ABSTRACT

ROUTE RECOVERY SCHEMES FOR LINK AND NODE FAILURE AND LINK CONGESTION

by
Ibrahim Takouna

Link/Node failure occurs frequently causing service disruption in computer networks. Hardware techniques have been developed to protect the network from Link/Node failure. These techniques work in physical layer, therefore their convergence time is very small. On the other hand, many schemes have been proposed to mitigate the failure influence on the network. These schemes work in upper layers such as the network layer. However, hardware solutions faster than other schemes, but they are expensive. Link/Node failure causes all flows which were using the failed link/node are temporarily interrupted till a new path reestablished.

Three recovery algorithms have been proposed that mitigate the changes occur in the network. These changes are link/node failure and link congestion. The algorithms mainly pre-compute a backup next hop for each destination in the network. This path is feasible to accommodate re-routed traffic when a failure occurs without causing congestion or loops. Simulations have been conducted to show the performance of the proposed algorithms using ns2 network simulation tool. The results show fast recovery for all flows were using the link/node failure. Furthermore, the throughput per node also increases due to decrease interruption service time.

ROUTE RECOVERY SCHEMES FOR LINK AND NODE FAILURE AND LINK CONGESTION

by
Ibrahim Takouna

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Engineering

Department of Electrical and Computer Engineering

August 2007

APPROVAL PAGE

ROUTE RECOVERY SCHEMES FOR LINK AND NODE FAILURE AND LINK CONGESTION

Ibrahim Takouna

08/08/07

Dr. Roberto Rojas-Cessa, Thesis Advisor — Date
Assistant Professor of Electrical and Computer Engineering, NJIT

8/8/07

Dr. Nirwan Ansari, Committee Member — Date
Professor of Electrical and Computer Engineering, NJIT

8/8/07

Dr. Edwin Hou, Committee Member — Date
Associate Professor of Electrical and Computer Engineering, NJIT

BIOGRAPHICAL SKETCH

Author:	Ibrahim Takouna
Degree:	Master of Science
Date:	August 2007

Undergraduate and Graduate Education:

- Master of Science in Computer Engineering,
  New Jersey Institute of Technology, Newark, NJ, 2007
- Bachelor of Science in Computer Engineering,
  Islamic University of Gaza, Gaza, Palestine, 2002

Major: Computer Engineering

In the name of Allah, Most Gracious, Most Merciful

“Are those who know equal to who know not” (Quran 39:9)

To my faithful wife, to my daughter Farah and my parents

ACKNOWLEDGMENT

I would like to express my deepest appreciation to Dr. Roberto Rojas-Cessa for his supervision, advice, and guidance from the very early stage of this research.

Special thanks are given to Dr.Nirwan Ansari and Dr. Edwin Hou for participating in my committee.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES
(Continued)

CHAPTER 1

INTRODUCTION

The number of Internet applications and services are growing very fast. These applications are extensions from science and business applications to personal communication applications. Most of the services are sensitive to both network failures and changes on network conditions, or they might require Quality of Service (QoS) guarantees. Consequently, failure recovery and network survivability has being studied extensively in recent years [1]-[11]. SDH/SONET automatic protection switching (APS) completes the recovery process within 50 ms [12]. This recovery time of tens of milliseconds satisfies most requirements. Network failure has been reported to occur very frequently in some networks. Figure 1.1 shows some statistics of the link failures in the Sprint IP backbone network as measured during seven months. Each single point at (t, l) is used to represent a failure on link l at time t.

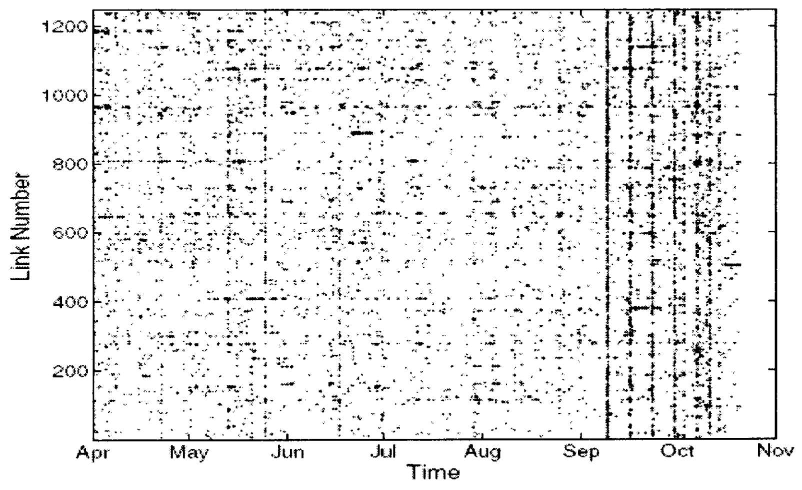


Figure 1.1 Characterization of failure in the Sprint IP backbone network [13].

This study of routing updates in the Sprint's IP backbone network [13] shows that 80% of all failures are unexpected. Of those, 70% affect a single link at a time while 30% involve shared link risk groups. These numbers shows the need of fast recovery techniques in the Internet. Planned link failures constitute the remaining 20%, can be handled by techniques such as graceful shutdown and ordered updates.

Resilience refers to the ability of a network to keep services running despite a link/node failure. For telephone networks, link/cable cuts are the largest cause of service interruption time. Resilient networks recover from a failure by repairing themselves automatically or circumventing a failure using additional resources. More specifically, failure recovery of a network can be achieved by rerouting traffic from the affected part of the network to another portion of the network. Rerouting is subject to several constraints. End-users may want rerouting to be fast enough so that the interruption time of services due to a link failure is either unnoticeable or minimal. The new path taken by rerouted traffic can be either computed at the time failures occurred reactive or before failures proactive. In the second case, rerouting is said to be pre-planned. Pre-planned rerouting mechanisms permit to decline interruption of service time but it requires additional hardware to provide redundancy in the network and consume valuable resources, such like computation cycles to find backup paths.

In this chapter, an overview of various network recovery techniques is presented. Section 1.1 presents re-routing of traffic after a link/node failure. Section 1.2 describes rerouting techniques at the physical and MAC layers.

Section 1.3 describes failure protection at the network layer, including an overview of IP Fast Re-Route (IPFRR) and a comparison of rerouting at lower layers with rerouting performed by the network layer without pre-planning and pre-planning. This section also includes a description of previous works.

1.1 Overview of Rerouting Schemes

This section presents a general overview of the rerouting concept. Rerouting schemes can be used in both circuit and packet switching networks. In either a link/node failure in a network or a congested link, traffic flows must change its route in order to reach its final destination in a timely fashion. Consequently, the service is interrupted for time $T_{int.service}$ as in Eq. 1.1. Service interruption time is the time since a failure occurs and gets detected until packets start using a backup path and the service is resumed. On the other hand, the transmission of packets is rerouted from a primary path to a backup path, which can be computed in advance before the failure occurred. Figure 1.2 depicts an example where a source node S sends traffic to a destination node D, and where a link on the primary path fails. The primary and the backup paths can be totally disjoint as in Figure 1.2.a or partially shared as in Figure 1.2.b.

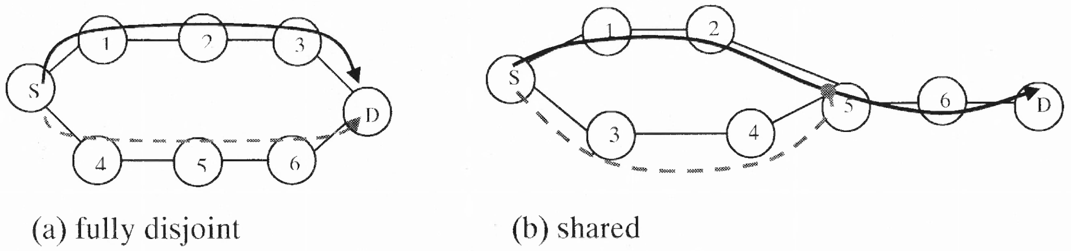


Figure 1.2 Backup path types.

$$T_{\text{int.service}} = T_{\text{detect}} + T_{\text{notif}} + T_{\text{switchover}} \quad 1.1$$

