path is setup for this LSP, this router id list needs to be included with the message that is sent to setup the recovery path. If this router id list is saved locally at each LSR in the recovery path, then the SRG check can be performed when a new recovery path is setup on that same segment.

Not much research has been done about sharing of resources for protection switching in MPLS. In [55] it is pointed out that it might not be enough to have only the node ids of the LSRs that a recovery path is set to protect. In this proposal a new object for RSVP-TE is introduced that also records the outgoing interface for each LSR in the working path. This might be needed to not only identify the nodes in the working path, but also the links that the recovery path is set to protect.

While some sharing of protection bandwidth can be realized with extensions to RSVP-TE, the amount of bandwidth reserved for restoration is best effort and may not be minimized. This is because the sharing potential is not known before the recovery path selection is done by the PSL, which does not know about the sharing potential in the possible recovery paths. The PSL choose a recovery path on other criteria as link weight and shortest path as in the setup of the working path. In this case it is possible to use extensions to RSVP-TE to locally compute, at every LSR along the recovery path, the amount of shared protection bandwidth required on each of the links in the backup path.

If the PSL shall be able to select the optimal backup path for shared backup protection it has to have a database that extends the link state database used to find paths in the network. This database needs to have entries for each of its links that contains information (bandwidth, link-id) about all the links in the network for which a link provides restoration capacity. In that way a backup path can also be selected on the criteria to use the least amount of new reserved bandwidth.

A proposal for extensions to the interior routing protocol OSPF-TE that enables for least amount of new reserved bandwidth calculations is presented in [56].

For en example of how the algorithm presented in this proposal can be used, se appendix A.

6. Simulation and analysis

In this chapter different simulation environments will be presented, and the simulations that I have performed will be presented and analysed.

6.1 Simulation environments

Before I decided which simulator environment I would use in this thesis, I looked at some different available network simulators. In this section I will give an overview of the available networks simulators with MPLS implemented, and describe why I choose ns-2 as simulator environment in this thesis.

6.1.1 J-Sim

J-Sim [57] (formally known as Java Sim) is a component based network simulator developed entirely in Java, developed by Hung-ying Tyan and some other people at the Ohio State University. J-Sim is an open source project and the simulator and some modules are available for download at the projects website.

There has been one MPLS module contributed to this simulator, developed by the Infonet Group of the University of Namur [58].

This model consists of two components: a forwarding table component and a MPLS component. The forwarding table component keeps all information about configured labels. It associates an IP prefix or an incoming label with an outgoing interface and an outgoing label. The MPLS component forwards packets according to the configuration of the forwarding table. This model does not include any label distribution protocol so LSPs must be setup statically. There was supposed to be an RSVP-TE implementation added to the project but this work has not been finished. Because of the lack of a signalling protocol I decided not to use this simulation model in my thesis.

6.1.2 OMNeT++

OMNeT++ [59] is a discrete event simulation environment programmed in C++ and developed by András Varga. The simulator is open source and free to use for academic and non-profit users. The simulator is component-based and there are many models in development for this simulator.

The MPLS model [60] in OMNeT++ was originally developed by Xuan Thang Nguyen from the University of Technology in Sydney. Now the model is maintained by András Varga [61]. The model includes components for MPLS forwarding, LDP, CR-LDP and RSVP-TE.

This model was developed to study the forwarding mechanisms in MPLS so there where some simplification done in the way that the Label Switched Routers was implemented. This made this model as it is today not suitable for failure recovery simulation. The forwarding tables for all routers in the network is implemented in one table, which means that all routers have the same copy of the network topology at the same time. If a failure happens in the network then all routers will be aware of the failure at the same time. This is not what happens in a real case scenario where topology changes like a failure has to be indicated to nodes by sending routing update information from the point of failure. There were also some problems with the implemented transport protocols for IP (TCP and UDP) at the time I tried to use this model. This looked like a promising simulator to use in my simulations because RSVP-TE was implemented, but after some testing of the included MPLS model and the transport protocols, I noticed that a lot of the implemented functionality in the model had to be rewritten. Because of these problems I decided not to use this model.

6.1.3 GLASS

GMPLS Lightwave Agile Switching Simulator, short GLASS, is a Java based network simulator [62]. This simulator is developed by the Internetworking Technologies Group of the Advance Network Technologies Division at NIST (National Institute of Standards and Technology). The simulator uses the basic framework of the Scalable Simulation Framework (SSF) with its extension SSFNet [63]. Although this is as the name implies a simulator that is developed for GMPLS simulations, it can be used for MPLS simulations as well. The simulator has implemented MPLS forwarding and label distribution with LDP, CR-LDP and RSVP-TE.

After some testing I found this simulator to be highly unstable. I tried to come in contact with the developers of this simulator, but never received any answer, so this simulator doesn’t seem to be in development anymore. Because of the instability of this simulator and that I couldn’t receive any information of why the simulator behaved as it did, I decided not to use this simulator.

6.1.4 NS-2

NS-2 (Network Simulator 2) is a discrete event simulator targeted at networking research [64]. The simulator has been in development since 1989 and there are many different contributed models available for the simulator. This simulator is also open source and there are a lot of users that has contributed to make this simulator. The main implementation is made by the VINT project at BL, Xerox PARC, UCB, and USC/ISI. NS-2 is coded in C++ and TCL.

When I first looked at this simulator there was one contributed MPLS module for NS-2. This module is called MNS (MPLS Network Simulator) and was developed by Gaeil Ahn [65]. This model had implemented MPLS forwarding and label distribution by LDP and CR-LDP. The model has also some functionality implemented for recovery mechanisms, like the ability to setup a backup path and associate it with a working path. Unfortunately this model did not include an implementation of RSVP-TE.

Because I wanted to use RSVP-TE in my simulations as this is the most used label distribution protocol, I thought about making my own model for RSVP-TE and include this in that model. But it turned out that others had the same idea.

René Böringer at IDEO Laboratories [66] extended MNS by adding Marc Greis' RSVP model [67] and modified this model to add TE abilities. This model added the abilities I needed to make my simulations, but there were some problems with the model. The routing protocols implemented in NS-2 were not compatible with the new model. Therefore no routing mechanism could be used and all LSP had to be setup explicitly.

Christian Callegari and Fabio Vitucci [68] have also made an implementation of RSVP-TE for NS-2 by extending Marc Greis' RSVP model. This model did not have the same problem with the implemented routing protocol.

I chose to use NS-2 with MNS and the RSVP-TE path by C.Callegari and F.Vitucci because NS-2 is the most used open source simulator available. MNS had almost all the implemented functionality I needed and the RSVP-TE patch made it possible to use RSVP-TE in my simulations.

There are also some commercial simulator environments available for MPLS simulations were OPNET probably is the most know simulator. Because there was no license for any commercial simulator at my university, I could not use any commercial simulator.

6.1.5 Installation of MNS with RSVP-TE for NS-2

NS-2 has been made in many versions and the one used in this thesis is version 2.26.

To install NS-2 it is recommended to use the ns-allinone distribution that includes all programs that NS-2 uses, such as xgraph, nam, tk and tcl. NS-2.26-allinone can be downloaded from [64].

The RSVP-TE model for NS-2 is made for NS-2 version 2.26. To use the model, MNS for NS-2.26 has to be installed before the RSVP-TE patch is applied.
MNS for NS-2.26 is not included in the main distribution of NS-2.26 and is not available for download from the author's webpage, since he is not working on the implementation anymore. I therefore made MNS for NS-2.26 available for download from my webpage [69].

The RSVP-TE patch for MNS 2.26 is available from [68]. There is also an installation instruction available on this web page that explains how to install the patch.

6.1.6 Extending the RSVP-TE patch

I wanted to do some simulations with failure detection by the RSVP-TE hello mechanism. Because this mechanism was not implemented in the patch, I had to implement the mechanism myself. My implementation of the patch is now included in the patch available from [68].

The implementation of the mechanism follows the standard in RFC3209. It makes it possible to activate the mechanism between any chosen RSVP-TE agents or all RSVP-TE agents. It is possible to set the hello interval for a separate hello session and to set the failure detection interval for this session. When a failure is detected the LIB (Label information table) is updated for all LSPs that are affected by the failure. There is also the possibility that a FIS can be sent to a configured node when a failure is detected.

To implement the hello mechanism in ns-2 I had to make a new RSVP object. This object follows the standard in RFC3209, it has the class number 22 and is called the HELLO object. The object is composed of two fields, the src_inst and dst_inst fields.
These fields are used to carry the 32 bit source and destination instance values.

I also needed to define a new message class type, called the HELLO class. This class has two different C_Types, 1 and 2. The C_Type 1 is used to identify the HELLO message as a REQUEST message and the C_Type 2 is used to identify the HELLO message as an ACK message.

I also needed to define this message type for the RSVP-TE agents so that they recognized the message as a RSVP-TE message. Otherwise the RSVP-TE agent would drop the message when the message type isn't recognized. This message type is set to 20 as in the RFC3209 and the protocol number is set to 46 which is the protocol number for RSVP-TE.

To find which LSPs that are affected of a failure, the path state block (PSB) in the node that detected the error is used. The PSB holds information about the LSP id and the explicit route for this id. This information is used to update the forwarding table (LIB) correctly for the node that detects the failure.

To implement this new functionality to ns-2 I had to make some changes to the existing RSVP-TE implementation and add some new functions. The source code for some of the changes made to ns-2 can be found in Appendix C.

I also contributed to the RSVP-TE patch with some bug fixes concerning the setup of LSPs that are now included in the patch.

6.2 A closer look at the RSVP-TE HELLO mechanism

As described in section 5.1.1, the RSVP-TE failure detection mechanism can be used to detect both node and link failures. The failure detection time for the mechanism depends on two values, the hello interval and the failure detection interval. The hello interval is used to set how often a RSVP-TE agent shall send out a new REQUEST message, the default for this interval is 5ms. If no Instance values are received from a neighbor within a configured number of hello intervals, then the requesting node presumes that it cannot communicate with its neighbor. The default for this number is 3.5.

The default failure detection interval is therefore set to 5 * 3.5 = 17.5ms. This means that the mechanism will check every 17.5 ms to se if it has received any ACK since the last check. If no ACK has been received during this interval, then the mechanism will interpret this as if a failure has occurred.

It could therefore be easily misunderstood that this mechanism should be able to detect failures within 17.5 ms with the default values. This is not true, the failure detection time depends on these values and when the failure occurs. The following flow charts show the scenarios for the fastest and slowest possible failure detection time with the hello mechanism.

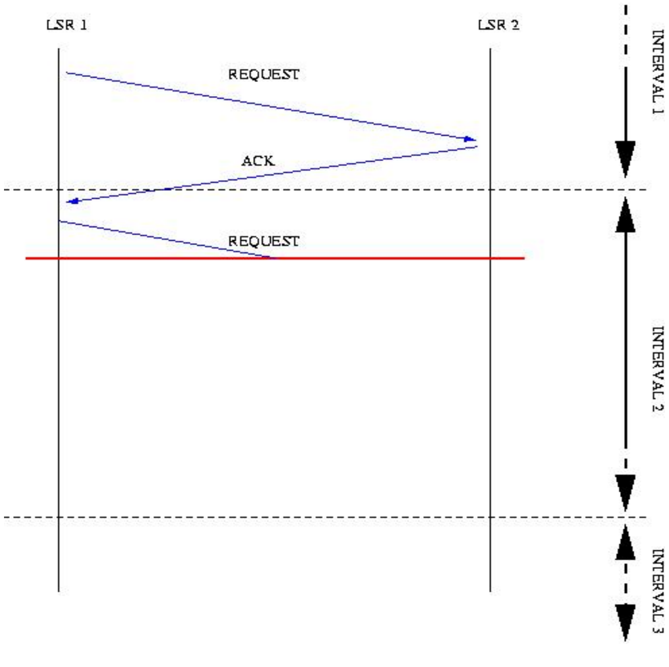


Figure 30 : Failure detection, worst case.

The chart above shows how a failure will be detected in the worst case concerning failure detection time. The chart show the messages used by LSR1 to detect a failure to its adjacent neighbour LSR2. The red line illustrates when the failure occurs and the blue arrows shows how messages are passed between the two nodes LSR1 and LSR2. The black arrows show when the failure detection interval times out and LSR1 performs a failure check.

In this case, a failure check is done just before an ACK message is received. Right after this failure check, the ACK is received and then the link between LSR1 and LSR2 breaks. When the next failure detection interval times out and a new check is performed, this check will not detect the failure because an ACK was received during this interval. When the next failure check is performed, at the end of interval 3 the failure will be detected. This shows that the actual failure detection time can be almost twice the size of the failure detection interval.

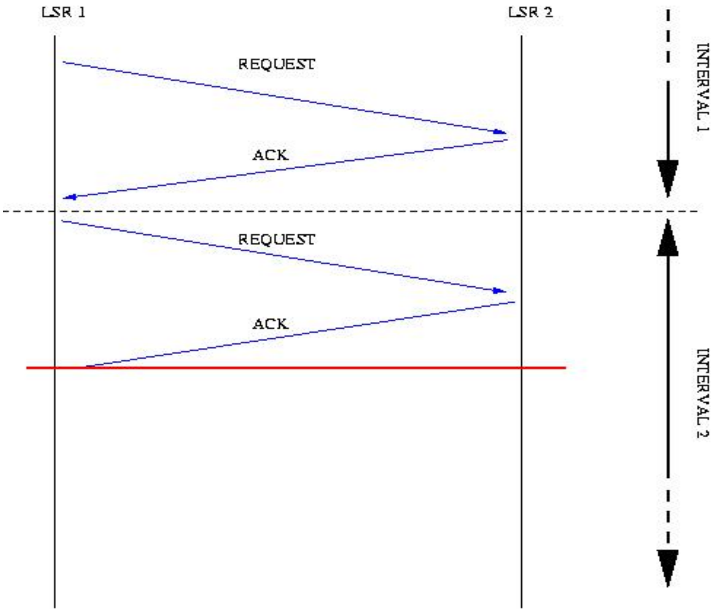


Figure 31 : Failure detection, best case

The above chart shows the case where the failure detection time is as fast as possible. In this case a failure check is performed just after an ACK is received. The failure occurs just before the next ACK should have been received by LSR1. In this case the failure is detected by the fail check at the end of interval 2. The failure detection time is therefore smaller then the failure detection interval.

This means that the fastest time that the mechanism can detect a failure is approximately: Failure detection interval – Hello interval. And the largest failure detection time is approximately: 2 * Failure detection interval. This can be setup as:

Ft: Failure detection time.
Fi: Failure detection interval.
Hi: Hello interval.

Fi – Hi < Ft < 2 * Fi

When the hello interval is set to a small value so that a failure can be detected fast, it might look like the mechanism would create a lot of control traffic. For the default hallo interval value of 5ms from RFC3209, the extra amount of control traffic can be calculated.

A send rate of 5 ms will generate 1 / 0.005 = 200 REQUEST packets per second.

The HELLO package consists of the following information: A common header of 8 bytes and the HELLO object contents of 8 bytes. This makes the size of the package 16 bytes + the encapsulation for the other layers. If IP version 4 is used, this means an addition of 20 bytes for the IP header. Depending of what link layer that is used, an addition for the link layer header of at least 12 bytes is added. This makes a minimum package size of 16+20+12 = 48 bytes for the HELLO package.

At 200 packets per second this makes 200 * 48 * 8 = 76.8Kbps. On a 10Mbps link this makes 0.768 % of the link capacity.

If both sides of a link use the mechanism, then ACKs also needs to be generated. This might increase the generated traffic to 2 * 200 * 48 * 8 = 153.6Kbps.

But an optimization can be done, because the REQUEST message also includes the instance value for the sender. If both sides have activated the hello mechanism, then a REQUEST message can act as an ACK message. This means that if a node has received a REQUEST message from its neighbour in the hello interval, then it does not need to send a new REQUEST to this neighbour in this interval, because it has already received the neighbours instance value in that interval. With this optimization the number of hello messages between two neighbours that has activated the hello mechanism can be decreased.

6.3 Simulations of MPLS recovery

The network topology and settings used in the following simulations are the same for all simulated cases. The picture below shows the network topology and how traffic flows on the working path in normal operation. The ns-2 allinone package includes a network viewer called nam (short for Network Animation) that can visualise the events in the network from trace files created by the ns-2 simulation. All network pictures in this chapter are screen shots from my simulations using the nam viewer.