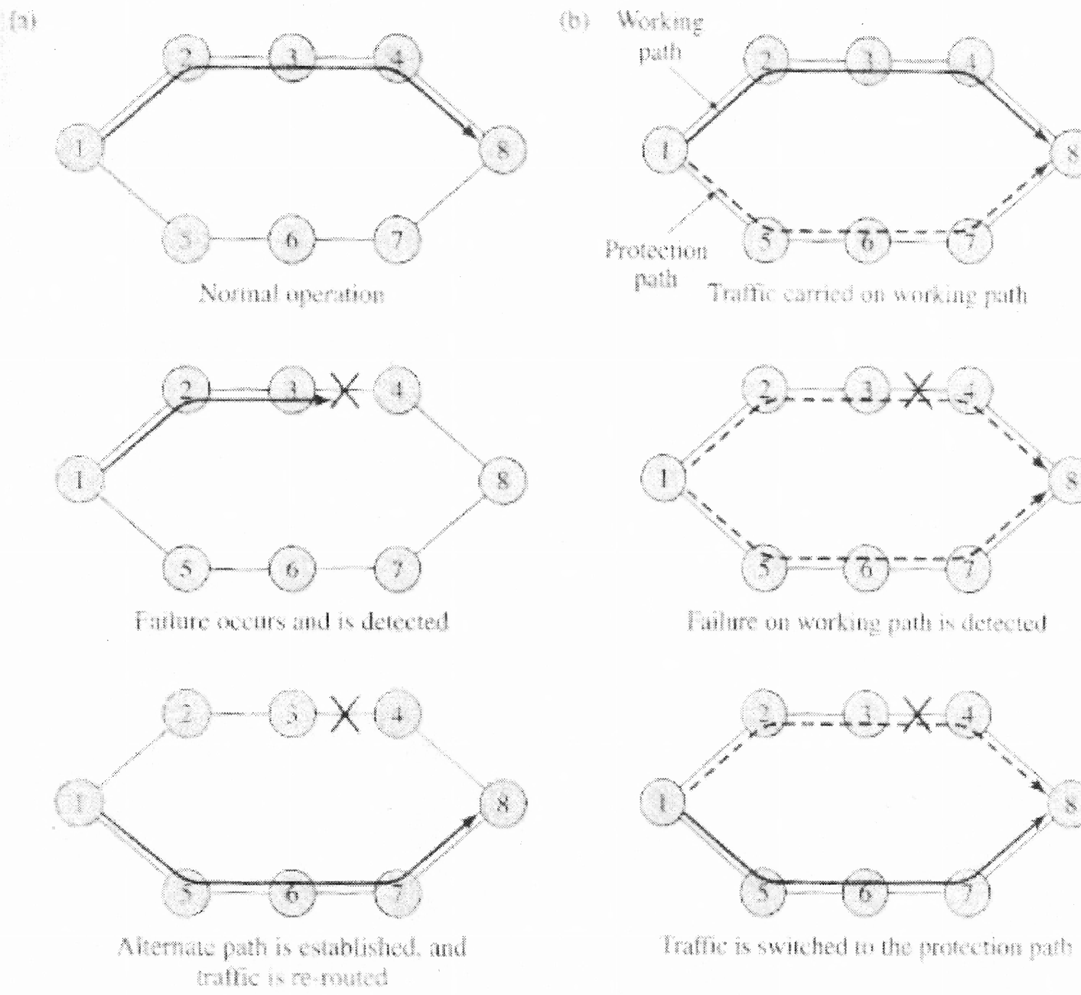

(a)
(b)
Working path
Protection path
Normal operation
Traffic carried on working path
Failure occurs and is detected
Failure on working path is detected
Alternate path is established, and traffic is re-routed
Traffic is switched to the protection path

Figure 1.3 Network survivability with (a) restoration (b) protection [14].

Figure 1.3 depicts two kinds of survivability restoration and protection techniques. The restoration of a new path is established after a failure occurred and the traffic is rerouted through the new path. In a protection technique, protection paths can be dedicated and pre-established. Protection paths are used as backup paths for the working paths. A rerouting technique consists of four general steps, where each one has a limited time for completion. These steps are the following. **First step**: the network must be able to detect link failures. Link failure detection can be performed by dedicated hardware or software at the end nodes or at the adjacency of the failed link. **Second step**: nodes that detect the link failure must notify the other nodes in the network of the failure. The selection of nodes that are actually notified of the failure depends on the rerouting technique used. **Third step**: a backup path must be computed proactively or reactively. In proactive rerouting schemes, however, this step is performed before link failure occurs. **Fourth step:** instead of sending traffic on the primary failed path, a node called path-switching node must send traffic on the backup path. This step is called switchover. Switchover completes the repairing of the routing path for continuation of the transmission after a link failure. When the failed link is repaired, traffic can be rerouted to the primary path, or keep being sent on the backup path, depending on the use requirements and network utilization.

1.2 Physical Layer Protection

A ring network is a network topology where all nodes are attached to the same set of physical links. Each link forms a loop. In counter-rotating ring topologies, all links are unidirectional and traffic flows in one direction on one half of the links, and in the reverse direction on the other half. Self-healing rings are particular rotating ring networks, which perform rerouting as follows. In normal operation, traffic is sent from a source to a destination in one direction only. If a link fails, then the other direction is used to reach the destination. Self-healing rings require expensive specific hardware and waste up to half of the available bandwidth to provide full redundancy. On the other hand, lower layer protection mechanisms are the fastest rerouting mechanisms available as self-healing rings can reroute traffic in less than 50 ms. This section presents four MAC and physical rerouting mechanisms that rely on a counter rotating ring topology: SONET UPSR and BLSR Automatic Protection Switching, FDDI protection switching, and RPR Intelligent Protection Switching. In SONET, protection with self-healing rings is called Automatic Protection Switching (APS [15]) and comes in two flavors. The first one, Unidirectional Path-Switched Ring Protection at the MAC and physical layers, as shown in Figure 1.4, is a self-healing ring network with benefits from 1+1 protection. In this 1+1 protection, two rings are used. Traffic is sent through both working and protection paths at ingress nodes. The destination receives the same data on each ring, but takes in from one ring only. In the event of a link failure, the receiver detects the increase of the bit error rate or the absence of traffic on one of the rings, and then decides to take in the traffic from the other ring. The SONET standards specify that the service interruption time should not exceed 50 ms [1], which is low enough for the outage to be unnoticeable by customers who participate in a live conversation where voice is carried over a SONET network. However, this technique requires detected backup paths for restoration, which is very costly.

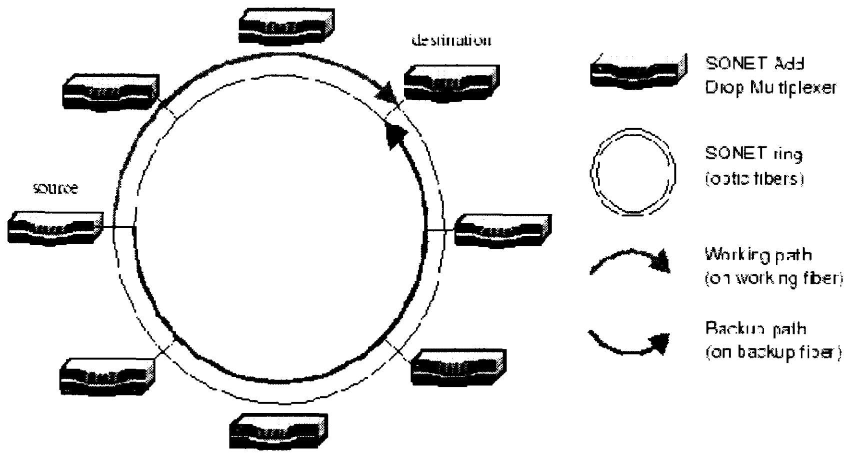

destination
source
SONET Add Drop Multiplexer
SONET ring (optic fibers)
Working path (on working fiber)
Backup path (on backup fiber)

Figure 1.4 SONET self-healing ring: Unidirectional Path-Switched Ring architecture. UPSR achieves 1+1 protection [16].

The second protecting scheme, Bidirectional Link-Switched Ring architecture (BLSR) is shown in Figure 1.5, benefits from 1:1 protection. In this 1:1 protection, traffic is sent to the working path during normal operation. On a link failure, the node upstream of the failed link wraps traffic from one ring to another ring in the reverse direction so that traffic still can reach its destination. BLSR is as fast as Unidirectional Path-Switched protection and does not waste as many resources. The MAC layer provides the means for IP to send packets over a local area network. Fiber Distributed Data Interface (FDDI) [17] implements a protection mechanism at the MAC layer that is similar to SONET BLSR. FDDI runs over dual rotating rings. In normal operation, traffic is sent on one ring only. Like BLSR, FDDI wraps paths when a link failure is detected and uses the second ring only as a backup ring. Therefore, FDDI implements 1+1 protection and requires full link redundancy. Resilient Packet Ring (RPR) is a more recent MAC protocol designed to run on multiple counter-rotating rings. In RPR, path protection is called Intelligent Protection Switching (IPS). IPS can be viewed as an enhanced SONET BLSR mechanism. Indeed, when a link failure occurs, traffic is first wrapped exactly like SONET BLSR does. The emitting node is notified of the failure and changes the ring on which it sends traffic. The new path taken by packets is therefore shorter than the wrapped path, resulting in both shorter delays for packets and a better utilization of the available resources. The lower layer rerouting mechanisms are fast because the nodes that detect the failure perform themselves the switchover step immediately, bypassing the notification step.