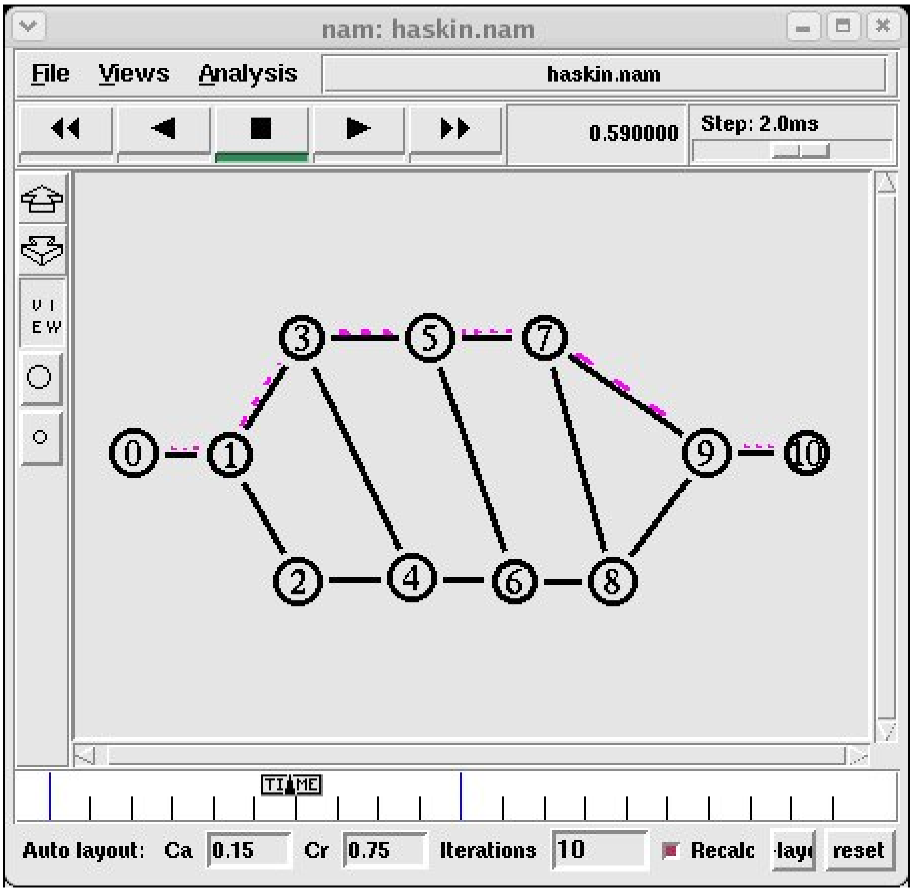


Figure 32 : The network topology shown in Nam

The network consists of nine MPLS enabled nodes (1-9). Each MPLS node has an RSVP-TE agent attached that is used for label distribution. The nodes 0 and 10 are of the standard node class in ns-2 and have no MPLS module or RSVP-TE agent attached.

Node 0 has a CBR (constant bit rate) UDP agent attached and node 10 is set up to be traffic sink. At time 0.5s the CBR/UDP agent starts to send an UDP stream of packets that has destination address set to node 10. The size of the packets is set to 200 bytes and the send rate is set to 5000kbps. At time 1.8s the agent at node 0 stops the UDP stream towards node 10.

The working path is set to the shortest path between node 0 and node 10 (1,3,5,7,9). This LSP is setup with RSVP-TE signalling before the agent at node 0 starts to send traffic, the lspid for the working path is set to 1000.

The propagation delay between two nodes is set to 1ms and the link bandwidth is set to 10MB.

At time 0.8s a link on the working path breaks. When the upstream LSR on the working path detects the link break, the chosen recovery mechanism is started. In the following simulations it is the link between LSR5 and LSR7 that breaks, but simulations has been done for every possible single link break on the working path, for each simulation model. Each simulated model is setup to use end-to-end recovery, so the models can recover from a single link break anywhere on the working path. The models can also recover from node failures anywhere on the working path, apart from on the ingress or egress LSR.

For failure detection the hello mechanism is used. The mechanism is activated for all nodes that have a RSVP-TE agent attached and start time it set to 0.1s. The hello interval is set to 5ms and the multiplier set for the failure detection interval is set to 3.5. So a failure check will be performed in 17.5ms intervals.

In the simulations that use rerouting, the time to perform a SPF calculation is set to 2ms. The algorithm used by most interior routing protocols for a SPF calculation is a version of Dijkstra's algorithm that has a complexity of *O*(E log N) , where e is the number of edges in the network and n is the number of nodes. This calculation can take 1 - 40ms to perform depending on the efficiency of the implementation, the processing load of a router's control-plane, and the number of nodes in the network. The time used in these simulations is an estimate found from the OSPF testing done in [70].

Each LSR is setup to drop traffic if it does not have a LSP binding for incoming traffic or if no backup resource can be activated if a link breaks.

RSVP-TE control traffic is set to a higher priority then other traffic in the network, and 1% of the link bandwidth is reserved for this traffic.

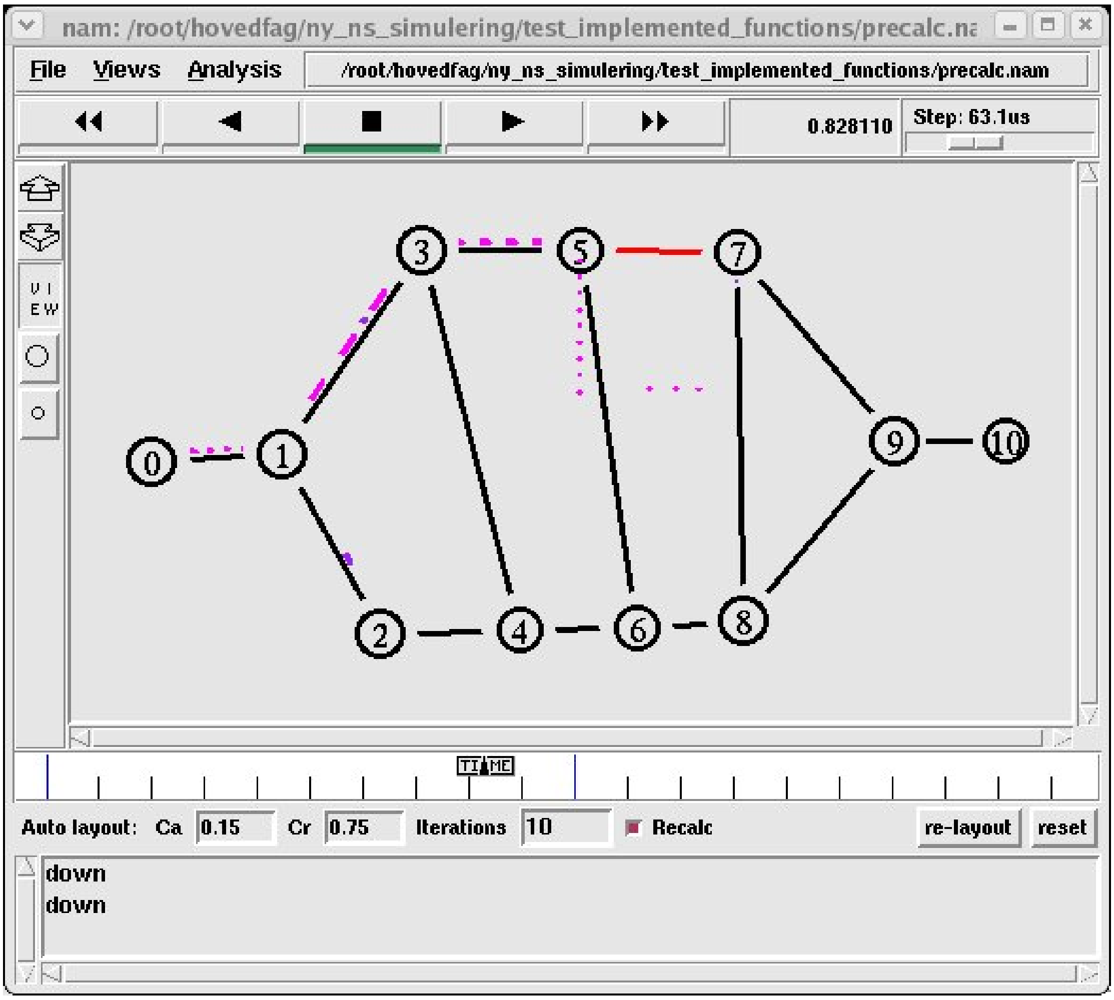
For each simulation I collect data for how many packets that are dropped when the link breaks. As the settings for the hello mechanism are set to the same values for all simulations, the packets dropped during the failure detection interval will be the same in all simulations.

I also collect information about the service disruption time. In these simulations the service disruption time is defined as the time from when the last package that that was sent over the link before the break, is received by the sink at node 10, until the next package is received by this node. This information is achieved from the trace file made by ns-2 that record the time for each event in the simulations model.

For each simulation I also look at how many resources that are used for backup purposes in the simulated model (bandwidth and buffer reserved). This number represents the resources that are reserved for backup purposes, which are not used until the simulated recovery mechanism starts.

6.3.1 Global protection with rerouting (best effort protection)

In this model the default mechanism for protection in an MPLS domain where LSPs are setup with RSVP-TE is simulated. The ingress LSR is setup as PSL and the egress LSR is setup as PML. When a failure is detected on the working path a PATHErr message is sent from the point of failure downstream to the ingress node (LSR1). The PATHErr message is used as an FIS and when it is received by the ingress LSR a new LSP that is link disjoint with the previous working path is calculated. Then this new path is setup with RSVP-TE on (1,2,4,6,8,9). The previous working path is torn down.


nam: /root/hovedfag/ny_ns_simulering/test_implemented_functions/precalc.nε
File
Views
Analysis
/root/hovedfag/ny_ns_simulering/test_implemented_functions/precalc.nam
0.828110
Step: 63.1us
VIEW
0
1
2
3
4
5
6
7
8
9
10
TIME
Auto layout:
Recalc
re-layout
reset

Ca 0.15 Cr 0.75 Iterations 10

down
down

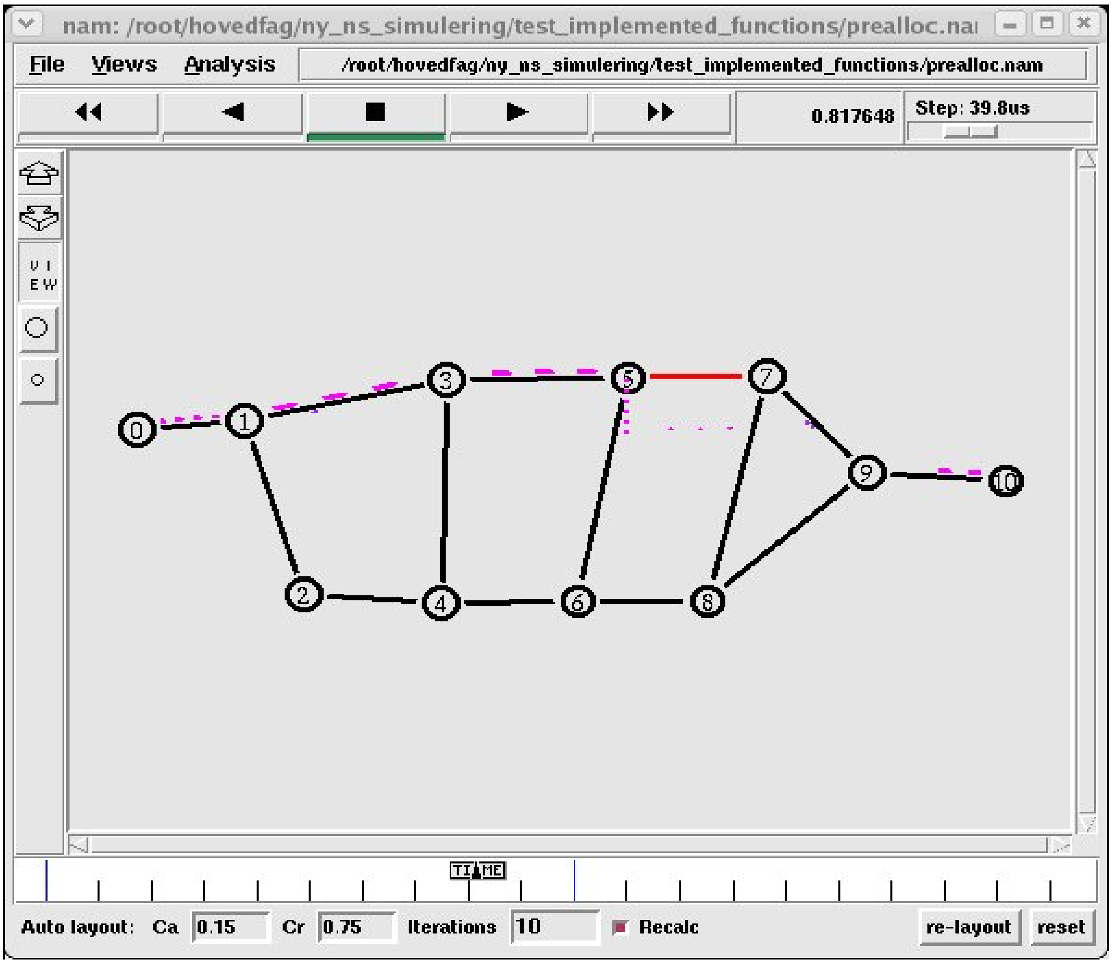
Figure 33 : Best Effort Model in Nam

The above picture shows the network after the failure has occurred. On the link between LSR1 and LSR2 the PATH message used to setup the new LSP is seen, and on the link between LSR1 and LSR3 the RESVTear message is shown. As traffic is forwarded on the working path until the new LSP is setup, traffic is dropped at LSR5.

The service disruption time at node 10 is calculated to: 0.83972s - 0.80232s = 0.0374s
The reserved resources used for backup is: 0
Total dropped packets: 112

6.3.2 Makam’s model (Global recovery with protection switching)

In this model a recovery path is setup from the ingress LSR1 to the egress LSR9 before traffic is sent in the network. This recovery path is setup with global protection and established on the path (1,2,4,6,8,9).
When the link between LSR5 and LSR7 breaks a PATHErr message is sent from LSR5 upstream to the ingress node. This PATHErr message is treated as a FIS and when LSR1 receives this message, the forwarding table at this node is updated to forward traffic on the recovery path. The old working path is torn down.



nam: /root/hovedfag/ny_ns_simulering/test_implemented_functions/prealloc.nai

/root/hovedfag/ny_ns_simulering/test_implemented_functions/prealloc.nam

0.817648

Step: 39.8us

Auto layout: Ca 0.15 Cr 0.75 Iterations 10

Figure 34 : Makam’s Model in Nam

In the above picture the PATHErr message is sent from LSR3 to the ingress LSR1. When LSR1 receives the FIS it will start to forward traffic on the recovery path (1,2,4,6,8,9). All packets that were sent on the working path after the failure occurred until the PATHErr has reached LSR1 will be dropped by LSR5.

The service disruption time at node 10 is calculated to: 0.82687s - 0.80232s = 0.02455s
The reserved resources used for backup is: 5
Total dropped packets: 72

6.3.3 Haskin’s model.

To simulate Haskins model, two backup paths are setup in the network. One LSP is setup in the reverse direction of the working path (7,5,3,1) and a global recovery path that is link disjoint with the working path is setup between the ingress and egress LSR (1,2,4,6,8,9). These paths are setup before traffic is sent in the network.

The LSRs 7, 5 and 3 are setup to forward traffic onto the reverse path if a link failure is detected on the working path. LSR 1 is setup to forward traffic from node 0 onto the working path if traffic is received on the interface from node 0, if it receives traffic on the interface from node 3 with the same flow id as traffic that belongs to the working path, it is setup to forward this traffic on the global recovery path.