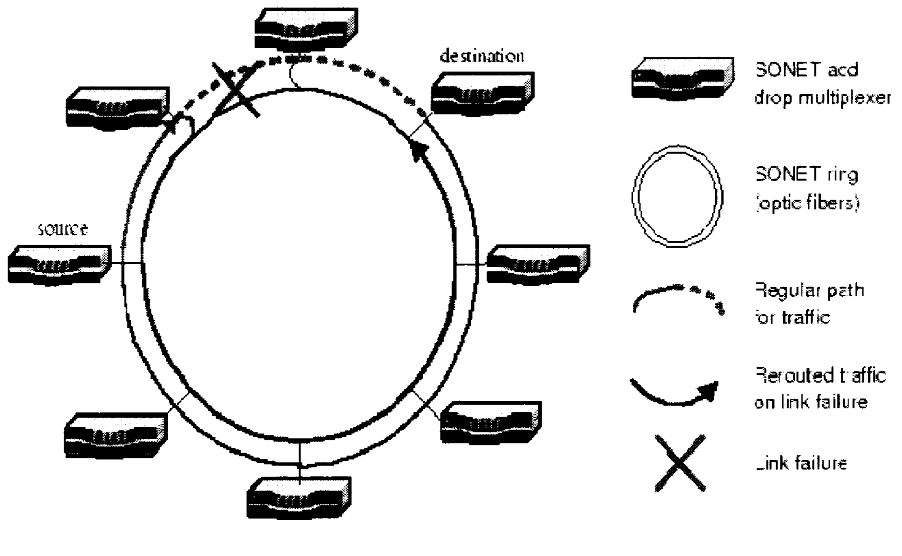

destination
source
SONET acd drop multiplexer
SONET ring (optic fibers)
Regular path for traffic
Rerouted traffic on link failure
Link failure

Figure 1.5 SONET self-healing ring: Unidirectional Path-Switched Ring architecture [16].

1.3 Network Layer Protection

Packet switching networks, like the Internet, are expected to be resilient to link/node failures. Routing protocols [18] [19] [20] [21] take into account topology changes such as a link failure and re-compute routing tables accordingly using a shortest path algorithm. When all routing tables of the network are recomputed and have converged, all paths that were using a failed link are rerouted through other links. However, convergence is slow and takes usually several tens of seconds. This long recovery time can potentially jeopardize the satisfaction of services and user requirements. Part of the reason for this long time is that routing protocols use timers to detect link failure with coarse granularity (1 second) making the T_{detect} term in Equation 2.2 large compared with lower layer rerouting mechanisms. Also, all routers in the network have to be notified of the failure. Propagating notification messages is done in an order of magnitude of tens of millisecond which makes T_{notif} negligible compared with T_{detect}. Indeed routers only need to forward the messages with no additional processing and routing tables have to be recomputed before paths are switched. Re-computing routing tables implies using CPU intensive e shortest path algorithms which can take a time $T_{switchover}$ of several hundred milliseconds in large networks. It is possible to perform faster IP rerouting by shrinking the T_{detect} and $T_{switchover}$ terms of Equation 1.1. In [22], first, they propose to use sub-second timers to detect failures and decrease the value of the T_{detect} term. Second, they suggest that routing convergence is slow due to the obsolescence of the shortest path algorithms employed in current routing protocols which would be able to recompute routing tables at the millisecond scale if faster, more modern algorithms were used. In summary, expected rerouting times in networks using modified routing protocols are below one second, but the authors also argue that millisecond network layer rerouting is achievable.

IPFRR is a proactive technique or pre-planned technique. The nodes of a network using IPFRR pre-compute a backup path to each destination before a failure occurs so that when a failure is detected, the routes are switched to backup paths without waiting to compute a new path to resume the service quickly. Consequently, the repair time is reduced and therefore, this leads to a reduction of interruption service time.

As mentioned in Section 1.1, the physical protection failure recovery is completed in less than 50 ms but it requires extra hardware and wastes recourses. Most IP networks are required a failure recovery time within a second. However some business applications require a failure recovery time shorter than 50 ms. Recently, it has been reported a reduced recovery time to one second in carefully configured networks using link state IGPs [22]. Figure 1.9 depicts a time comparison between proactive and reactive routing recovery schemes when link (3,4) fails. In Figure 1.9 (a), the backup path is computed before a real failure occurs so it saves recovery time. However, a proactive scheme computes the new path after a failure detected that takes some milliseconds.

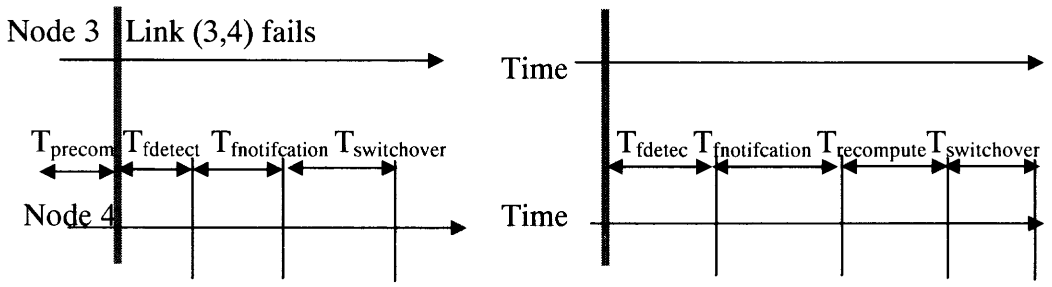


Figure 1.6 Service interruption time a) proactive (pre-planned) and b) reactive.

The following table shows the required time for each operation for normal recovery process. Updating routing table process takes most of recovery process time. As seen in the table, to update 500 prefixes takes 280 ms in the worst case, which is a significant amount of time compared with the others values.

Table 1.1 IGP Convergence [23]

The operation	**Time**
Failure Detection (SONET today)	20 ms
Origination	10 ms
Queueing, Serialization, Propagation	30 ms
Flooding < 5 * 2ms =	10 ms
– SPF	n * 40 us
FIB Distribution Delay:	50 ms
500 important prefixes: Worst-case:	280 ms

1.4 Link State Update Schemes

There are many studies try to improve link state update scheme. Some of these schemes are presented as following. First, some of these schemes are concerned about efficiency and reliability of dissemination of the link state. For reliability and to avoid flooding update, [24] has proposed a Tree-Based Reliable Topology

(TRT) to disseminate link state information. This scheme proposes a multiple tree based link state dissemination scheme. This scheme satisfies the reliability and efficiency but it cannot satisfy the requirement for a fast recovery time when a failure occurs. In [25], it has been proposed a dissemination scheme that uses broadcast for building the first tree. Then, this approach uses the tree to disseminate link state. The proposed approach alternates between flooding and tree-based broadcasting modes. In other words, when the link state dissemination is guaranteed, the time required for source node to update its routing table to recover a failure path cannot be guaranteed. Second, other studies intend to overcome link state information that causes the false routing, either positive or negative. A framework has been proposed in [26] to solve such problems that result from the dissemination of inaccurate or outdated link state information. The search of alternative paths have been proposed in [27][28]. The primary objective of multiple-path routing approach is to compensate for the inaccuracy of the knowledge accuracy available to routing nodes, but it cannot provide an alternative path for each flow in a network in all cases of failures. Third, researchers have proposed an update policy mechanism for link dissemination. In [29], it has been stated that routing without considering the staleness of link state information introduced by update policies may generate significant percentage of false routing.[29] presents the Least Cost Multiple Additively Constrained Path (LCMACP) scheme to mitigate the effect of staleness of link state. In [30], it been presented two different link-state update policies to advertise the availability of wavelength and converter resource in all-optical DWDM transport networks.

1.5 Related Work

IP Fast ReRoute IPFRR was proposed as framework in [31]. The simple scheme of IPFRR uses equal cost multi-paths (ECMP)[32], where at least two paths with the same cost are calculated for each source/destination pair. However, multipath routing with ECMP usually do not satisfy the minimal resilience requirement of at least two outgoing paths at each node [33]. For this reason, ECMP does not guarantee the network survivability. Failure insensitive routing (FIR) is presented in [34] for single-link failures. Given a primary path Source to Destination {S –to-D}, FIR identifies a number of key links such that removing any of these links forces the packets go back to S. Therefore, the failure of any key links can be inferred by S at a deflected packet. To provide an alternate path, FIR removes the key links and runs shortest path routing from S to D. Extending FIR to cover single-node failures is presented in [35]. In [36], a loop-free alternate path is presented where It choose an alternative backup path that satisfy this criteria cost (N, D) < cost (N, S) + cost(S,D). On other words, S should not be the next hop for N the backup next hop, but this is not guarantee the recovery process. Finally, a scheme is proposed to set up a tunnel from node S to node Y that is multiple hops away [37]. This scheme adds extra packet processing where all packets are encapsulated by S and routed towards the tunnel endpoint. Then, the tunnel endpoint decapsulates the packets and forwards them according to its routing table. Multiple routing configuration (MRC) algorithm has been presented in [38]. In this work, the approach in each node has multiple configurations or a multi-routing table. Then, when a failure occurs and is detected, the node tries to find suitable configuration to overcome this failure. Managing the multiple routing tables is significant overhead issue here.

1.6 Contributions of this Thesis

Many studies have been investigated the failure recovery either link or node. This thesis addresses some of the problems related to failure recovery and congestion recovery.

For these, this thesis proposes three algorithms.

- First, an algorithm for link failure recovery using proactive routing against failures is proposed. When a node detects a failure, the pre-calculated alternative paths make the recovery faster.
- Second, a proactive algorithm for node failure recovery is proposed. This scheme uses some of the ideas presented for link failure but it considers the differences on the effects created by node failures that are not present with link failure.
- Third, a proactive congestion mitigation algorithm is also proposed.

This thesis provides the complexity analysis and implementation discussion of the algorithms, and present simulations to test their performance under different failures. The simulations are performed by using ns2 [39] simulation software.

This thesis is structured as follows. Chapter 2 introduces three proposed algorithms for self-recovery networks from link and node failures, and for congestion mitigation. Chapter 3 presents simulations and results. Chapter 4 contains the conclusions and future work.

CHAPTER 2

ROUTING TREE RECOVERY ALGORITHM

In this chapter, three algorithms have been proposed for data routing tree recovery. The proposed algorithms aims to recover the transmission of packets quickly and effectively from link failure, node failure, and link congestion. The scheme is based on proactive computation of a backup path for each network destination. Each obtained backup path is capable of accommodating rerouted traffic and of being loop free. A modified Breadth First Search (BFS) algorithm [33] is used to determine the backup path and several conditions to select eligible links (paths).

The rest of this chapter is organized as follows. Section 2.1 proposes a link failure recovery algorithm that pre-computes the backup path. Section 2.2 proposes a route recovery algorithm for node failure. Section 2.3 presents a link congestion mitigation algorithm. Section 2.3 discusses some implementation issues, such as routing table extension. Section 2.4 discusses the algorithm time complexity for the three proposed algorithms.

2.1 Link Failure Recovery Algorithm

In this section, an algorithm that aims at determining a backup path is described, which is feasible to accommodate the rerouted traffic result from a link failure. The backup path is pre-computed before a failure occurred. A primary port used to forward data through routing tree or down stream tree for any node.

However, when a failure occurs, a subset of these nodes switch to their backup ports for fast rerouting, and the routing tree is updated according to the used backup ports.

The following is a table with the notation used to describe the proposed algorithms.

Table 2.1 Link Failure Recovery Algorithm Notation

$G(V,E)$	Network with V nodes and E edges
P_n	Primary port of node n before any failure
$b_n^{i,j}$	Backup port of node n when link (i,j) fails
λ_{n0}	Traffic generated in node n.
Ci,j	Link(i, j) capacity
$\lambda_{Tc^u_i}$	The traffic destined to sub-tree T c^u_i.
$\lambda_{i,j}$	The traffic in link (i ,j) or link utilization
U	The total traffic in the network or utilization
$T_{rerouted}$	The total rerouted traffic
BP_{total}	The total number of backup port when a link fails

$$\mathrm{T}_{\text{rerouted}} = \lambda_u + \sum_i \lambda_T u_i \qquad 2.1$$

$$\lambda_{i,j} < c_{i,j} \qquad 2.2$$

Minimize:

$$U = \sum_{i,j} \lambda_{i,j} \tag{2.3}$$

$$BP_{total} = \sum_{n \in V} b_n^{i,j} \tag{2.4}$$

In addition, it is required to find a feasible path capable to accommodate the rerouted traffic without causing additional congestion in others links in the tree. Equation 2.2 guarantees that link(i,j) utilization is less than link(i,j) capacity. The second constrain is to minimize overall network traffic, which means that the selected path is used to reroute traffic and be the shortest path to the destination. The third constrain is to minimize number of switchover nodes.

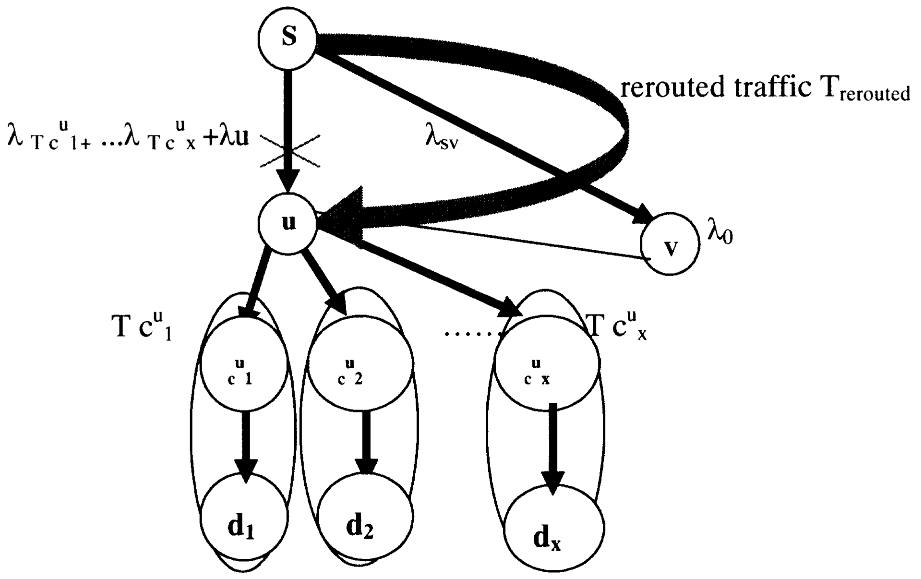

S
rerouted traffic $T_{rerouted}$
$\lambda_{Tc^u_1} + \dots \lambda_{Tc^u_x} + \lambda u$
λ_{sv}
u
v
λ_0
$T c^u_1$
c^u_1
c^u_2
c^u_x
$T c^u_x$
d_1
d_2
d_x

Figure 2.1 Data Routing Tree for source node S.

After determining the routing table for node S, the routing information would look similar to the contents in Table 2.1. By looking at the destination and the next hop columns, u node is the next hop for all the destination nodes in the sub-trees $T\,c^{u}_{1}$, $T\,c^{u}_{2}$,..... $T\,c^{u}_{x}$. So when node S assumes that its next hop is fail, it tries to find a backup path or backup next hop. The mean point of our algorithm 1 is to determine a backup next hop node before the primary next hop become unreachable.

Table 2.2 Sample of Routing Table for Node S

Destination	**Path Type**	**Cost**	**Next hop(s)**
U			u
V			v
$c^{u}\ 1$			u
$c^{u}\ 2$			u
$c^{u}\ x$			u
d_1			u
d_x			u

Definition 1 The backup path is a path capable to accommodate the rerouted traffic from the source to the new next hop.

Algorithm 1 Link Failure Recovery

Step 1: Initialization: Set the backup ports for node S to null.

Step 2: Mark children nodes (next hop nodes for each destination) u={1,2,, m} as unreachable and doing the following:

Step 2.1: Color all nodes in sub tree T(u) black, unreachable node u as red, and the other nodes in the topology white "where the forwarding path is not affected by failure"

Step 2.2: Compute how much is traffic should be rerouted ($T_{rerouted}$)

Step 2.3: Compute a back up path by using BFSpath(G, S, u , $T_{rerouted}$) feasible to accommodate the rerouted traffic $T_{rerouted}$ to reconnect the black and red nodes to the main tree T(S)

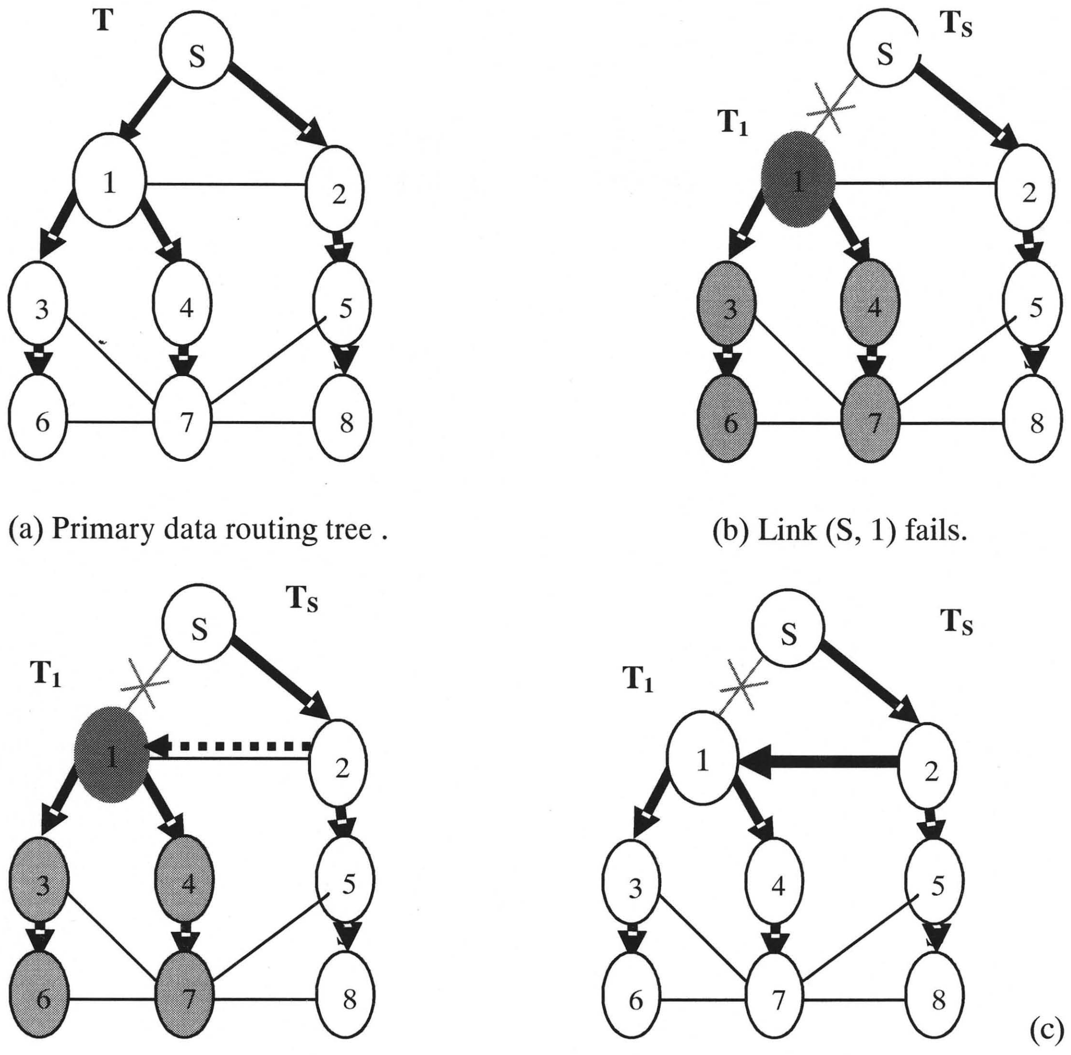
Step 2.4: Set the backup port according the discovered path from S to u