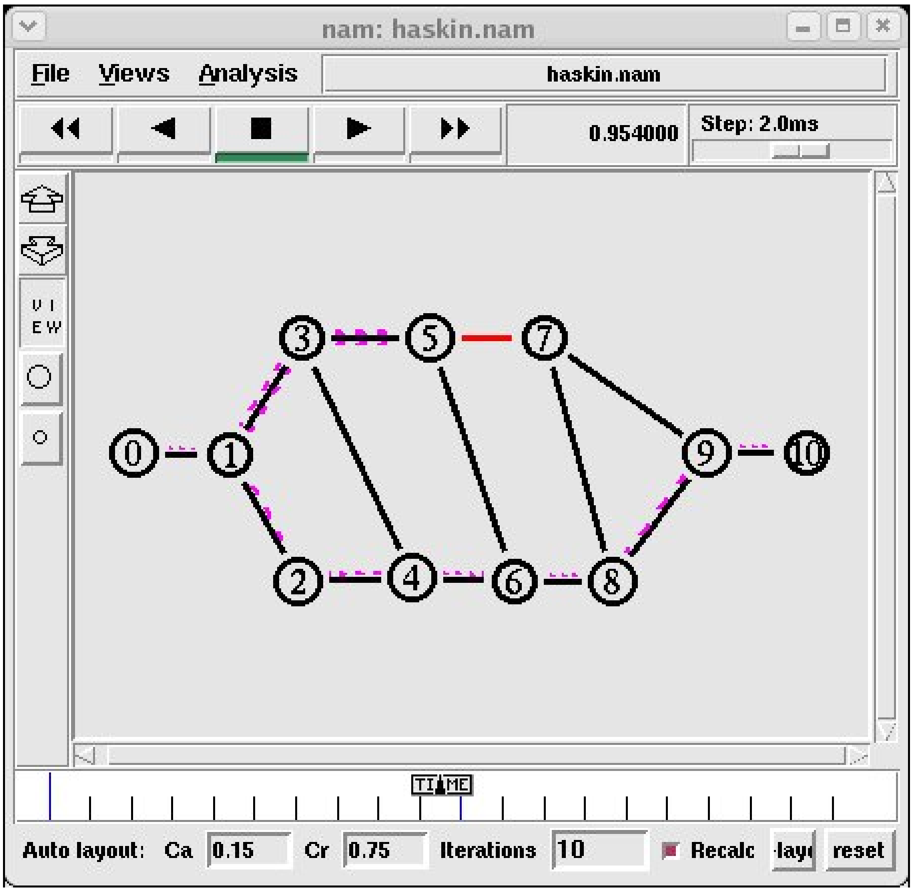


Figure 35 : Haskins Model in Nam

The above picture shows how traffic flows in the network when the link between LRS 5 and LSR 7 is down.

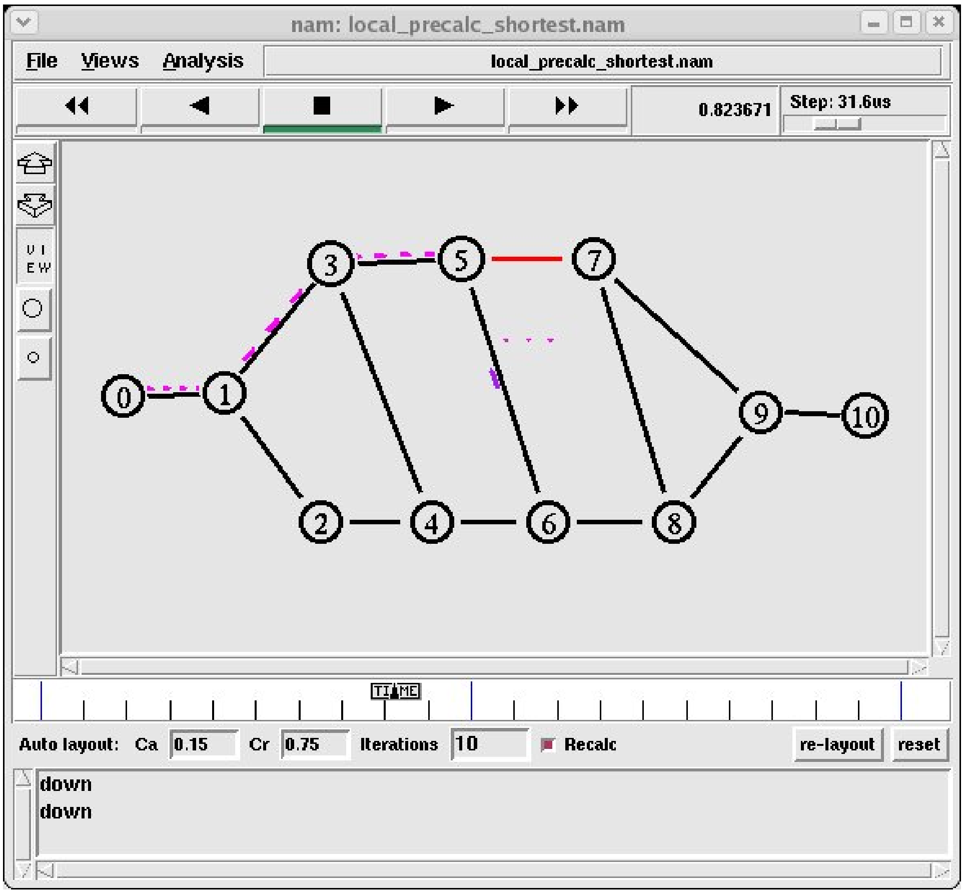
The service disruption time at node 10 is calculated to: 0.82700s - 0.80232s = 0.02468s
The reserved resources used for backup is: 8
Total dropped packets: 58

6.3.4 Local rerouting

In this model local rerouting is used. When LSR5 detects that the link to LSR7 is down, it will start rerouting to find a new path to the egress LSR9. This will be done by pruning LSR7 from its local copy of the network topology and calculate a new shortest path that can be merged with the working path downstream from LSR7. In this case the path (5,6,8,9) is found and RSVP-TE is used to setup labels for this new LSP. When the new LSP is setup the working path will be spliced with the new LSP and traffic will be forwarded on this path. While the setup of the new recovery path is done, packets will be dropped by the LSR next to the failed link.

Depending on which link that breaks the following recovery paths will be found by the rerouting algorithm:

Link between LSR1 and LSR3 breaks: recovery path (1,2,4,6,5)
Link between LSR3 and LSR5 breaks: recovery path (3,4,6,8,9)
Link between LSR5 and LSR7 breaks: recovery path (5,6,8,9)
Link between LSR7 and LSR9 breaks: recovery path (7,8,9)



Auto layout: Ca 0.15 Cr 0.75 Iterations 10

down
down

Figure 36 : Local Rerouting in Nam

The above picture shows the RESV message sent from LSR6 to LSR5, when this message is received by LSR5 traffic will be forwarded on the recovery path.

The service disruption time at node 10 is calculated to: 0.83108s - 0.80232s = 0.02876s

The reserved resources used for backup is: 0

Total dropped packets: 79

6.3.5 Fast reroute with one-to-one backup

In this model fast reroute with one-to-one backup is simulated. Local node recovery is used for LSR3, LSR5 and LSR7. Link protection is used for the link between LSR7 and LSR9. There are 4 recovery paths that should be setup to fully protect this working path.

Node protection for LSR3:	(1,2,4,6,5)
Node protection for LSR5:	(3,4,6,8,7)
Node protection for LSR7:	(5,6,8,9)
Link protection for link (5,7):	(7,8,9)

As explained in section 5.3.6 those detours will be merged to share the necessary reservations. At LSR8 the backup from LSR7 and LSR5 will be merged. At LSR6 the backup from LSR3 and LSR5 will be merged and at LSR4 the reservations from LSR1 and LSR3 will be merged. This results in the following backup reservations:

Backup path 1: From LSR7: (7,8,9)
Backup path 2: From LSR5: (5,6,8, merged at LSR8 with Backup path 1)
Backup path 3: From LSR3: (3,4,6, merged at LSR6 with Backup path 2)
Backup path 4: From LSR1: (1,2,4, merged at LSR4 with Backup path 3)

The picture below shows the final backup tree.

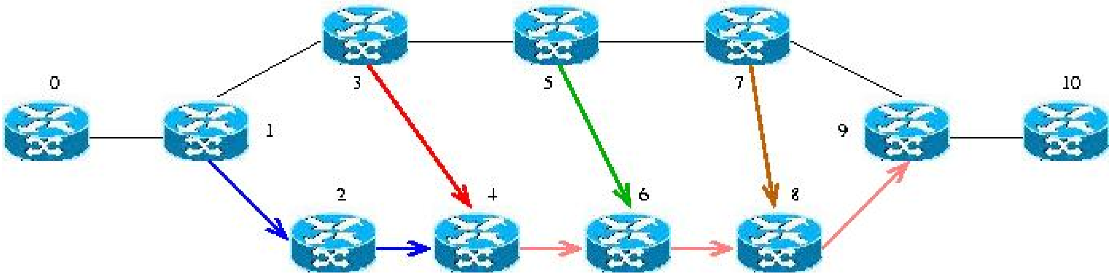
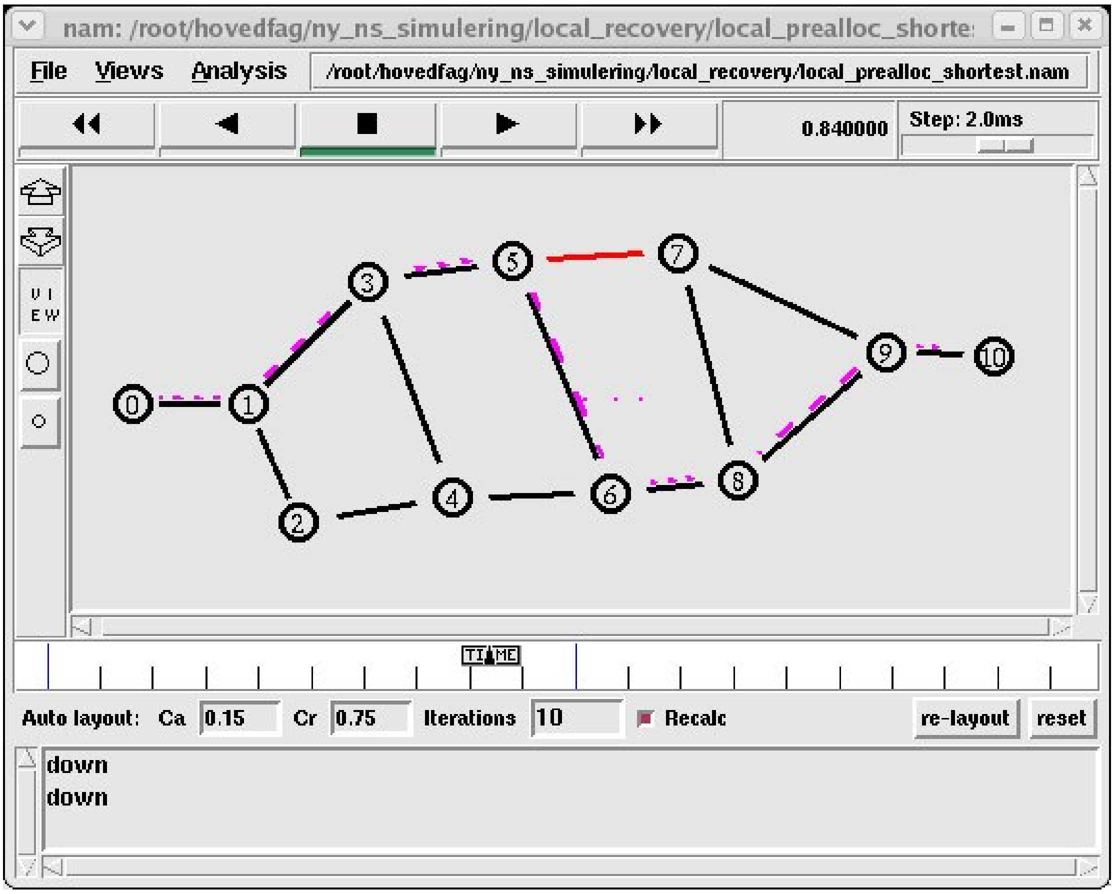


Figure 37 : Backup tree

The pink arrows show links where the backup paths have been merged.

For these simulations it was not necessary to implement the full one-to-one mechanism as in the draft [47]. The detours is set static before traffic starts to flow and not calculated by each LSR in the working path as in the draft. The merging of the detours is neither simulated as detours are set statically. Therefore each detour is reserved separately.



nam: /root/hovedfag/ny_ns_simulering/local_recovery/local_prealloc_shorte

/root/hovedfag/ny_ns_simulering/local_recovery/local_prealloc_shortest.nam

0.840000 Step: 2.0ms

Auto layout: Ca 0.15 Cr 0.75 Iterations 10 Recalc

```
down
down
```

Figure 38 : Fast Reroute one-to-one

The picture shows the network after the failure has occurred. When LSR5 detects that the failure has occurred, traffic is switched onto the recovery path (5,6,8,9).

The service disruption time at node 10 is calculated to: 0.82236s - 0.80232s = 0.02004s

The reserved resources used for backup is: 8

Total dropped packets: 58

6.4 Results

6.4.1 Packets dropped

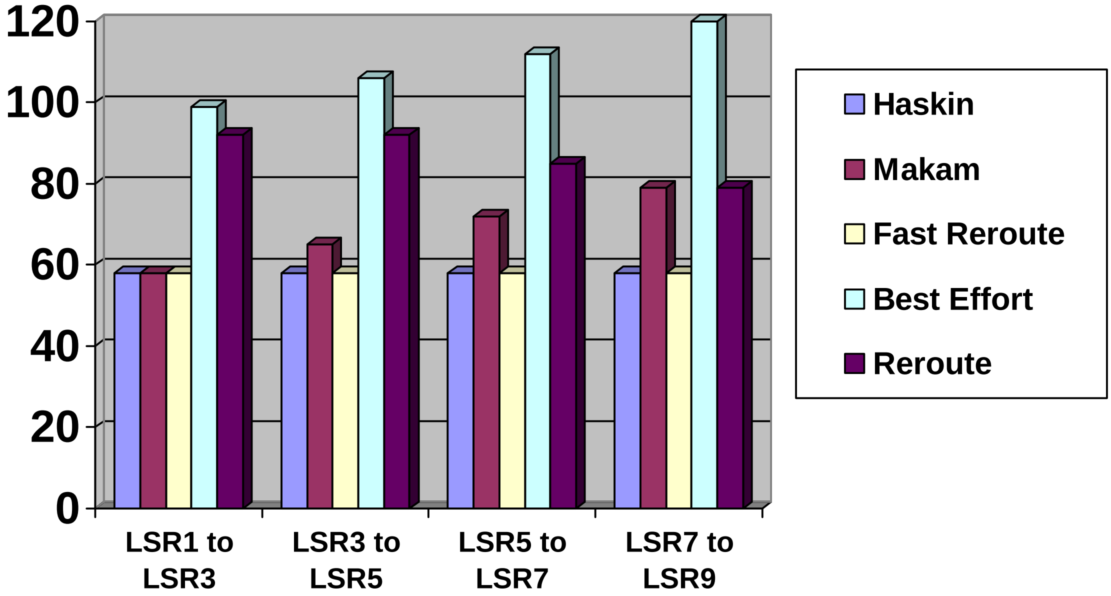


Figure 39 : Packets dropped

The chart shows the number of dropped packets for the different models and depending on which link that breaks. In both Haskins model and the fast reroute model, traffic is switched onto a pre-setup backup path by the LSR that detects the failure, so for both of those models the only packets that are dropped are the ones dropped during the failure detection time.

For Makam's and the best effort model, the number of dropped packages increases the further away from the ingress LSR the failure occurs. This is because the FIS has to be sent back to the ingress before traffic can be switched to the backup path. In Makam's model the number of dropped packages is the same as for Haskin and fast reroute only if the failure occurs on the link out from the ingress, this is because in this case no FIS has to be sent before traffic can be forwarded on the backup path. For the best effort model, the number of dropped packages is large because packages are dropped during the failure detection time, the time for the FIS to be sent to the ingress, the time for the path calculation and during the time the backup path is setup.

For the rerouting model the number of dropped packages is the same if the link failure occurs on the link between LSR1-to-LSR3 or LSR3-to-LSR5. This is because the backup path in both of these cases is four hops long. The number of dropped packages for this model then decreases the closer to the egress LSR the failure occurs. This is because the backup path to setup becomes shorter the closer to the egress the failure occurs.