Step 2.5: Colors the recovered nodes to white.

This algorithm will be explained by giving three examples in the following.

Case I

Figure 2.2 (a) shows the primary data routing tree T for node S. In Figure 2.2 (b), it is assumed that the link between nodes S and 1 failed and the nodes are colored according to step 2.1. After link(S, 1) fails, two disconnected sub-trees T_1 and T_s were formed resulting from the primary tree T. Then in step 2.2 node S compute how much traffic should be reroute. Step 2.3 returns a feasible path that could accommodate the rerouted traffic as in Figure 2.2 (c). Finally in step 2.4 the node S set the back up port according the returned path as shown in Figure 2.2 (d) the next hop will be node 2 for all nodes in the sub-tree T_1 when actual failure occurs instead of node 1.



(a) Primary data routing tree .

(b) Link (S, 1) fails.

(c)

The next hop to the disconnected sub-tree. (d) Survivable node white.

Figure 2.2 Example of Link failure Recovery Algorithm-Case I.

The example in Figure 2.2 shows that node S switches its data to the backup port for each affected destination. Meanwhile, all other nodes keep using their original ports. In this way, the routing tree survives using all nodes in the disconnected sub-tree T_1 and the algorithm reduces the number of nodes that need to be changed for recovery. The routing table should be extended as in Table 2.2. An extra column is added specify the backup next hop.

Table 2.3 Extended Routing Table for Node 1 Case I.

Destination	Path Type	Cost	Next hop(s)	backup hop(s)
S			-	-
1			1	2
2			2	2
3			1	2
4			1	2
5			2	2
6			1	2
7			1	2
8			2	2

Claim 1. Given a primary tree for node S T=(V_T, E_T), and two nodes i, j V_T, the primary path PPi,j a backup path BPi,j, and b is a link(i,j). b $\in$ E ppi,j, there exists a tree T'=($V_{T'}$,$E_{T'}$) such that

1-V_T= $V_T \cup V_{BPi,j}$

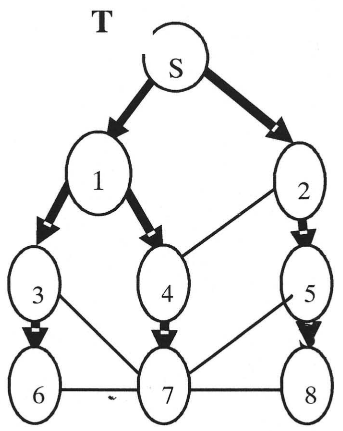
2- E_T={$E_T \cup E_{BPi,j}$} \{b}

Proof:

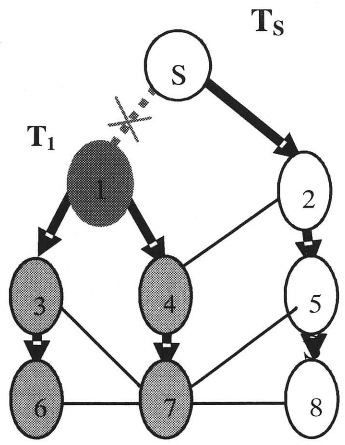
Consider T_1=(V_{T1},E_{T1}) and T_S=(V_{Ts},E_{Ts}) two trees results from a link b fails in the primary tree T. Trees T and T_S are rooted at node S. Assume, without losing generality, that j $\in$ V_{T1} and i V_{Ts}. Let i be the root of tree T_S. Trees T_1 and T_S are link and node disjoint therefore the graph T=($V_T \cup V_{BPi,j}$), { $E_T \cup E_{BPi,j}$} \{b} When link b fails, the traffic for sub-tree T_1 that was using the primary path $PP_{i,j}$ is rerouted to $BP_{i,j}$ without make a loop a recovery all the nodes affected by failure, hence (1) and (2).

Case II

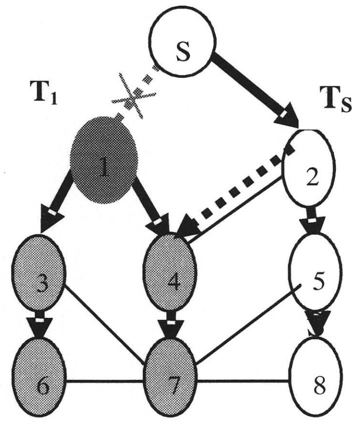
In first case, previous example, a direct link was between the next hop of T_1 and the root of sub-tree T_S. However, in second case, the next example shows how the algorithm works if there is not a direct link between T_1 and T_S. The same steps like the previous example. Figure 2.3 (a) shows the primary data routing tree for node S. Node S assumes that the next hop, Node 2, is unreachable. The nodes are colored according to Step 2.1. After link(S,1) fails, there are two disconnected sub-trees T_1 and T_S as result from the original tree T. Then in Step 2.2, node S computes how much traffic needs to be rerouted. Step 2.3 returns a feasible path that can accommodate the rerouted traffic, as Figure 2.3(c) shows. In Step 2.4, the node S sets the back up port according the returned path as shown in Figure 2.3 (d). The next hop is Node 2 for all nodes in the sub-tree T_1 when actual failure occurs instead of Node 1. However, as shown Figure 2.3 (d), Node 4 has two sources, one from Node 1 and the other from Node 4 after the algorithm terminates.



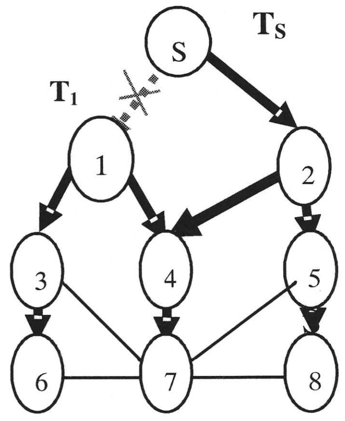
(a) Primary data routing tree.



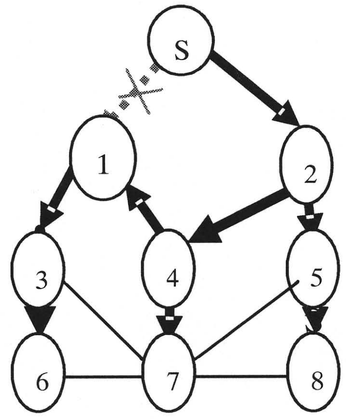
(b) Link (S, 1) fails.



(c) The next hop to the disconnected sub-tree.



(d) Survivable node white.



(e) Actual recovery routing tree

Figure 2.3 Example of Link Failure Recovery Algorithm-Case II.

In this case, Node 4 does not have two sources, because Node 1 has no more node as next hop for node S to forward traffic for Node 4 and the new next hop for Node 4 is Node 2 after a link (S, 1) fails. For Node 1 to receive its data from node S work as follows: when the data packets arrive to Node 4, this node is the responsible for forwarding the traffic for Node 1 only. Therefore, the new recovery routing tree is as seen in Figure 2.3 (e) and the recovered T1, which was rooted by Node 1, becomes rooted by Node 4. As observed, in this algorithm, the routing table of other nodes can be used to assist in the recovery of the main routing tree for any node. In Table 2.3, the routing table for Node 4 without backup hops. Therefore, when node S detects a failure in link (S,1) then it switches the use of its original port to its backup port. All the destinations that are using as next hop Node 1 will switch to Node 2. For example, the path S→1 before the failure for destination Node 1 occurs, and then after the failure occurs, the new path is S→2→4→1, as shown in Figure 2.3 (e). The number of hops for destination Node 1 after failure is increased by 2 hops. For destination Node 4, the path before the failure is S→1→4, then after failure, the new path is S→2→4. Here, the number of hops is the same before and after the failure.