6.4.2 Service disruption

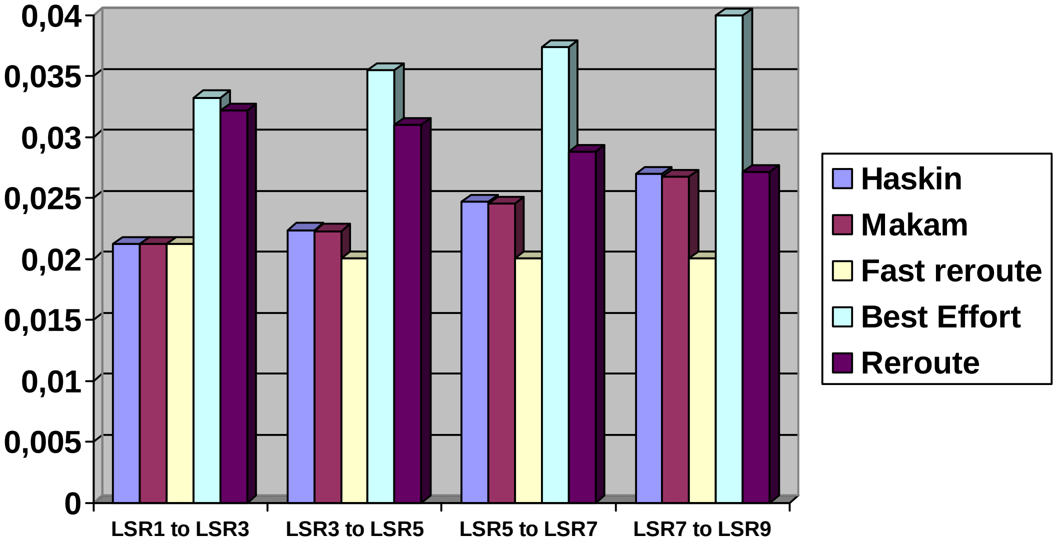


Figure 40 : Service disruption time

The above chart shows the service disruption time, measured from the last packet that was sent over the link before it breaks is received by node 10, until the first package that is using the backup path is received by this node.

For fast reroute this time is constant as the node that detects the failure is the node that switches traffic over to the backup path, and no time is used for backup path setup as the backup path is setup before the failure occurs. For all possible link breaks the backup path concatenated with the working path has the same length, so the service disruption time is the same for all possible link failures.

The time for Haskin's and Makam's models are the same, this time is larger then fast reroute in all cases where the failure occurs on a different link then on the ingress node. The time increases for these models the further away from the ingress node the failure occurs, this is because the FIS or reversed traffic has to be sent upstream to the ingress before it can be switched over on the global recovery path. For the best effort model, the service disruption time is further increased by the time to calculate and setup the backup path.

For the reroute model the service disruption time decreases the closer to the egress LSR the failure occurs. This is partly because the closer to the egress the failure occurs, the shorter the new setup backup path can be. When LSR1 setup a new backup path it is 4 hops long (2,4,6,5) , the same when LSR3 setup the backup path (4,6,8,9). The time decreases in this case because the total length of the new working path becomes longer when LSR1 has setup its backup path, but stays the same from LSR3. When LSR5 setup a backup path it is 3 hops long, and when LSR7 setup a backup paths it is 2 hops long.

6.4.3 Pre-reserved backup resources

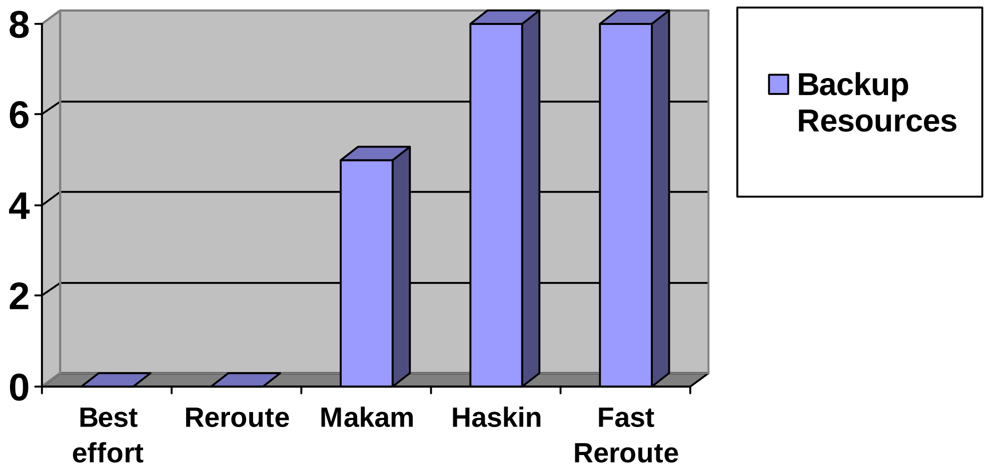


Figure 41 : Pre reserved backup resources

The chart shows the number of resources reserved for backup traffic in the network before the failure occurs. Both the best effort and rerouting model setup the backup path on demand after the failure has occurred, and therefore no backup resources are reserved before the failure in those models. Makam's models holds five resources reserved for the global backup path. Haskin's model holds five resources for the global backup path and three for the reversed path, a total of eight. The fast reroute holds eight resources reserved for the backup path.

Observe that the number of resources reserved depends on the topology of the network. If this network is compared with the fast reroute one-to-one example in the last chapter, it could be noted that adding one link to this topology would increase the resources used by fast reroute to nine.

Both Makam's and Haskin's model depends on a global recovery path, Haskins model will always use more resources then Makam's because it needs the reverse backup path in addition to this global recovery path.

In this example Haskin's and the Fast reroute model use the same amount of backup resources. This is because fast reroute after merging is set up as a global backup path with one hop connections from each LSR in the working path to this backup path, as can be seen in the picture in section 6.3.5. This is the ideal setup for the fast reroute mechanism as it uses minimal amount of backup resources after the merging.

For most topologies fast reroute will use more resources then Haskin's model, but this example is used to show that when the topology is right, fast reroute will use the same amount of backup resources as Haskins model.

7. Conclusion

Most of the techniques used for recovery in MPLS are already available at other network layers. It might therefore seem to be unnecessary to implement these mechanisms again in another layer, but because MPLS operates between layer 2 and layer 3 in the network model, it can take advantage of recovery techniques used at these layers and implement them in this new layer. This opens up for new possibilities in network recovery, as these techniques, when implemented at the MPLS layer can get new functionality. It is also important that MPLS has its own mechanisms for recovery, as MPLS is designed to work with many different network technologies. If MPLS shall be classed as a layer that can be used with fast recovery, then there must be techniques in MPLS that makes this kind of recovery possible, independent of other layer mechanisms.

Recovery mechanisms at the lower layers can be fast because protection switching can be used, but these approaches are often not optimal in a network utilization perspective. Lower layer mechanism also suffers from the non visibility into higher layers, as traffic can not be differentiated and different traffic classes can not have different recovery schemes. In layer 3, the rerouting recovery mechanism can be used for network recovery. Rerouting is time consuming but resource usage for this recovery mechanism is much smaller then for protection switching.

In MPLS the granularity for which traffic that is being protected by a recovery mechanism is per LSP. This means that if QoS schemes and traffic separation is used in the network, then traffic that is highly sensitive to delay can be classified to a high priority class with for example DiffServ and forwarded on separate paths in the network. This high priority traffic can use recovery mechanism that cause the least amount of service disruption time in the case of a network failure, while best effort traffic can use other less resource consuming approaches for recovery. With MPLS this opens up for the ability to offer different recovery classes in a network. This means that fever resources will be bound for recovery as not all LSPs in the network requires the same treatment when a failure occurs.

In my simulations I looked at three different comparison criteria’s for the network recovery models:

- The packet loss during the recovery operation.
- The service disruption time.
- The number of pre-reserved resources used for the recovery operation.

Packet loss during the recovery operation

The recovery mechanism that causes the most packet loss is the rerouting mechanism, as with rerouting packets can be dropped during the time it takes to do failure detection, failure notification, recovery path calculations and recovery path setup.
The packet loss in rerouting can be reduced by minimizing these intervals.

The failure detection time depends on the failure detection mechanism used. For link and control plane failures the MPLS mechanism to use for failure detection is the hello mechanism in RSVP-TE. This mechanism can be optimized by setting the most optimal values for the hello interval and the failure check multiplier. For faster failure detection lower layer mechanism can be used if available.

The failure notification time can be minimized by setting each LSR in the working path as PSL, the node that detects the failure can then start the rerouting mechanism without the need of a failure notification message.

The rerouting mechanism can be further optimized by reducing the path calculation time, this can be achieved if recovery paths are pre-calculated by each PSL every time a new link state advertisement is received.

The recovery path setup time is highly dependent of the network topology and the available resources in the network, the longer the path has to be, the more packets will be dropped during the setup time.

With the use of the protection switching mechanism fewer packets can be dropped as the recovery path is pre-setup. In protection switching packets can be dropped during the time for failure detection and failure notification. To reduce the number of dropped packages the failure notification time can be minimized with the use of local recovery, as in the fast reroute one-to-one proposal. Another alternative to reduce the number of packets dropped during the failure notification time is to use the traffic itself as a fault indication signal. This is done by the use of a reverse recovery path as in Haskins model.

Service disruption time

The service disruption time depends just like the number of dropped packages on the time for failure detection, failure notification, recovery path calculations and recovery path setup.

This means that the rerouting mechanism will have higher service disruption time then protection switching, because with rerouting time is used for path calculations and path setup and this is not needed in protection switching.

With protection switching, local recovery can be used to minimize the failure notification time, as in the fast reroute models. But the service disruption time also depends on the length of the new path that traffic needs to traverse before it reaches the egress node. When the recovery path is in use, the shorter the path from the point of failure to the egress node can be, the shorter the service disruption will be.
This was shown in the comparison of Haskins model to Makam’s model. For the number of packets dropped, Haskins model is an improvement over Makam’s model. But in service disruption time, the models will use the same time. Although Haskins model has a local switchover to the reverse backup path, this will not be an improvement over Makam’s models, as both models use a global recovery path.

The mechanism that will have the least amount of service disruption time is 1+1 protection switching. With this mechanism, no time is used for failure notification, recovery path calculations or recovery path setup. The fast reroute model is the fastest way to do a protection switching recovery operation without using the 1+1 model.

Pre-reserved resources used for the recovery operation.

The number of pre-reserved resources used is only of interest if protection switching is used, as rerouting does not reserve any resources before the recovery operation starts. Rerouting is therefore the best alternative in consideration of the least amount of pre-reserved resources.

For protection switching the number of pre-reserved resources used to fully protect a working path end-to-end, depends on the network topology, the recovery model and which kind of protection switching that is used.

For protection switching with 1+1 the same amount of recourses will be reserved as for 1:1 models. But the difference is how the resources are used. In 1+1 the resources are in use and can not be used for other traffic, but in the 1:1 case those resources can be used by extra traffic with low priority.

Haskins model will always use more resources then Makam's model, as the reverse path is used in addition to the global recovery path. Fast reroute will in most cases use more recourses then Haskins model, but in the optimal topology the same amount of resources will be used by these models.

This can be seen if we think that both models use a global protection path. For each node on the working path, there needs to be a way to connect from this node to the global protection path. In Haskins model this is done by the reversed path. For a working path with N links, the reversed path needs to reserve resources on (N-1) links, as no reservation in the reversed direction is needed on the link to the egress node. For fast reroute each node in the working path needs to connect directly to a merge point with the global recovery path. This means that if each node in the working path except for the ingress and egress node needs to be connected with a link to a node in the global path. A working path with N links has N+1 nodes. The egress and ingress does not need a direct connection to the global recovery path, this means that N+1-2 = N-1 nodes needs to make reservations. This means that Haskins and Fast reroute can be using the same amount of pre-reserved resources in the right topology.

But the situation becomes more complicated with resource sharing. Then the most optimal resource usage can be achieved by placing the recovery paths so that reserved resources can be shared in the most optimal way. Then a new recovery path in the network might not need to make any new backup reservations, if already established recovery path resources can be shared.

To pick one model that always will be best suited for recovery in a MPLS domain can not be done, which model to use depends on what traffic that needs protection and what resources that are available for recovery in the network.

For high priority traffic that needs fast recovery, protections switching is the best mechanism to use. The fastest way to have recovery with the least amount of package loss in a network is through protection switching with 1+1 protection. This is used in the SONET/SDH UPRS approach and should also be possible to setup in MPLS. This approach is the most costly way to do recovery concerning recourses used, as traffic is sent simultaneously over two disjoint paths. In SONET/SDH there is no differentiation between traffic and when this model is used all traffic that passes through the protected path will be duplicated onto a protection path. In MPLS fewer resources can be reserved as the protection can be setup per LSP instead. But the resources reserved for this kind of recovery in MPLS will be in constant use and can therefore not be used for extra traffic.

It is therefore more resource efficient to use 1:N protection switching where the reserved resources can be used by extra traffic. To get the shortest service disruption time in 1:N protection switching, local recovery shall be used which lead to the fast reroute model. This model needs many resources reserved for backup purposes, and in a network with many high priority flows this might lead to a non optimal utilization of the network. The facility model is a proposal that can be used to decrease the amount of control traffic for the fast reroute proposal, but this does not decrease the amount of reserved resources. To decrease the amount of reserved resources when protection switching with 1:N is used, protection sharing should be used were possible.

For best effort traffic, rerouting can be used as service disruption time is not critical. If the traffic is non sensitive to disruption and marked as low priority, it can be classed as extra traffic and can use links that are reserved for other high priority traffic that use recovery by protection switching. In this case the best effort traffic can use rerouting to find another path if a failure occurs on the working path, or if the high priority traffic needs the bandwidth as a failure has occurred elsewhere in the network.

7.1 Topics for further research

Further research of protection sharing is needed, as sharing of backup resources is not implemented in any of today’s recovery models for MPLS recovery. If fast reroute is used this can consume a lot of recourses in the network and sharing could be done where this is possible. This requires an extension to the routing protocols used and the ability to transport the information of for which nodes a recovery path is setup to protect. An extension to RSVP-TE can be used to transport this information to the LSR that makes a reservation so that a check for available sharing of recourses can be performed. OSPF-TE can be extended to include a sharing database so that the optimal route for sharing can be computed before path setup is performed.

This thesis has mainly looked at recovery from link failures and control plain failures, these failures can be detected by the RSVP-TE hello mechanism. Currently the BDF protocol is under development to implement a faster data plain failure detection mechanism. Further research can be done in the field of failure detection in the data plain.

All models presented in this thesis have one thing in common. They can protect a working path end-to-end in one MPLS domain. But there is no protection for node failures on the ingress or egress LSR. If recovery shall be used on paths cross multiple MPLS domains, there needs to be a way that these border nodes can be protected cross domains.

This work has focused on MPLS-based technologies. GMPLS control planes have been considered to extend MPLS to optical networks. An in-depth analysis of current and future optical network technology should be considered for future work

During the work with this thesis I tried many different simulation environments for recovery in MPLS domains. Further work can be done in getting a good open source environment for MPLS simulations.

8. Referances

[1] E.Rosen, A. Viswanathan, R.Callon
“Multiprotocol Label Switching Architecture (RFC 3031)”
http://www.ietf.org/rfc/rfc3031.txt
January 2001

[2] V. Alwayn
“Advanced MPLS Design and Implementation”
ISBN 1-58705-020-X
2001

[3] L. Andersson, P. Doolan, N. Feldman, A. Fredette, B. Thomas
“LDP Specification (RFC 3036)”
http://rfc-3036.rfc-list.net/
January 2001

[4] S. Harnedy
“The MPLS Primer”
ISBN 0-13-032980-0
2002

[5] B. Jamoussi, L. Andersson, R. Callon, R. Dantu, L. Wu, P. Doolan, T. Worster, N. Feldman, A. Fredette, M. Girish, E. Gray, J. Heinanen, T. Kilty, A. Malis
“Constraint-Based LSP Setup using LDP (RFC 3212)”
http://rfc-3212.rfc-list.net/
January 2002

[6] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, G. Swallow
“RSVP-TE: Extensions to RSVP for LSP Tunnels (RFC 3209)”
http://rfc-3209.rfc-list.net/
December 2001

[7] R. Braden, L. Zhang, S. Berson, S. Herzog, S. Jamin
“Resource ReSerVation Protocol (RSVP) (RFC2205)”
http://rfc-2205.rfc-list.net/
September 1997

[8] D.Awduche, A.Hannan, X.Xiao
“Applicability Statement for Extensions to RSVP for LSP-Tunnels (RFC 3210)”
http://www.ietf.org/rfc/rcf3210.txt
December 2001

[9] Y. Rekhter, E. Rosen
“Carrying Label Information in BGP-4 (RFC 3107)”
http://rfc-3107.rfc-list.net/
May 2001

[10] D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell, J. McManus
“Requirements for Traffic Engineering Over MPLS (RFC 2702)”
http://rfc-2702.rfc-list.net/rfc-2702.htm
September 1999

[11] S. Blake, D. Black et. al
“An Architecture for Differentiated Services (RFC 2475)“
http://www.ietf.org/rfc/rfc2475.txt
December 1998

[12] A.R.Sawant, J.Qaddour
“MPLS DiffServ: A Combined Approach”
Applied Computer Science, Illinois State University

[13] G. Camarillo
“Routing architecture in Diffserv MPLS networks”
IPANA research project
April 2000

[14] C. Huang, V. Sharma, K. Owens, S.Makam
“Building Reliable MPLS Networks Using a Path Protection Mechanism”
IEEE Communications Magazine
March 2002

[15] G.Swallow
“MPLS Advantages for traffic engineering”
IEEE Communications Magazine
December 1999

[16] M. Goyal, K.K. Ramakrishnan, Wu-chi Feng
“Achieving Faster Failure Detection in OSPF Networks”
IEEE ICC´ 03
May 2003

[17] C.Alaettinoglu, V.jacobson, H Yu
“Towards millisecond IGP convergence”
NANOG 20
October 2000

[18] A. Basu, J. Riecke
“Stability issues in OSPF routing”
ACM SIGCOMM
August 2001

[19] “SONET Dual-Fed Unidirectional Path Switched Ring Equipment General Criteria”
Bellcore
GR-1400-Core, Issue 1, Revision 1, October 1995

[20] “SONET Bidirectional Line-Switched Ring Equipment Generic Criteria”
Bellcore
GR-1230-Core, Issue 2, November 1995

[21] IEEE 802.17 Resilient Packet Ring Working Group.
http://grouper.ieee.org/groups/802/17/

[22] E. Dobranowska
“Network ideology, Traffic Engineering, MPLS, Resiliency“
Master thesis
August 2003

[23] A. Autenrieth
“Differentiated Resilience in IP-Based Multilayer Transport Networks”
Ph.D thesis
September 2002

[24] G. Ellianas, E. Bouillet, R. Ramamurthy, JF. Labourdette, S. Chaudhuri, K. Bala
“Restoration in Layered Architectures with a WDM Mesh Optical Layer”
IEC Annual Review of Communications
June 2002

[25] D.A Schupke, C.G Gruber, A. Autenrieth
“Optimal Configuration of p-Cycles in WDM networks”
IEEE ICC´02
May 2002

[26] W.D Grover, D. Stamatelakis
“Bridging the ring - mesh dichotomy with p-cycles”
DRCN 2000
April 2000

[27] D. Stamatelakis, 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.
October 2000

[28] K.Owens, V.Sharma, M.Oommen, F.Hellstrand
“Network Survivability Considerations for Traffic Engineered IP Networks”
draft-owens-te-network-survivability-03.txt
May 2002

[29] J.P. Lang et al.
“Link Management Protocol (LMP)”
draft-ietf-mpls-lmp-02.txt

September 2001

[30] A. Farrel et al.
“Crankback Signaling Extensions for MPLS Signaling”
draft-iwata-mpls-crankback-07.txt
October 2003

[31] E. Harrison, A. Farrel, B. Miller.
“Protection and restoration in MPLS networks”
Data Connection White Paper
October 2001

[32] V. Sharma, F. Hellstrand
“Framework for Multi-Protocol Label Switching (MPLS)-based Recovery“
IETF, RFC 3469
February 2003

[33] C. Huang, V. Sharma, K. Owens, S.Makam
“Building Reliable MPLS Networks Using a Path Protection Mechanism”
IEEE Communications Magazine
March 2002

[34] D. Haskin, R.Krishnan
“A Method for Setting an Alternative Label Switched Paths to Handle Fast Reroute”
draft-haskin-mpls-fast-reroute-05.txt
November 2000

[35] S.Yoon, H.Lee, D. Choi ,Y.Kim
“An Efficient Recovery Mechanism for MPLS-based Protection LSP”
IEEE ICATM-2001
September 2001

[36] G.Ahn, W.Chun
“MPLS Restoration Scheme Using Least-Cost Based Dynamic Backup Path.”
ICN 2001 – Volume 2, page 319-328
July 2001

[37] C.Semeria,
“Traffic Engineering For The New Public Network”
White Paper, Juniper Networks
September 2000

[40] B.Fortz, J.Rexford, M.Thorup
“Traffic Engineering With Traditional IP Routing Protocols”
IEEE Communication Magazine
October 2002

[41] S.Makam, V.Sharma, K.Owens, C.Huang
“Protection/Restoration of MPLS Networks”
draft-makam-mpls-protection-00.txt
October 1999

[42] L.Hundessa, J.Pascual
“Fast Rerouting mechanism for a protected label switched path”
Proceedings of the IEEE International Conference on Computer Communications 01
October 2001

[43] L. Berger
“GMPLS Signaling – RSVP-TE extensions”
RFC 3473
January 2003

[44] P.Pan et al.
“Graceful Restart Mechanism for RSVP-TE”
Internet Draft
draft-pan-rsvp-te-restart-01.txt
August 2001

[45] M.Leelanivas, Y. Rekhter, R. Aggarwal
“Graceful Restart Mechanism for Label Distribution Protocol”
RFC 3478
February 2003

[46] P. Pan, G. Swallow, A.Atlas
“Fast Reroute Extensions to RSVP-TE for LSP Tunnels”
Internet Draft
draft-ietf-mpls-rsvp-lsp-fastreroute-06.txt
June 2004

[47] K. Kompella, G. Swallow
“Detecting MPLS Data Plane Failures”
Internet Draft
Draft-ietf-mpls-lsp-ping-06.txt
July 2004

[48] R. Aggarwal, K. Kompella, T. Nadeau, G. Swallow
”BDF For MPLS LSPs”
Internet Draft
Draft-ietf-bfd-mpls-00.txt
July 2004

[49] Network Strategy Partners, LLC
“Reliable IP Nodes: A Prerequisite To Profitable IP Services”
Whitepaper
November 2002

[50] C. Labovitz, A. Ahuja, F, Jahanian University of Michigan
“Experimental Study of Internet Stability and Wide-Area Backbone Failure”
Technical report
1999

[51] L. Andersson, G. Swallow
“The MPLS Working group decision on MPLS signaling protocols”
RFC 3468
February 2003

[52] L. Berger, Y. Rekhter
“Generalized MPLS Signaling – Implementation Survey”
draft-ietf-ccamp-gmpls-signaling-survey-preview-3a
November 2002

[53] Y. T'Joens, G.Ester, M. Vandenhoute
“Resilient Optical and SONET/SDH-based IP networks”
2nd International Workshop on the Design of Reliable Communication Networks
April 2000

[54] R. Kuhn
“Sources of failure in the public switched telephone network”
IEEE Computer
April 1997

[55] H. Liu, A. Sathyanath, Y.T. Wang, B. Doshi
“RSVP-TE Extension for Shared Mesh Protection”
Internet draft
draft-liu-mpls-shared-protection
October 2002

[56] A. Sathyanath, Z. Dziong, R. Nagarajan, Y.T. Wang, B. Doshi
“OSPF-TE Extensions for Shared Mesh Protection”
Internet draft
draft-sathyanath-ospf-mpls-shared-protection
October 2002

[57] Hung-ying Tyan , Ohio state university
J-sim Network simulator
http://www.j-sim.org/

[58] C. Pelsser, L. Swinnen
MPLS model for J-sim
http://www.info.ucl.ac.be/~bqu/jsim/

[59] András Varga
OMNeT++ network simulator
http://www.omnetpp.org/

[60] Xuan Thang Nguyen, University of Technology, Sydney
MPLS model for OMNeT++
http://charlie.it.uts.edu.au/~tkaphan/xtn/capstone/

[61] András Varga
INET framework for OMNeT++
http://ctieware.eng.monash.edu.au/twiki/bin/view/Simulation/INETFramework

[62] Advance Network Technologies Division, NIST
GLASS network simulator
http://dns.antd.nist.gov/glass/

[63] A. Ogielski, D. Nicol, J. Cowie
SSFNet network simulator
http://www.ssfnet.org/

[64] VINT project at LBL, Xerox PARC, USB and USC/ISI
The Network Simulator ns-2
http://www.isi.edu/nsnam/ns/

[65] G, Ahn
MNS , MPLS Network Simulator
http://flower.ce.cnu.ac.kr/~fog1/mns/

[66] R, Boeringer
RSVP-TE patch for MNS / ns-2
http://www-ihs.theoinf.tu-ilmenau.de/mitarbeiter/boeringer/rsvp_te_ns2.html
http://www.ideo-labs.com/index.php?structureID=44

[67] M, Greis´
RSVP model for ns-2
http://www.ncc.up.pt/~rprior/ns/index-en.html

[68] C. Callegari, F. Vitucci
RSVP-TE patch for MNS / ns-2
http://netgroup-serv.iet.unipi.it/rsvp-te_ns/

[69] http://heim.ifi.uio.no/~johanmp/ns-2/

[70] A. Shaikh, A. Greenberg
“Experience in black-box OSPF measurement”
ACM SIGCOMM Internet Measurement Workshop (IMW) 2001, pp. 113-125
November 2001

APPENDIX A : Resource sharing algorithm

Example of how to calculate a recovery path that uses the least amount of newly reserved bandwidth.

The calculations require the following three sets:

P-set: The links that is used for the primary path.
B-set: The links in a candidate backup path.
B-Li set: The set of primary path links that link Li protects.

The LSR that shall setup a recovery path calculates all possible paths between itself and the PML. Each set of links that are in a possible recovery path is kept in the B-set.

For each link Li in the B-set, the links that the link Li protects in a primary path is kept in the B-Li set.

The P-set contains all the links in the primary path that the new recovery path shall protect.

For each B-set the following steps are checked to choose the best suited path:

1) If a B-Li set is empty for a link in a B-set, then there is no sharing possible for that link Li. All bandwidth needs to be reserved for that link.

2) If the intersection between the P-set and the B-Li set for the link Li in the B-set is empty, then the link Li can potentially provide backup bandwidth sharing for the specified primary path. If the required backup bandwidth is less then what has already been reserved on Li, then no extra bandwidth reservation is required. Else, the difference between what is needed and what is already reserved is what is needed in additional amount of bandwidth required to support the new backup path.

3) Let the non-empty intersection between the P-set and any of the B-Li set in a B-set be represented as "La, Lb, Lc,…" this set of links includes at least one link that is the same as in the new working path and possibly other links that Li protects. Even though there is at least one link that is in the same SRG as the link in the working path, sharing with the other possible reservations can still be achieved.

The bandwidth reserved for this link is "Ba, Bb, Bc,…" Let the amount of backup bandwidth already reserved on the link Li be:

Bm = MAX [Ba, Bb, Bc,…]

And the let the backup bandwidth needed to support the new backup route be Br. Then, the amount of additional backup bandwidth that is needed to be reserved on Li is given by:

MAX [Z(Ba+Br –Bm) , Z(Bb+Br –Bm), Z(Bc+Br –Bm),…]

Z(x) = x : if x > 0 and Z(x) = 0 : if x <= 0

An example with a network

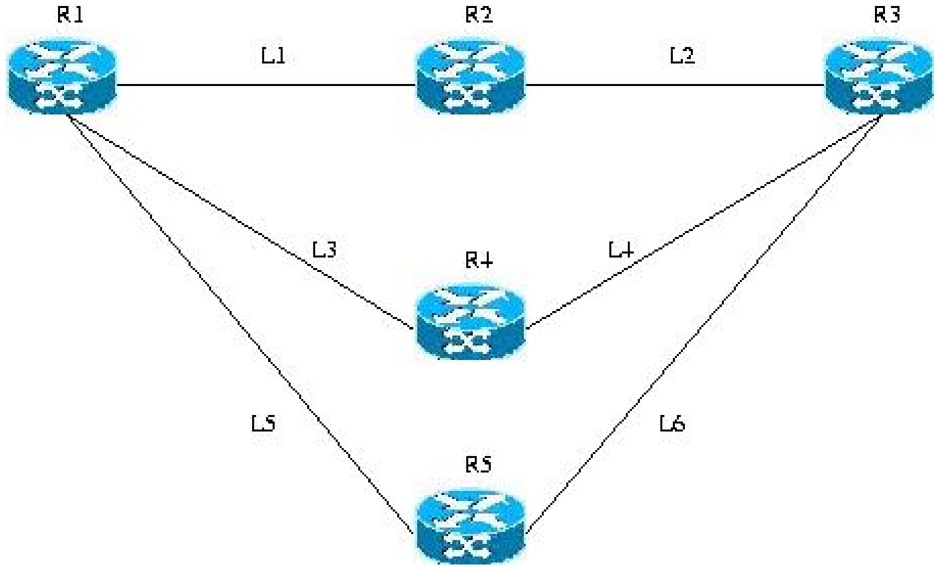


Figure 42 : Example network

Consider the above figure which is a part of a network. Consider the working path "L1, L2" with 4 units of bandwidth reserved. There are 2 possible disjoint routes to the destination that can be used for backup, they are "L3, L4" and "L5, L6". The entries of current TE/Share Database are as:

Entry for L3 --> “4::L1, L8; 5::L10"
Entry for L4 --> "4::L8, L9"
Entry for L5 --> "4::L9, L10"
Entry for L6 --> "4::L8, L10"

Where N::A,B,C means that links A, B, and C are part of a working path that this link has reserved backup recourses for of N units of bandwidth.

Router R1 shall compute a new backup path for the working path and compares the working path "L1, L2" against the entries for every link in a backup path, and repeat this for all possible backup paths. Thus on comparing with path "L3, L4", it is found from case 3) above that link L3 already has a reservation for recovery of L1 that is part of the working path that this new recovery path shall protect. If router R1 were to choose this backup path, L3 would need to reserve 3 (4 + 4 - 5) additional units of bandwidth on L3. For L4 case 2) is used and there is already 4 units reserved for this link so no new recourses needs to be reserved.

If however, backup path "L5, L6” were chosen, the intersection of P-set "L1, L2" and the B-Li set "L8, L9, L10" is null and no additional units of bandwidth would need to be reserved (since 4 units have been reserved already). Therefore under such circumstances the backup path "L5, L6" should be considered as a better backup path.

Without the modifications to OSPF-TE, RSVP-TE would have chose one of these paths
And tried to share the resources on this path without consideration of what path would have been the best to choose in the perspective of bandwidth sharing. So RSVP-TE could have chose “L3, L4” and reserved and addition of 4 units in the network.

APPENDIX B : Simulation scripts

This is the simulation scripts in TCL used for the simulations in ns-2.