Table 2.4 Routing Table for Node 1 in Case II

Destination	Path Type	Cost	Next hop(s)	Backup hop(s)
S			-	-
1			1	2
2			2	2
3			1	2
4			1	2
5			2	2
6			1	2
7			1	2
8			2	2

Table 2.5 Routing table for Node 4 in Case II

Destination	Path Type	Cost	Next hop(s)	backup hop(s)
1			1	-
4			4	-
7			7	-

Case III

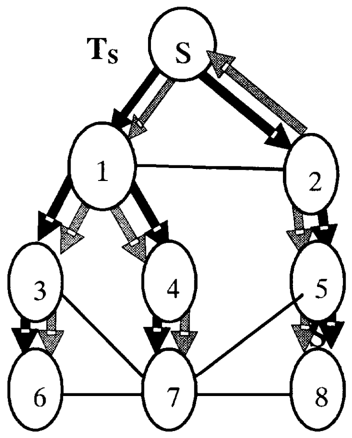
This case describes the case of how loops are avoided by the recovery algorithm. The following example explains this case. Consider having two primary trees, one for Node S colored black and a routing table as in Table 2.5 and the other tree for Node 2 colored gray and a routing table as Table 2.6. The problem here is when Node S selects Node 2 as the new next hop after link (S,1) fails and a loop could result. For example, the path for destination Node 1 before failure is S→1, and after failure would be S→2→S→2, becoming a loop. By applying the forward policy in Figure 2.5, when Node 2 receives a packet, it decides its next hop for a destination, for example Node 1, Node 2 considers the next hops for these destinations unreachable and enables the backup next hop only for these destinations. Therefore, flows destined from Node 2 to Node S still have the same primary path. In other hand, the flow generated from node 2 and destined to node 1 according to primary tree for node 2 the next hop for this flow is node S, but node S select node 2 as a backup next hop when considered link S,1 fails. At this way, a loop is formed S→2→S→2. According to forward policy in figure 2.5 when node 2 receive packet from node S and find that packet just forwarded by itself. Then, node 2 considers the next hop for this flow- destination node 1- is failed and switch to its backup next hop. Similarly, all flows next hop are S and make a loop will switch to their backup next hop. Meanwhile, flows from node S to destination node 2 after link S,1 fails is S→2→1, without any loop.

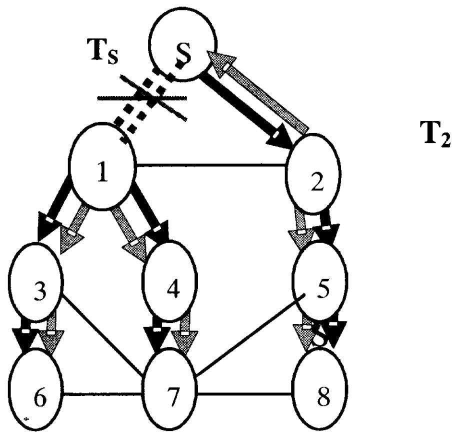
Table 2.6 Routing table for Node S in Case III

Destination	Path Type	Cost	Next hop(s)	Backup hop(s)
S			-	-
1			1	2
2			2	2
3			1	2
4			1	2
5			2	2
6			1	2
7			1	2
8			2	2

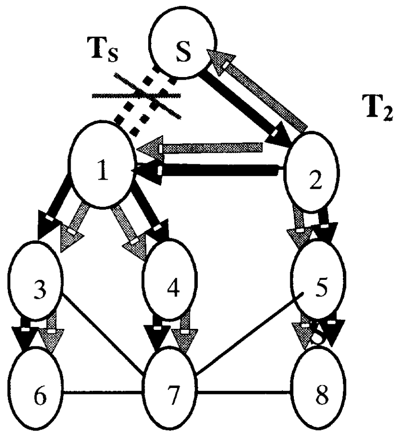
Table 2.7 Routing table for Node 2 in Case III

Destination	Path Type	Cost	Next hop(s)	backup hop(s)
S			S	-
1			S	1
2			-	-
3			S	1
4			S	1
5			5	
6			S	1
7			S	1





(a) Primary routing tree for Nodes S and 2. (b) Link (S,1) fails and splits the tree.



(c) Survivable trees for node S and 2.

Figure 2.4 Example of the Link Failure Recovery Algorithm-loop free Case III.

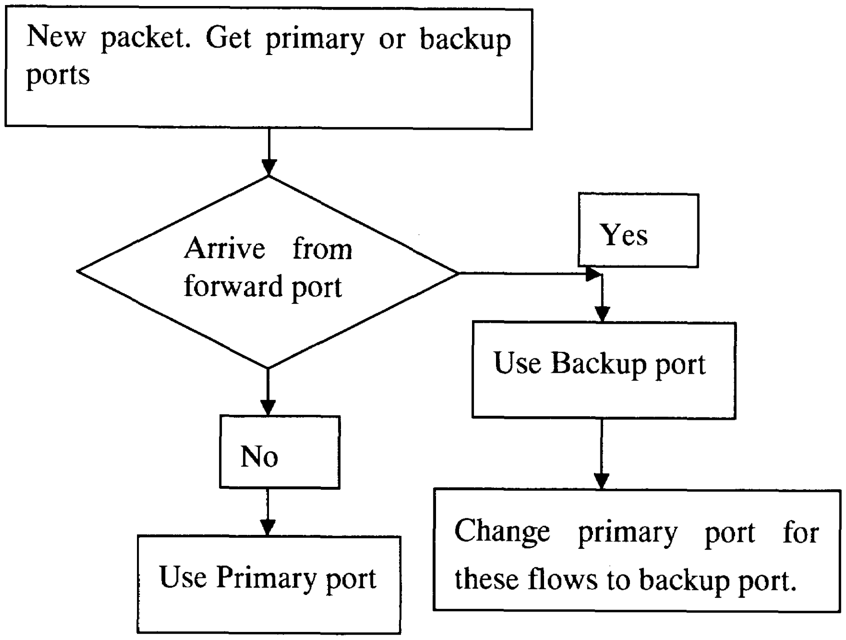

New packet. Get primary or backup ports
Arrive from forward port
Yes
Use Backup port
No
Change primary port for these flows to backup port.
Use Primary port

Figure 2.5 Packet Forwarding Policy.

2.2 Node Failure Recovery Algorithm

In this section introduces an algorithm that aims at determining a backup path. A path is feasible to accommodate the rerouted traffic result from a node failure. The backup path also pre-planned before a failure occurred. The node recovery is different from link recovery as this is more complex than link failure recovery. Figure 2.3 shows how traffic is rerouted and the dependency that this has on the number of sub-tree at Node u.

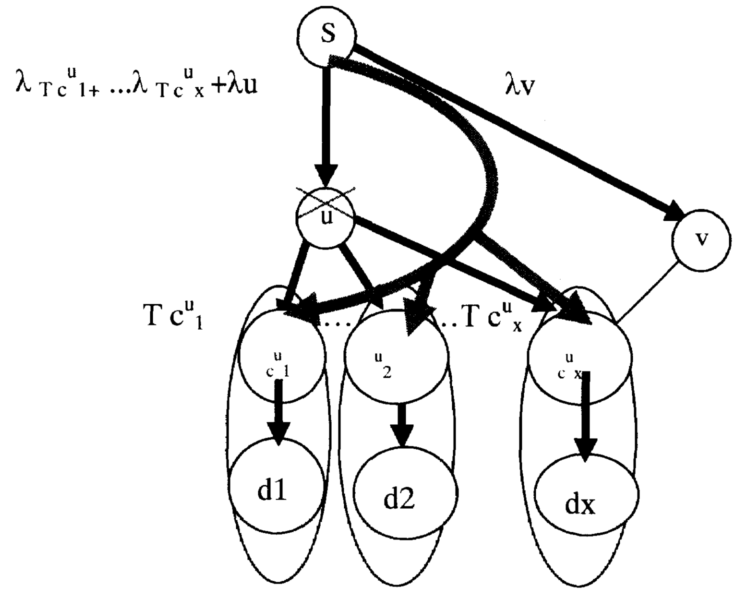


Figure 2.6 Example of Node Failure.