Fast_reroute_one-to-one.tcl

```
#Create a new simulator object
set ns [new Simulator]

#create tracefiles
set na [open fully_local.tr w]
set nf [open fully_local.nam w]
$ns trace-all $na
$ns namtrace-all $nf
set f0 [open fully_local-bw.tr w]
set fs [open fully_local-seq.tr w]

# Set the colors for the different packets
$ns color 0 black
$ns color 100 blue
$ns color 2 green
$ns color 50 black
$ns color 46 purple
$ns color 3 red
$ns color 4 magenta

proc finish {} {
                global ns na nf f0 fs
                $ns flush-trace
                close $na
                close $nf
                close $f0
                close $fs
                exec nam fully_local.nam &
                exec xgraph -m fully_local-bw.tr -geometry 800x400 &
                exit 0
}

proc attach-expoo-traffic {node sink size burst idle rate} {
                global ns
                set source [new Agent/CBR/UDP]
                $ns attach-agent $node $source
                set traffic [new Traffic/Expoo]
                $traffic set packet-size $size
                $traffic set burst-time $burst
                $traffic set idle-time $idle
                $traffic set rate $rate
                $source attach-traffic $traffic
                $ns connect $source $sink
                return $source
}

# Define a procedure which periodically records the bandwidth received by the
# traffic sink sink0 and writes it to the file f0.
set totalpkt 0

proc record {} {
                global sink0 f0 totalpkt
                set ns [Simulator instance]

                #Set the time after which the procedure should be called again
                set time 0.065

                #How many bytes have been received by the traffic sink?
                set bw0 [$sink0 set bytes_]

                #Get the current time
                set now [$ns now]
```

```
            #Calculate the bandwidth (in MBit/s) and write it to the file
            puts $f0 "$now [expr $bw0/$time*8/1000000]"

            #Reset the bytes_ values on the traffic sink
            $sink0 set bytes_ 0

            #Re-schedule the procedure
            $ns at [expr $now+$time] "record"

            set bw0 [expr $bw0 / 200]
            set totalpkt [expr $totalpkt + $bw0]
}

proc recv-pkts {} {
            global totalpkt seqerrnb prvseqnb

            puts "The Number of Total sent packages are $prvseqnb"
            puts "The Number of Total received packages are $totalpkt"
            puts "The number of dropped packages are [expr $prvseqnb - $totalpkt]"
            puts "The Number of Total recieved unordered packages are $seqerrnb"
}

set prvseqnb -1
set seqerrnb 0

proc seq-record {size rate ftime} {
            global prvseqnb seqerrnb sink0 fs

            set ns [Simulator instance]

            #Set the time after which the procedure should be called again
            set tsize [parse-bw $size]
            set trate [parse-bw $rate]
            set time [expr double($tsize)/double($trate)/8.0]

            #Get the current time
            set now [$ns now]

            # seek the sequence number of packet.
            set revseqnb [$sink0 set expected_]

            if {$prvseqnb > $revseqnb} {
                        incr seqerrnb 1
            }

            # write the sequence number of packet to the file
            if {$prvseqnb != $revseqnb} {
                        puts $fs "$now [$sink0 set expected_]"
                        set prvseqnb $revseqnb
            }

            #Re-schedule the procedure
            if { [expr $now+$time] < $ftime } {
                        $ns at [expr $now+$time] "seq-record $size $rate $ftime"
            }
}

# routing protocol Distance Vector
$ns rtproto DV

#
# make nodes & MPLSnodes
#

set n0  [$ns node]

set n1  [$ns mpls-node]
set n2  [$ns mpls-node]
set n3  [$ns mpls-node]
set n4  [$ns mpls-node]
set n5  [$ns mpls-node]
set n6  [$ns mpls-node]
set n7  [$ns mpls-node]
set n8  [$ns mpls-node]
set n9  [$ns mpls-node]

set n10 [$ns node]


# Add RSVP-TE agents to all nodes


set rsvp1 [$n1 add-rsvp-agent]
set rsvp2 [$n2 add-rsvp-agent]
set rsvp3 [$n3 add-rsvp-agent]
set rsvp4 [$n4 add-rsvp-agent]
set rsvp5 [$n5 add-rsvp-agent]
set rsvp6 [$n6 add-rsvp-agent]
set rsvp7 [$n7 add-rsvp-agent]
set rsvp8 [$n8 add-rsvp-agent]
set rsvp9 [$n9 add-rsvp-agent]

# add variables for mpls modules

set LSRmpls1   [eval $n1 get-module "MPLS"]
set LSRmpls2   [eval $n2 get-module "MPLS"]
set LSRmpls3   [eval $n3 get-module "MPLS"]
set LSRmpls4   [eval $n4 get-module "MPLS"]
set LSRmpls5   [eval $n5 get-module "MPLS"]
set LSRmpls6   [eval $n6 get-module "MPLS"]
set LSRmpls7   [eval $n7 get-module "MPLS"]
set LSRmpls8   [eval $n8 get-module "MPLS"]
set LSRmpls9   [eval $n9 get-module "MPLS"]

#
# make links
#

$ns duplex-rsvp-link $n0 $n1 10Mb 1ms 0.99 1000 10000 Param Null

$ns duplex-rsvp-link $n1 $n3 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n3 $n5 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n5 $n7 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n7 $n9 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n1 $n2 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n2 $n4 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n4 $n6 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n6 $n8 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n8 $n9 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n1 $n2 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n3 $n4 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n5 $n6 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n7 $n8 10Mb 1ms 0.99 1000 10000 Param Null

$ns duplex-rsvp-link $n9 $n10 10Mb 1ms 0.99 1000 10000 Param Null


# Create a traffic sink and attach it to the node node10
set sink0 [new Agent/LossMonitor]
$ns attach-agent $n10 $sink0
$sink0 clear

# Create a traffic source
set src0 [attach-expoo-traffic $n0 $sink0 200 0 0 5000k]
$src0 set fid_ 100
$ns color 100 magenta

# Enable upcalls on all nodes
Agent/RSVP set noisy_ 255

# Set re-route option to drop
$ns enable-reroute drop

# Start recording
$ns at 0.0 "record"
$ns at 0.3 "seq-record 200 5000k 2.0"
```

```
# The setup of working LSP
$ns at 0.0 "$LSRmpls1 create-crlsp $n0 $n9 0 100 1000 +400000 5000 32 1_3_5_7_9_"

# setup of the local recovery paths

$ns at 0.2 "$LSRmpls1 create-erlsp $n0 $n5 1 100 2000 1_2_4_6_5_"
$ns at 0.3 "$LSRmpls3 create-erlsp $n0 $n9 1 100 3000 3_4_6_8_9_"
$ns at 0.4 "$LSRmpls5 create-erlsp $n0 $n9 2 100 4000 5_6_8_9_"
$ns at 0.5 "$LSRmpls7 create-erlsp $n0 $n9 1 100 5000 7_8_9_"

# bind a flow to working LSP
$ns at 0.4 "$LSRmpls1 bind-flow-erlsp 10 100 1000"

#bind backup path to lsp
$ns at 0.7 "$LSRmpls1 reroute-lsp-binding 1000 2000"
$ns at 0.7 "$LSRmpls3 reroute-lsp-binding 1000 3000"
$ns at 0.7 "$LSRmpls5 reroute-lsp-binding 1000 4000"
$ns at 0.7 "$LSRmpls7 reroute-lsp-binding 1000 5000"

#start to send
$ns at 0.5 "$src0 start"

#active failure detection
$ns at 0.1 "$ns activate-rsvp-hello 0.005 3.5"

#break link
$ns rtmodel-at 0.8 down $n5 $n7

#stop sending
$ns at 1.8 "$src0 stop"

#finish simulation
$ns at 2.0 "recv-pkts"
$ns at 2.0 "record"
$ns at 2.0 "finish"

$ns run
```

Haskin.tcl

```
#Create a new simulator object
set ns [new Simulator]
set na [open haskin.tr w]
set nf [open haskin.nam w]
$ns trace-all $na
$ns namtrace-all $nf
set f0 [open haskin-bw.tr w]
set fs [open haskin-seq.tr w]

#set colors of messages
$ns color 0 black
$ns color 1 blue
$ns color 2 green
$ns color 50 black
$ns color 46 purple
$ns color 3 red

proc finish {} {
                global ns na nf f0 fs
                $ns flush-trace
                close $na
                close $nf
                close $f0
                close $fs
                exec nam haskin.nam &
                exec xgraph -m haskin-bw.tr -geometry 800x400 &
                exit 0
}


# used to setup the sender
proc attach-expoo-traffic {node sink size burst idle rate} {
                global ns
                set source [new Agent/CBR/UDP]
                $ns attach-agent $node $source
                set traffic [new Traffic/Expoo]
                $traffic set packet-size $size
                $traffic set burst-time $burst
                $traffic set idle-time $idle
                $traffic set rate $rate
                $source attach-traffic $traffic
                $ns connect $source $sink
                return $source
}

# Define a procedure which periodically records the bandwidth received by the
# traffic sink sink0 and writes it to the file f0.
set totalpkt 0

proc record {} {
                global sink0 f0 totalpkt
                set ns [Simulator instance]

                #Set the time after which the procedure should be called again
                 set time 0.03532

                #How many bytes have been received by the traffic sink?
                set bw0 [$sink0 set bytes_]

                #Get the current time
                set now [$ns now]

                #Calculate the bandwidth (in MBit/s) and write it to the file
                puts $f0 "$now [expr $bw0/$time*8/1000000]"

                #Reset the bytes_ values on the traffic sink
                $sink0 set bytes_ 0

                #Re-schedule the procedure
                $ns at [expr $now+$time] "record"
                set bw0 [expr $bw0 / 200]
                set totalpkt [expr $totalpkt + $bw0]
}

proc recv-pkts {} {
                global totalpkt seqerrnb prvseqnb

                puts "The Number of Total sent packages are $prvseqnb"
                puts "The Number of Total received packages are $totalpkt"
                puts "The number of dropped packages are [expr $prvseqnb - $totalpkt]"
                puts "The Number of Total recieved unordered packages are $seqerrnb"

}

set prvseqnb -1
set seqerrnb 0

proc seq-record {size rate ftime} {
                global prvseqnb seqerrnb sink0 fs
                set ns [Simulator instance]

                #Set the time after which the procedure should be called again
                set tsize [parse-bw $size]
                set trate [parse-bw $rate]
                set time [expr double($tsize)/double($trate)/8.0]

                #Get the current time
                set now [$ns now]

                # seek the sequence number of packet.
                set revseqnb [$sink0 set expected_]

                if {$prvseqnb > $revseqnb} {
                                incr seqerrnb 1
                }

                # write the sequence number of packet to the file
                if {$prvseqnb != $revseqnb} {
                                puts $fs "$now [$sink0 set expected_]"
                                set prvseqnb $revseqnb
                }

                #Re-schedule the procedure
                if { [expr $now+$time] < $ftime } {
                                $ns at [expr $now+$time] "seq-record $size $rate $ftime"
                }
}

# routing protocol distance vector
$ns rtproto DV


# make nodes & MPLSnodes

set n0  [$ns node]

set n1  [$ns mpls-node]
set n2  [$ns mpls-node]
set n3  [$ns mpls-node]
set n4  [$ns mpls-node]
set n5  [$ns mpls-node]
set n6  [$ns mpls-node]
set n7  [$ns mpls-node]
set n8  [$ns mpls-node]
set n9  [$ns mpls-node]

set n10 [$ns node]


# Add RSVP-TE agents to all nodes

set rsvp1 [$n1 add-rsvp-agent]
set rsvp2 [$n2 add-rsvp-agent]
set rsvp3 [$n3 add-rsvp-agent]
set rsvp4 [$n4 add-rsvp-agent]
set rsvp5 [$n5 add-rsvp-agent]
set rsvp6 [$n6 add-rsvp-agent]
set rsvp7 [$n7 add-rsvp-agent]
set rsvp8 [$n8 add-rsvp-agent]
set rsvp9 [$n9 add-rsvp-agent]

# add variables for mpls modules

set LSRmpls1   [eval $n1 get-module "MPLS"]
set LSRmpls2   [eval $n2 get-module "MPLS"]
set LSRmpls3   [eval $n3 get-module "MPLS"]
set LSRmpls4   [eval $n4 get-module "MPLS"]
set LSRmpls5   [eval $n5 get-module "MPLS"]
set LSRmpls6   [eval $n6 get-module "MPLS"]
set LSRmpls7   [eval $n7 get-module "MPLS"]
set LSRmpls8   [eval $n8 get-module "MPLS"]
set LSRmpls9   [eval $n9 get-module "MPLS"]


# make links

$ns duplex-rsvp-link $n0 $n1 10Mb 1ms 0.99 1000 10000 Param Null

$ns duplex-rsvp-link $n1 $n3 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n3 $n5 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n5 $n7 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n7 $n9 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n1 $n2 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n2 $n4 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n4 $n6 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n6 $n8 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n8 $n9 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n1 $n2 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n3 $n4 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n5 $n6 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n7 $n8 10Mb 1ms 0.99 1000 10000 Param Null

$ns duplex-rsvp-link $n9 $n10 10Mb 1ms 0.99 1000 10000 Param Null


# Create a traffic sink and attach it to the node node10
set sink0 [new Agent/LossMonitor]
$ns attach-agent $n10 $sink0
$sink0 clear

# Create a traffic source
set src0 [attach-expoo-traffic $n0 $sink0 200 0 0 5000k]
$src0 set fid_ 100
$ns color 100 magenta

# Enable upcalls on all nodes
Agent/RSVP set noisy_ 255

# Set re-route option to drop
$ns enable-reroute drop

# Start recording
$ns at 0.0 "record"
$ns at 0.3 "seq-record 200 5000k 2.0"

# The setup of working LSP
$ns at 0.0 "$LSRmpls1 create-erlsp $n0 $n9 0 100 1000 1_3_5_7_9_"


# The setup of global recovery LSP
$ns at 0.1 "$LSRmpls1 create-erlsp $n1 $n9 1 100 2000 1_2_4_6_8_9_"

#activate the hello mechanism
$ns at 0.1 "$ns activate-rsvp-hello 0.005 3.5"

#Setup of reverse path. php used..
$ns at 0.4 "$LSRmpls7 create-erlsp $n7 $n2 1 100 2005 7_5_3_1_2_"
```

```
# bind a flow to working LSP
$ns at 0.4 "$LSRmpls1 bind-flow-erlsp 10 100 1000"

#start sending traffic
$ns at 0.5 "$src0 start"

#binding working LSP to alternative LSP
$ns at 0.6  "$LSRmpls1 reroute-lsp-binding      1000    2000"
$ns at 0.6  "$LSRmpls3 reroute-lsp-binding      1000    2005"
$ns at 0.6  "$LSRmpls5 reroute-lsp-binding      1000    2005"
$ns at 0.6  "$LSRmpls7 reroute-lsp-binding      1000    2005"


#break link
$ns rtmodel-at 0.8 down $n5 $n7

#stop trffic
$ns at 1.8 "$src0 stop"

#end simulation
$ns at 2.0 "recv-pkts"
$ns at 2.0 "record"
$ns at 2.0 "finish"

$ns run
```

Local_Reroute.tcl

```
#Create a new simulator object
set ns [new Simulator]

set na [open local_precalc_shortest.tr w]
set nf [open local_precalc_shortest.nam w]
$ns trace-all $na
$ns namtrace-all $nf
set f0 [open local_precalc_shortest-bw.tr w]
set fs [open local_precalc_shortest-seq.tr w]

#colors for the message types
$ns color 0 yellow
$ns color 100 blue
$ns color 2 green
$ns color 50 black
$ns color 46 purple
$ns color 3 red
$ns color 4 magenta

proc finish {} {
			global ns na nf f0 fs
			$ns flush-trace
			close $na
			close $nf
			close $f0
			close $fs
			exec nam local_precalc_shortest.nam &
			exec xgraph -m local_precalc_shortest-bw.tr -geometry 800x400 &
			exit 0
}

proc attach-expoo-traffic {node sink size burst idle rate} {
			global ns
			set source [new Agent/CBR/UDP]
			$ns attach-agent $node $source
			set traffic [new Traffic/Expoo]
			$traffic set packet-size $size
			$traffic set burst-time $burst
			$traffic set idle-time $idle
			$traffic set rate $rate
			$source attach-traffic $traffic
			$ns connect $source $sink
			return $source
}

# Define a procedure which periodically records the bandwidth received by the
# traffic sink sink0 and writes it to the file f0.
set totalpkt 0

proc record {} {
			global sink0 f0 totalpkt
			set ns [Simulator instance]

			#Set the time after which the procedure should be called again
			set time 0.065

			#How many bytes have been received by the traffic sink?
			set bw0 [$sink0 set bytes_]

			#Get the current time
			set now [$ns now]

			#Calculate the bandwidth (in MBit/s) and write it to the file
			puts $f0 "$now [expr $bw0/$time*8/1000000]"

			#Reset the bytes_ values on the traffic sink
			$sink0 set bytes_ 0

			#Re-schedule the procedure
			$ns at [expr $now+$time] "record"
			set bw0 [expr $bw0 / 200]
			set totalpkt [expr $totalpkt + $bw0]
}

proc recv-pkts {} {
			global totalpkt seqerrnb prvseqnb

			puts "The Number of Total sent packages are $prvseqnb"
			puts "The Number of Total received packages are $totalpkt"
			puts "The number of dropped packages are [expr $prvseqnb - $totalpkt]"
			puts "The Number of Total recieved unordered packages are $seqerrnb"

}

set prvseqnb -1
set seqerrnb 0

proc seq-record {size rate ftime} {
			global prvseqnb seqerrnb sink0 fs
			set ns [Simulator instance]

			#Set the time after which the procedure should be called again
			set tsize [parse-bw $size]
			set trate [parse-bw $rate]
			set time [expr double($tsize)/double($trate)/8.0]

			#Get the current time
			set now [$ns now]

			# seek the sequence number of packet.
			set revseqnb [$sink0 set expected_]

			if {$prvseqnb > $revseqnb} {
						incr seqerrnb 1
			}

			# write the sequence number of packet to the file
			if {$prvseqnb != $revseqnb} {
						puts $fs "$now [$sink0 set expected_]"
						set prvseqnb $revseqnb
			}

			#Re-schedule the procedure
			if { [expr $now+$time] < $ftime } {
						$ns at [expr $now+$time] "seq-record $size $rate $ftime"
			}
}

# routing protocol
$ns rtproto DV

#
# make nodes & MPLSnodes
#

set n0  [$ns node]

set n1  [$ns mpls-node]
set n2  [$ns mpls-node]
set n3  [$ns mpls-node]
set n4  [$ns mpls-node]
set n5  [$ns mpls-node]
set n6  [$ns mpls-node]
set n7  [$ns mpls-node]
set n8  [$ns mpls-node]
set n9  [$ns mpls-node]

set n10 [$ns node]


# Add RSVP-TE agents to all nodes


set rsvp1 [$n1 add-rsvp-agent]
set rsvp2 [$n2 add-rsvp-agent]
set rsvp3 [$n3 add-rsvp-agent]
set rsvp4 [$n4 add-rsvp-agent]
set rsvp5 [$n5 add-rsvp-agent]
set rsvp6 [$n6 add-rsvp-agent]
set rsvp7 [$n7 add-rsvp-agent]
set rsvp8 [$n8 add-rsvp-agent]
set rsvp9 [$n9 add-rsvp-agent]

# add variables for mpls modules

set LSRmpls1   [eval $n1 get-module "MPLS"]
set LSRmpls2   [eval $n2 get-module "MPLS"]
set LSRmpls3   [eval $n3 get-module "MPLS"]
set LSRmpls4   [eval $n4 get-module "MPLS"]
set LSRmpls5   [eval $n5 get-module "MPLS"]
set LSRmpls6   [eval $n6 get-module "MPLS"]
set LSRmpls7   [eval $n7 get-module "MPLS"]
set LSRmpls8   [eval $n8 get-module "MPLS"]
set LSRmpls9   [eval $n9 get-module "MPLS"]

#
# make links
#

$ns duplex-rsvp-link $n0 $n1 10Mb 1ms 0.99 1000 10000 Param Null

$ns duplex-rsvp-link $n1 $n3 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n3 $n5 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n5 $n7 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n7 $n9 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n1 $n2 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n2 $n4 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n4 $n6 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n6 $n8 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n8 $n9 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n1 $n2 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n3 $n4 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n5 $n6 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n7 $n8 10Mb 1ms 0.99 1000 10000 Param Null

$ns duplex-rsvp-link $n9 $n10 10Mb 1ms 0.99 1000 10000 Param Null


# Create a traffic sink and attach it to the node node10
set sink0 [new Agent/LossMonitor]
$ns attach-agent $n10 $sink0
$sink0 clear

# Create a traffic source
set src0 [attach-expoo-traffic $n0 $sink0 200 0 0 5000k]
$src0 set fid_ 100
$ns color 100 magenta

# Enable upcalls on all nodes
Agent/RSVP set noisy_ 255


#function that calculates a new recovery path

proc setup_backup_path {lspid} {

            global ns LSRmpls5 n0 n9

            set oiface   [$LSRmpls5 get-outgoing-iface $lspid]
            set fec      [$LSRmpls5 get-fec-for-lspid  $lspid]

            set qosid [$LSRmpls5 get-flowid-for-lspid $lspid]
            set failnext [$LSRmpls5 get-failnext-for-lspid $lspid]

            if {$failnext != -1} {
                        set fnodes [split $failnext "_"]
                        set fnodes [linsert $fnodes 0 $oiface]
            } else {
```

```
                                set fnodes $oiface
                }

                set er [$ns get-shortest-path [[$LSRmpls5 node] id] $fec $fnodes "" ]

                while { $er == -1 } {
                                if { [llength $fnodes] == 0} {
                                                puts "error: routing failure"
                                                return -1
                                }
                                set dst [lindex $fnodes end]
                                set fnodes [lreplace $fnodes end end]
                                set er [$ns get-shortest-path [[$LSRmpls5 node] id] $dst $fnodes "" ]
                }

                set er [append $er $er _]
                puts "Setting up recovery path $er"

                $ns at [$ns now] "$LSRmpls5 create-crlsp $n0 $n9 1 100 2000 +400000 5000 32 $er"

                #bind backup path to lsp, value for time found in tracefiles
                $ns at 0.8263 "$LSRmpls5 reroute-lsp-binding 1000 2000"
}

# Set re-route option to drop
$ns enable-reroute drop


# Start recording
$ns at 0.0 "record"
$ns at 0.3 "seq-record 200 5000k 2.0"

# The setup of working LSP
$ns at 0.0 "$LSRmpls1 create-crlsp $n0 $n9 0 100 1000 +400000 5000 32 1_3_5_7_9_"


# bind a flow to working LSP
$ns at 0.4 "$LSRmpls1 bind-flow-erlsp 10 100 1000"

#start sending
$ns at 0.5 "$src0 start"

#break link
$ns rtmodel-at 0.8 down $n5 $n7

#active failure detection
$ns at 0.1 "$ns activate-rsvp-hello 0.005 3.5"

# The setup of backup LSP, time added for path calculation
$ns at 0.81949 "setup_backup_path 1000"

$ns at 1.8 "$src0 stop"


$ns at 2.0 "recv-pkts"
$ns at 2.0 "record"
$ns at 2.0 "finish"

$ns run
```

Makam.tcl

```
#Create a new simulator object
set ns [new Simulator]

set na [open prealloc.tr w]
set nf [open prealloc.nam w]
$ns trace-all $na
$ns namtrace-all $nf
set f0 [open prealloc-bw.tr w]
set fs [open prealloc-seq.tr w]

#colors for the different message types
$ns color 0 yellow
$ns color 100 blue
$ns color 2 green
$ns color 50 black
$ns color 46 purple
$ns color 3 red
$ns color 4 magenta

proc finish {} {
			global ns na nf f0 fs
			$ns flush-trace
			close $na
			close $nf
			close $f0
			close $fs
			exec nam prealloc.nam &
			exec xgraph -m prealloc-bw.tr -geometry 800x400 &
			exit 0
}

proc attach-expoo-traffic {node sink size burst idle rate} {
			global ns
			set source [new Agent/CBR/UDP]
			$ns attach-agent $node $source

			set traffic [new Traffic/Expoo]
			$traffic set packet-size $size
			$traffic set burst-time $burst
			$traffic set idle-time $idle
			$traffic set rate $rate

			$source attach-traffic $traffic

			$ns connect $source $sink
			return $source
}

# Define a procedure which periodically records the bandwidth received by the
# traffic sink sink0 and writes it to the file f0.
set totalpkt 0

proc record {} {
			global sink0 f0 totalpkt
			set ns [Simulator instance]

			#Set the time after which the procedure should be called again
			set time 0.065

			#How many bytes have been received by the traffic sink?
			set bw0 [$sink0 set bytes_]

			#Get the current time
			set now [$ns now]

			#Calculate the bandwidth (in MBit/s) and write it to the file
			puts $f0 "$now [expr $bw0/$time*8/1000000]"

			#Reset the bytes_ values on the traffic sink
			$sink0 set bytes_ 0
```

```
                #Re-schedule the procedure
                $ns at [expr $now+$time] "record"

                set bw0 [expr $bw0 / 200]
                set totalpkt [expr $totalpkt + $bw0]
}

proc recv-pkts {} {
                global totalpkt seqerrnb prvseqnb

                puts "The Number of Total sent packages are $prvseqnb"
                puts "The Number of Total received packages are $totalpkt"
                puts "The number of dropped packages are [expr $prvseqnb - $totalpkt]"
                puts "The Number of Total recieved unordered packages are $seqerrnb"
}

set prvseqnb -1
set seqerrnb 0
proc seq-record {size rate ftime} {
                global prvseqnb seqerrnb sink0 fs
                set ns [Simulator instance]

                #Set the time after which the procedure should be called again
                set tsize [parse-bw $size]
                set trate [parse-bw $rate]
                set time [expr double($tsize)/double($trate)/8.0]

                #Get the current time
                set now [$ns now]

                # seek the sequence number of packet.
                set revseqnb [$sink0 set expected_]

                if {$prvseqnb > $revseqnb} {
                                incr seqerrnb 1
                }

                # write the sequence number of packet to the file
                if {$prvseqnb != $revseqnb} {
                                puts $fs "$now [$sink0 set expected_]"
                                set prvseqnb $revseqnb
                }

                #Re-schedule the procedure
                if { [expr $now+$time] < $ftime } {
                                $ns at [expr $now+$time] "seq-record $size $rate $ftime"
                }
}

# routing protocol
$ns rtproto DV

#
# make nodes & MPLSnodes
#

set n0  [$ns node]

set n1  [$ns mpls-node]
set n2  [$ns mpls-node]
set n3  [$ns mpls-node]
set n4  [$ns mpls-node]
set n5  [$ns mpls-node]
set n6  [$ns mpls-node]
set n7  [$ns mpls-node]
set n8  [$ns mpls-node]
set n9  [$ns mpls-node]

set n10 [$ns node]


# Add RSVP-TE agents to all nodes
```

```
set rsvp1 [$n1 add-rsvp-agent]
set rsvp2 [$n2 add-rsvp-agent]
set rsvp3 [$n3 add-rsvp-agent]
set rsvp4 [$n4 add-rsvp-agent]
set rsvp5 [$n5 add-rsvp-agent]
set rsvp6 [$n6 add-rsvp-agent]
set rsvp7 [$n7 add-rsvp-agent]
set rsvp8 [$n8 add-rsvp-agent]
set rsvp9 [$n9 add-rsvp-agent]

# add variables for mpls modules

set LSRmpls1   [eval $n1 get-module "MPLS"]
set LSRmpls2   [eval $n2 get-module "MPLS"]
set LSRmpls3   [eval $n3 get-module "MPLS"]
set LSRmpls4   [eval $n4 get-module "MPLS"]
set LSRmpls5   [eval $n5 get-module "MPLS"]
set LSRmpls6   [eval $n6 get-module "MPLS"]
set LSRmpls7   [eval $n7 get-module "MPLS"]
set LSRmpls8   [eval $n8 get-module "MPLS"]
set LSRmpls9   [eval $n9 get-module "MPLS"]

#
# make links
#

$ns duplex-rsvp-link $n0 $n1 10Mb 1ms 0.99 1000 10000 Param Null

$ns duplex-rsvp-link $n1 $n3 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n3 $n5 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n5 $n7 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n7 $n9 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n1 $n2 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n2 $n4 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n4 $n6 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n6 $n8 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n8 $n9 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n1 $n2 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n3 $n4 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n5 $n6 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n7 $n8 10Mb 1ms 0.99 1000 10000 Param Null

$ns duplex-rsvp-link $n9 $n10 10Mb 1ms 0.99 1000 10000 Param Null


# Create a traffic sink and attach it to the node node10
set sink0 [new Agent/LossMonitor]
$ns attach-agent $n10 $sink0
$sink0 clear

# Create a traffic source
set src0 [attach-expoo-traffic $n0 $sink0 200 0 0 5000k]
$src0 set fid_ 100
$ns color 100 magenta

# Enable upcalls on all nodes
Agent/RSVP set noisy_ 255


# Set re-route option to drop
$ns enable-reroute drop


# Start recording
$ns at 0.0 "record"
$ns at 0.3 "seq-record 200 5000k 2.0"

# The setup of working LSP
$ns at 0.0 "$LSRmpls1 create-crlsp $n0 $n9 0 100 1000 +400000 5000 32 1_3_5_7_9_"

# The setup of backup LSP
$ns at 0.2 "$LSRmpls1 reroute-prealloc $n0 $n9 $n10 0 1 100 2000 +400000 5000 32 1_2_4_6_8_9_"

# bind a flow to working LSP
```

```
$ns at 0.4 "$LSRmpls1 bind-flow-erlsp 10 100 1000"

# start sending traffic
$ns at 0.5 "$src0 start"

#Break link
$ns rtmodel-at 0.8 down $n5 $n7

#activate hello
$ns at 0.1 "$ns activate-rsvp-hello 0.005 3.5"

$ns at 1.8 "$src0 stop"

$ns at 2.0 "recv-pkts"
$ns at 2.0 "record"
$ns at 2.0 "finish"

$ns run
```