Table 2.8 Node Failure Recovery Algorithm Notation

G(V,E)	a network with V nodes and E edges
P_n	primary port of node n before any failure
$b_n^{i,j}$	backup port of node n when link (i,j) fails
λ_{n0}	traffic generated in node n.
Ci,j	link(i,j) capacity
$\Lambda_{Tc^u_i}$	The traffic destined to sub-tree T c^u_i.
$\lambda_{i,j}$	the traffic in link (i ,j) or link utilization
U	the total traffic in the network or utilization
$T_{rerouted}$	the total rerouted traffic
BP_{total}	the total number of backup port when a link fails

$$\mathrm{T}_{\text{rerouted}} = \lambda_u + \sum_i \lambda_T u_i \quad 2.5$$

$$\lambda_{i,j} < c_{i,j} \quad 2.6$$

Minimize:

$$U = \sum_{i,j} \lambda_{i,j} \quad 2.7$$

$$BP_{total} = \sum_{n \in V} b_n^{i,j} \quad 2.8$$

Two constrains have to be satisfied. The first one is to minimize the overall traffic in network after the traffic is rerouted, as in Equation 2.6. Choosing the shortest path to destination decreases the number of links that the rerouted traffic go through, consequently, it decreases the overall network utilization. The second constrain is to minimize number of used backup ports for stability of the routing tree.

As shown in Figure 2.6, Node u, which is assumed failed, has x children, each of which is denoted as c^u_1,c^u_2,.... c^u_x, where each child is the root for sub-trees T c^u_1 ,...., T c^u_x. Here, the goal is to bypass the failed node u to its children c^u_1,c^u_2,..., c^u_x, that satisfy the two constrains.

Algorithm 2 Node failure recovery algorithm

Step1: Initialization: Set the backup ports for node to null.

Step2: Mark the children nodes (next hop nodes for each destination) u={1,2,, m} as unreachable and do the following:

Step2.1: Color all nodes in sub tree T(u) black, failed node u as red, and the other nodes in the topology white where the forwarding path is not affected by failure.

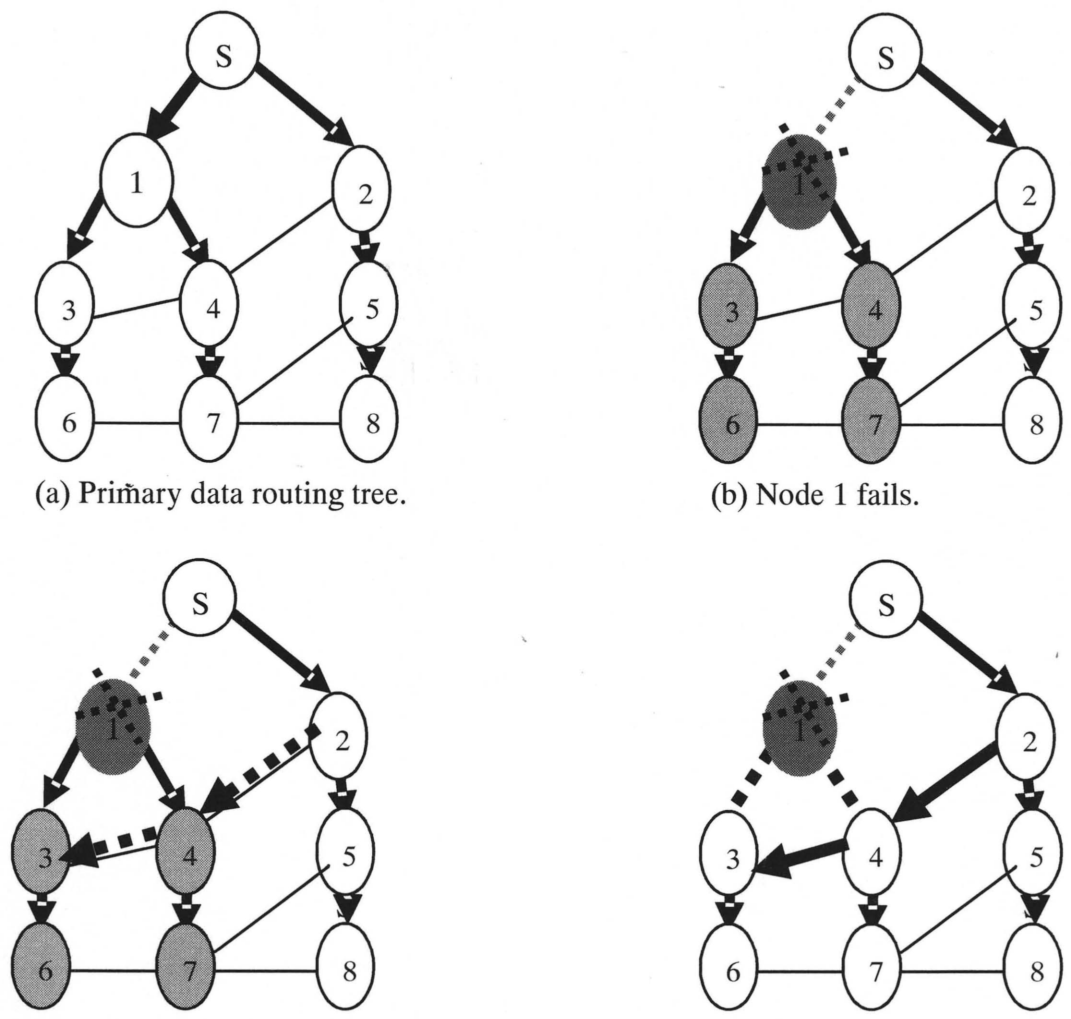
For (i=1 to i=x) do step 2.2 to 2.5

Step 2.2: Compute how much is traffic in the links (link utilization) between (u, c^u_i) { where c^u_i is the root of sub-tree T c^u_i and d^u_i the computed utilization value}.

Step 2.3: Compute a path by using BFSpath(G, S, c^u_i ,d^u_i) feasible to carry the rerouted traffic d^u_i (where $d^u_i = \lambda_{T c^u_i}$) to connect the root ($_c{}^u{}_i$) of the T c^u_i to the main T(n).

Step 2.4: Set the backup port according the discovered path from v to c^u_i.

Step 2.5: Color the recovered nodes to white.



(a) Primary data routing tree.

(b) Node 1 fails.

(c) Computing reroute path.

(d) Link (S, 1) fails.

Figure 2.7 Example of the Node Recovery Algorithm.

Example of Algorithm 2 is illustrated in Figure 2.7. The routing tree for Node S is shown in Figure 2.7 (a). Figure 2.7 (b) shows Node 1 as a failed node, then Node S finds a path to reroute the affected flows. After finding the new next hop for the rerouted flows, some of these flows change their path after the failure occurs. In this example, the new path from Node S to Node 6 is S→2→4→3→6, which is longer than the primary path S→1→3→6 by one hop.

2.3 Link Congestion Mitigation Algorithm

The last proposed algorithm is for link congestion mitigation. This algorithm is introduced in this section. The purpose of this algorithm is to determine a backup feasible path to accommodate the rerouted traffic results from link congestion. The backup path is also pre-planned before any link congestion occurs. Why do not use any of previous two proposed (link and node failure) algorithms? Because link congestion is a different phenomenon from the two pervious proposed algorithms in the following characteristics:

1.- In link recovery, all traffic has to be rerouted, but in link congestion, only part of the congesting traffic can be rerouted.

2.- Congestion in other links to relieve the currently congested ones has to be avoided. Figure 2.9 shows how some flows of a congested link can be rerouted to other links without causing congestion in other parts of the network while mitigating effectively the congestion.

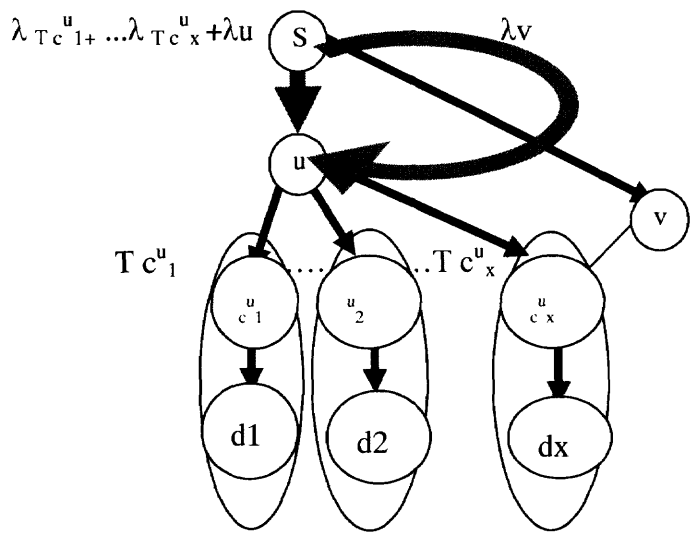
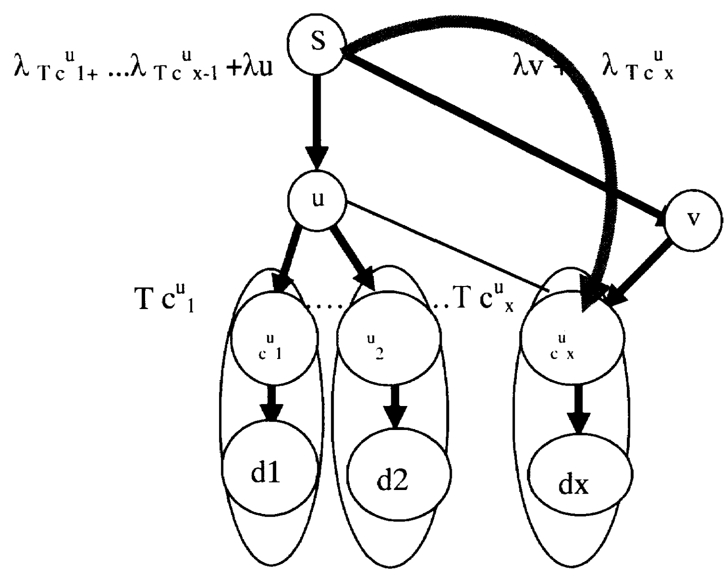


Figure 2.8 Consider link(S,u) as congested link.