Best_Effort.tcl

```
#Create a new simulator object
set ns [new Simulator]

set na [open precalc.tr w]
set nf [open precalc.nam w]
$ns trace-all $na
$ns namtrace-all $nf
set f0 [open precalc-bw.tr w]
set fs [open precalc-seq.tr w]

#colors for the different message types
$ns color 0 yellow
$ns color 100 blue
$ns color 2 green
$ns color 50 black
$ns color 46 purple
$ns color 3 red
$ns color 4 magenta

proc finish {} {
			global ns na nf f0 fs
			$ns flush-trace
			close $na
			close $nf
			close $f0
			close $fs
			exec nam precalc.nam &
			exec xgraph -m precalc-bw.tr -geometry 800x400 &
			exit 0
}

proc attach-expoo-traffic {node sink size burst idle rate} {
			global ns
			set source [new Agent/CBR/UDP]
			$ns attach-agent $node $source
			set traffic [new Traffic/Expoo]
			$traffic set packet-size $size
			$traffic set burst-time $burst
			$traffic set idle-time $idle
			$traffic set rate $rate
			$source attach-traffic $traffic
			$ns connect $source $sink
			return $source
}

# Define a procedure which periodically records the bandwidth received by the
# traffic sink sink0 and writes it to the file f0.
set totalpkt 0

proc record {} {
			global sink0 f0 totalpkt
			set ns [Simulator instance]

			#Set the time after which the procedure should be called again
			set time 0.065

			#How many bytes have been received by the traffic sink?
			set bw0 [$sink0 set bytes_]

			#Get the current time
			set now [$ns now]

			#Calculate the bandwidth (in MBit/s) and write it to the file
			puts $f0 "$now [expr $bw0/$time*8/1000000]"

			#Reset the bytes_ values on the traffic sink
			$sink0 set bytes_ 0

			#Re-schedule the procedure
			$ns at [expr $now+$time] "record"
```

```
        set bw0 [expr $bw0 / 200]
        set totalpkt [expr $totalpkt + $bw0]
}

proc recv-pkts {} {
        global totalpkt seqerrnb prvseqnb

        puts "The Number of Total sent packages are $prvseqnb"
        puts "The Number of Total received packages are $totalpkt"
        puts "The number of dropped packages are [expr $prvseqnb - $totalpkt]"
        puts "The Number of Total recieved unordered packages are $seqerrnb"

}

set prvseqnb -1
set seqerrnb 0
proc seq-record {size rate ftime} {
        global prvseqnb seqerrnb sink0 fs
        set ns [Simulator instance]

        #Set the time after which the procedure should be called again
        set tsize [parse-bw $size]
        set trate [parse-bw $rate]
        set time [expr double($tsize)/double($trate)/8.0]

        #Get the current time
        set now [$ns now]

        # seek the sequence number of packet.
        set revseqnb [$sink0 set expected_]

        if {$prvseqnb > $revseqnb} {
                incr seqerrnb 1
        }

        # write the sequence number of packet to the file
        if {$prvseqnb != $revseqnb} {
                puts $fs "$now [$sink0 set expected_]"
                set prvseqnb $revseqnb
        }

        #Re-schedule the procedure
        if { [expr $now+$time] < $ftime } {
                $ns at [expr $now+$time] "seq-record $size $rate $ftime"
        }
}

# routing protocol
#$ns rtproto DV

#
# make nodes & MPLSnodes
#

set n0  [$ns node]

set n1  [$ns mpls-node]
set n2  [$ns mpls-node]
set n3  [$ns mpls-node]
set n4  [$ns mpls-node]
set n5  [$ns mpls-node]
set n6  [$ns mpls-node]
set n7  [$ns mpls-node]
set n8  [$ns mpls-node]
set n9  [$ns mpls-node]

set n10 [$ns node]


# Add RSVP-TE agents to all nodes


set rsvp1 [$n1 add-rsvp-agent]
set rsvp2 [$n2 add-rsvp-agent]
set rsvp3 [$n3 add-rsvp-agent]
set rsvp4 [$n4 add-rsvp-agent]
set rsvp5 [$n5 add-rsvp-agent]
set rsvp6 [$n6 add-rsvp-agent]
set rsvp7 [$n7 add-rsvp-agent]
set rsvp8 [$n8 add-rsvp-agent]
set rsvp9 [$n9 add-rsvp-agent]

# add variables for mpls modules

set LSRmpls1   [eval $n1 get-module "MPLS"]
set LSRmpls2   [eval $n2 get-module "MPLS"]
set LSRmpls3   [eval $n3 get-module "MPLS"]
set LSRmpls4   [eval $n4 get-module "MPLS"]
set LSRmpls5   [eval $n5 get-module "MPLS"]
set LSRmpls6   [eval $n6 get-module "MPLS"]
set LSRmpls7   [eval $n7 get-module "MPLS"]
set LSRmpls8   [eval $n8 get-module "MPLS"]
set LSRmpls9   [eval $n9 get-module "MPLS"]

#
# make links
#

$ns duplex-rsvp-link $n0 $n1 10Mb 1ms 0.99 1000 10000 Param Null

$ns duplex-rsvp-link $n1 $n3 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n3 $n5 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n5 $n7 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n7 $n9 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n1 $n2 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n2 $n4 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n4 $n6 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n6 $n8 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n8 $n9 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n1 $n2 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n3 $n4 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n5 $n6 10Mb 1ms 0.99 1000 10000 Param Null
$ns duplex-rsvp-link $n7 $n8 10Mb 1ms 0.99 1000 10000 Param Null

$ns duplex-rsvp-link $n9 $n10 10Mb 1ms 0.99 1000 10000 Param Null


# Create a traffic sink and attach it to the node node10
set sink0 [new Agent/LossMonitor]
$ns attach-agent $n10 $sink0
$sink0 clear

# Create a traffic source
set src0 [attach-expoo-traffic $n0 $sink0 200 0 0 5000k]
$src0 set fid_ 100
$ns color 100 magenta

# Enable upcalls on all nodes
Agent/RSVP set noisy_ 255

# Set re-route option to drop
$ns enable-reroute drop



# Start recording
$ns at 0.0 "record"
$ns at 0.3 "seq-record 200 5000k 2.0"

# The setup of working LSP
$ns at 0.0 "$LSRmpls1 create-crlsp $n0 $n9 0 100 1000 +400000 5000 32 1_3_5_7_9_"

# The setup of backup LSP
$ns at 0.2 "$LSRmpls1 reroute-precalc $n0 $n9 $n10 0 100 2000 +400000 5000 32 1_2_4_6_8_9_"


# bind a flow to working LSP
```

```
$ns at 0.4 "$LSRmpls1 bind-flow-erlsp 10 100 1000"

#start traffic sending
$ns at 0.5 "$src0 start"

#break link
$ns rtmodel-at 0.8 down $n5 $n7

#activate hello
$ns at 0.1 "$ns activate-rsvp-hello 0.005 3.5"


$ns at 1.8 "$src0 stop"

$ns at 2.0 "recv-pkts"
$ns at 2.0 "record"
$ns at 2.0 "finish"

$ns run
```

APPENDIX C : Ns-2 modifications

This is some of the modifications that had to bee done to implement failure detection by RSVP-TE hellos in ns-2. For the full implementation a patch is available for download from [68].

Ns-mpls-node.tcl

```
RtModule/MPLS instproc activate_rsvp_hello {Dest Hinterv Dinterv} {
                global ns
                set class [$self set classifier_]
                set node1 [$class set mpls_node_]
                set agent1 [$node1 set rsvp_]
                set node1_ID [$node1 set id_]
                set dest_ [$Dest set id_]

                $agent1 start-hello $dest_ $Hinterv
                $agent1 hello_check_failure $dest_ $Hinterv $Dinterv $node1_ID

}

Agent/RSVP instproc reschedule_hello_request {Dest H_interv} {

                global ns
                $ns at [expr [$ns now] + $H_interv] "$self start-hello $Dest $H_interv"
}

Agent/RSVP instproc hello_check_failure {Dest Hintervall Dintervall my_id} {

                global ns
                set intervall [expr $Hintervall * $Dintervall]
                set check_time [expr [$ns now] + $Hintervall * $Dintervall]
                $ns at $check_time "$self hello_check $Dest $intervall $my_id"

}

Agent/RSVP instproc reschedule_hello_check {Dest interv my_id} {

                global ns
                set check_time [expr [$ns now] + $interv]
                $ns at $check_time "$self hello_check $Dest $interv $my_id"
}

Agent/RSVP instproc hello_set_lib_error {from_node to_node lspid} {
                global ns
                puts "Breaking connection for LSP $lspid at [$ns now]"
                $self instvar node_
                set MPLSnode [eval $node_ get-module "MPLS"]
                $ns at [$ns now] "$MPLSnode set-lib-error-for-lspid $lspid 1"
}

Simulator instproc activate-rsvp-hello {hello_int fail_int} {

                $self instvar linked_mplsnodes_
                set len [llength $linked_mplsnodes_]

                 for {set i 0} {$i < $len} {incr i} {

                                set me   [lindex $linked_mplsnodes_ $i]
                                set class [$me set classifier_]
                                set node1 [$class set mpls_node_]
                                set agent1 [$me set rsvp_]
                                set node1_ID [$me set id_]

                                set mpls_node1 [eval $node1 get-module "MPLS"]
```

```
        set neighborlist [$mpls_node1 get-mpls-neighbor]
        set neighborlen [llength $neighborlist]

        set used ""

        for {set k 0} {$k < $neighborlen} {incr k} {

            set peer [lindex $neighborlist $k]
            if {[lsearch $used $peer] == -1} {
                set dest_ [$peer set id_]

                #calls the function in agent1
                $agent1 start-hello $dest_ $hello_int
                $agent1 hello_check_failure $dest_ $hello_int
                $fail_int $node1_ID
                set used [lappend used $peer]
                }

        }
     }
}


Agent/RSVP instproc break-link-hello {nodo2} {
    global ns
    set nodo1 [$self set node_]
    set n1 [$nodo1 set id_]
    set agent1 [$nodo1 set rsvp_]
    $agent1 failure $nodo1 $nodo2

    $ns instvar linked_mplsnodes_
    set len [llength $linked_mplsnodes_]
    for {set i 0} {$i < $len} {incr i} {
        set nn [lindex $linked_mplsnodes_ $i]
        set nn_ID [$nn set id_]
        if { $nn_ID == $nodo2 } {
        set agent2 [$nn set rsvp_]

        }
    }
    $agent2 fail $n1
}
```

RSVP-objects.h

```
class HELLO : public RSVPobject {
public:
  HELLO(long src, long dst,int ctype);
  HELLO(unsigned char *cont);
  virtual ~HELLO() {};
  inline long get_src_inst() { return con->src_inst;}
  inline long get_dst_inst() { return con->dst_inst;}
  void dump_object();
private:
  struct construct {
    long src_inst;
    long dst_inst;
  };
  struct construct *con;
};
```

RSVP-objects.cc

```
HELLO::HELLO(long src, long dest, int ctype) {
  int cl;
  cl = sizeof(struct objheader) + sizeof(struct construct);
  contents = new unsigned char[cl];
  header = (objheader *)contents;
  con = (construct *)(contents + sizeof(struct objheader));
  header->ctype = ctype; // 1 is for REQUEST , 2 is for ACK
  header->length = 12;
  con->src_inst = src;
  con->dst_inst = dest;
  header->classnum = 22;
  header->conlength = cl;

}

HELLO::HELLO(unsigned char *cont) {
  header = (objheader *)cont;
  contents = new unsigned char[header->conlength];
  memcpy(contents, cont, header->conlength);
  header = (objheader *)contents;
  con = (construct *)(contents + sizeof(struct objheader));
}

void HELLO::dump_object() {
  RSVPobject::dump_object();
  printf("    HELLO:  src instance: %ld  dst instance: %ld\n",
                              con->src_inst, con->dst_inst);
}
```

RSVP.cc

```
if (strcmp(argv[1], "start-hello") == 0) {
  nsaddr_t destination;

  if (argc != 4) {
   return (TCL_ERROR);
  }
  destination = atoi(argv[2]);
  send_hello_message(destination, 1); // start by sending a REQUEST message

  //reschedule a new REQUEST message
  tcl.evalf("%s reschedule_hello_request %i %f", name(), atoi(argv[2]),atof(argv[3]));

  return (TCL_OK);
 }


 // Used to check if HELLO has detected a failure

 if(strcmp(argv[1],"hello_check") == 0) {
  nsaddr_t destination;
  nsaddr_t this_node;
  float intervall;

  hello_values *hello_req;
  hello_values *hello_ack;

  hello_req = hello_requested_;
  hello_ack = hello_acked_;

  double req_time;
  double ack_time;

  if(argc != 5) {
   return (TCL_ERROR);
  }

  destination = atoi(argv[2]);
  intervall = atof(argv[3]);
  this_node = atoi(argv[4]);

  if(hello_req != NULL){
   while((hello_req->node != destination) && (hello_req != NULL)){
               hello_req = hello_req->next;
   }
   if(hello_req->node == destination){
               ack_time = hello_req->acked_time;
   }
   else {
               printf("[HELLO] Something is wrong, didn't find instance value for node %i in node %i instance
list\n",destination,this_node);
   }

  }
  else {
   printf("[HELLO] Something is wrong, there is no instance values list for node % in node %i \n",destination,this_node);
  }

  if((Scheduler::instance().clock() - intervall) > ack_time ) { // did I loose the connection ?

   // set instance value for neighbor to 0
   hello_req->neighbor_instance = 0;
   // take a new instance value to send to this neighbor
   hello_req->my_instance = rand();

   elem *node_list;
   session *s_list;
   psb *p_list;

   s_list = s_list_;
   while (s_list != NULL) {
               p_list = s_list->psb_list;
               while (p_list != NULL){
                node_list = p_list->ero->get_ero();
                while (node_list != NULL) {
                 if((node_list->nodo == this_node) && (node_list->pun != NULL)){
                  if(node_list->pun->nodo == destination){
                          tcl.evalf("%s hello_set_lib_error  %i %i %i",name(),destination,this_node,p_list->tid);
                          tcl.evalf("%s break-link-hello %i", name(),destination);
                  }
                  else {break;}
                 }
                 node_list = node_list->pun;
                }
                p_list = p_list->next;
               }
               s_list = s_list->next;

  }

  //reschedule a new failure check
  tcl.evalf("%s reschedule_hello_check %i %f %i", name(), destination ,atof(argv[3]),this_node);

  return (TCL_OK);
 }
return (Agent::command(argc, argv));
}


void RSVPAgent::process_hello_message(RSVPmessage *msg, nsaddr_t fromhop) {

 hello_values *hello_list;
 long src_inst;
 long dst_inst;
 int c_type;
 int rand_int;

 HELLO *he = (HELLO *)msg->get_first(22);

 src_inst = he->get_src_inst();
 dst_inst = he->get_dst_inst();
 c_type = he->get_ctype();

 if(c_type == 1) { // This is a REQUEST message

  hello_list = hello_acked_; // get the instance list

  if (dst_inst == 0) { // is this the first message recieved or the sender has lost connection
                 // during the set time period.

   if(hello_list != NULL){
               while(hello_list->next != NULL){
                 if(hello_list->node == fromhop)
                  {break;}
                 else {
                  hello_list = hello_list->next;
                }
               }
               if(hello_list->node != fromhop) {
                hello_list->next = new hello_values();
                hello_list = hello_list->next;
                hello_list->neighbor_instance = src_inst;

                rand_int = rand();
                hello_list->my_instance = rand_int;
                hello_list->node = fromhop;
               }
               else {
                // this node has restarted
                hello_list->neighbor_instance = src_inst;
                hello_list->my_instance = rand(); // create a new instance for my self since i have restarted
               }
   }
   else // hello_list == NULL
               {
                hello_list = new hello_values();
                hello_list->neighbor_instance = src_inst;

                rand_int = rand();
                hello_list->my_instance = rand_int; // the instance value is a random number
                hello_list->node = fromhop;
                hello_acked_ = hello_list;
               }
  }
  send_hello_message(fromhop,2);  // send ACK message
 }

 else if(c_type == 2) { // recieved an ACK message

  hello_list = hello_requested_;

  if(hello_list == NULL){
   printf("Something is wrong, recieved an HELLO ACK object but there is no instance value for this node in the list\n");
  }
  else {
   while(hello_list->next != NULL) {
               if(hello_list->node != fromhop){
                hello_list = hello_list->next;
               }
               else {
                break;
               }
   }
   if(hello_list->node == fromhop){ // check if the node has restarted

               if(hello_list->neighbor_instance != src_inst){
                if(hello_list->neighbor_instance == 0){
                 hello_list->neighbor_instance = src_inst;
                }
                 else {
                  // printf("Node has restarted\n");
                  hello_list->neighbor_instance == src_inst;
                }
               }
               hello_list->acked_time = Scheduler::instance().clock(); // updates time for hello_requested
   }
  }
 }
}

void RSVPAgent::send_hello_message(nsaddr_t destination, int ctype) {

 if (nam_) {
  type_ = PT_HELLO;
 }

 int rand_int;
 hello_values *hello_sessions;

 if (ctype == 1) {
  hello_sessions = hello_requested_;
 }
 else if (ctype == 2){
  hello_sessions = hello_acked_;
 }

 long src_inst;
 long dst_inst;

 // save the values internally in the agent.

 if (hello_sessions == NULL){ // shall we make a new record
  hello_sessions = new hello_values();
  hello_sessions->node = destination;
  rand_int = rand();
  hello_sessions->my_instance = rand_int;
  hello_sessions->neighbor_instance = 0;
  if(ctype == 1){
   hello_requested_ = hello_sessions; // saves the newly created list object
  }
  }
  else {
    while (hello_sessions->next != NULL) {    // find this HELLO session
      if (hello_sessions->node == destination)
                        { break; }
      else
                        { hello_sessions = hello_sessions->next; }
    }

    if(hello_sessions->node != destination) {

      hello_sessions->next = new hello_values();
      hello_sessions = hello_sessions->next;
      hello_sessions->node = destination;

      if(ctype == 1){
                        hello_sessions->neighbor_instance = 0;
                        rand_int=rand();
                        hello_sessions->my_instance = rand_int;
      }
    }
   }//else (hello_sessions == NULL)

   if (ctype == 1) { // This is a REQUEST message

      src_inst = hello_sessions->my_instance;
      dst_inst = hello_sessions->neighbor_instance;
      hello_sessions->old_time = Scheduler::instance().clock();
      // printf("REQUEST src_inst : %i dst_inst : %i\n",src_inst,dst_inst);
    }

    if (ctype == 2) { // This is an ACK message

      src_inst = hello_sessions->my_instance;        // instance values for ACK
      dst_inst = hello_sessions->neighbor_instance;  // is created in the recieving function
      //printf("ACK src_inst : %i dst_inst : %i\n",src_inst,dst_inst);
    }

    // Create and send out the HELLO message

    RSVPmessage *msg = new RSVPmessage(HELLO_TYPE, 1); // ttl set to 1
    msg->add_object(new HELLO(src_inst, dst_inst, ctype));

    if (ip6_) {
      size_ = msg->get_length() + 40;
    } else {
      size_ = msg->get_length() + 24;
    }

    dst_.addr_ = destination;
    fid_ = 46; // needs to be 46 to be assosiated with RSVP
    Packet *pkt = allocpkt();
    pkt->allocdata(msg->get_conlength());
    memcpy(pkt->accessdata(), msg->get_contents(), msg->get_conlength());

    send(pkt, 0);

    delete msg;
}
```