$\lambda_{Tc^u_1} + \dots \lambda_{Tc^u_{x-1}} + \lambda u$
S
$\lambda v + \lambda_{Tc^u_x}$
u
v
$T c^u_1$
$\dots T c^u_x$
c^u_1
c^u_2
c^u_x
d1
d2
dx

Figure 2.9 Recovery from the congestion in link(S,u).

Table 2.9 Link Congestion Recovery Algorithm Notation

G(V,E)	a network with V nodes and E edges
P_n	primary port of node n before any failure
$b^{i,j}_n$	backup port of node n when link (i,j) fails
λ_{n0}	traffic generated in node n.
Ci,j	link(i,j) capacity
$\lambda_{Tc^u_i}$	The traffic destined to sub-tree T c^u_i.
$\lambda_{i,j}$	the traffic in link (i ,j) or link utilization
U	the total traffic in the network or utilization
$T_{rerouted}$	the total rerouted traffic
BP_{total}	the total number of backup port when a link fails

$$T_{\text{rerouted}} = \min\{\lambda_u, \lambda_{Tc^{u_i}} \text{ Where i=0 to i=x}\} \quad 2.9$$

$$\lambda_{i,j} < c_{i,j} \quad 2.10$$

Minimize:

$$U = \sum_{i,j} \lambda_{i,j} \quad 2.11$$

$$BP_{total} = \sum_{n \in V} b_n^{i,j} \quad 2{,}13$$

Algorithm 3 Mitigation of Link Congestion

Step 1: Initialization: Set the backup ports for Node S to null.

Step 2: Mark children nodes (next hop nodes for each destination) u={1,2,, m} as congested and doing the following:

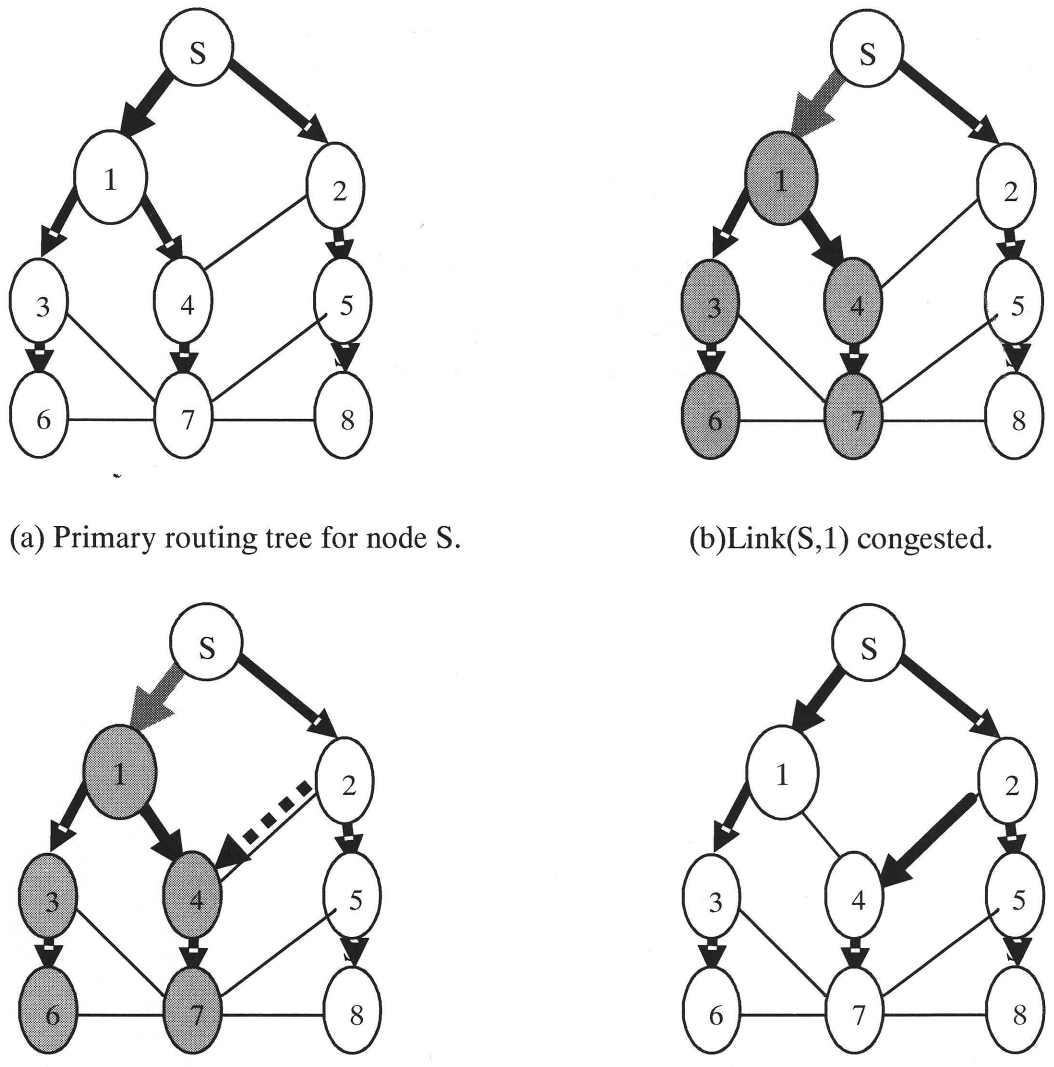
Step 2.1: Color all nodes in sub tree T(u) black, the next hop node u as red, and the other nodes in the topology white where the forwarding path is not affected by congestion.

Step 2.2: Compute how much is traffic should be rerouted (T_{rerouted}).

Step 2.3: Compute a back up path by using a feasible BFS path(G, S, $c^u{}_x$, T_{rerouted}) to accommodate the rerouted traffic T_{rerouted} to reconnect the black and red nodes to the main tree T(S).

Step 2.4: Set the backup port according the discovered path from S to u.

Step 2.5: Colors the recovered nodes as white.



(a) Primary routing tree for node S.

(b)Link(S,1) congested.

(c) Compute a backup path.

(d) New routing tree.

Figure 2.10 Example for link congestion mitigation algorithm.

Figure 2.10 shows an example of the normal Case I. Figure 2.10(a) is a primary routing tree. Link (S,u) is considered congested link in Figure 2.10 (b). In Figure 2.10 (c), Node S computes a feasible path to reroute the smallest flow in this link to other path to mitigate the congestion. In Step 2.2, the node S decides which flows should be rerouted and in Step 2.3, it computes the path for the rerouted traffic $T_{rerouted}$. The smaller flows are selected for rerouting to avoid congestion in other links. The routing tree for Node S is changed as shown in Figure 2.10 (d). Consequently, flows from Node S to Nodes 4 and 7 change their paths to S→1→4 and S→1→4→7, respectively. For other cases, the same procedure applies to Cases II and III, benefiting from the routing tables of other nodes and from the forward policy to avoid loops.

2.4 Time Complexity Analysis

In this section, the complexity analysis for the three proposed algorithms will be presented. Sections 2.4.1, 2.4.2, and 2.4.3 discuss the analysis of the link recovery algorithm, the node recovery algorithm complexity, and the link congestion mitigation algorithm, respectively.

2.4.1 Link Failure Recovery Algorithm Time Complexity

Step 2.3 computes a path in time complexity O (V+E) where V network nodes and E edges. Step 2.3 is executed m times, where m number of the assumed failed links that Node S has. Therefore the complexity is O (m(V+E)). For Step 2.3, most of cases, a path is found before traversing all network nodes V and E edges. For variable m, m is the node degree or number of edges always, which is very small, compared to V and E. for example, m could be between 3 to 10, which number is negligible. As result, the total time complexity of the algorithm is O(V+E).

2.4.2 Node Failure Recovery Algorithm Time Complexity

Again, the main step in the algorithm is Step 2.3, which computes a path in time complexity O(V+E), where V network nodes and E edges. Step 2.3 is executed x times for each assumed next hop of Node S, where x is the number of children of the failed node and m is the Node S's degree or its number of edges. Therefore, the time complexity is O(x.m(V+E)). As for Step 2.3, in most of the cases, a node finds the path before traversing all network nodes V and E edges. considering that variables x and m are very small compared to V and E, the total time complexity of the algorithm is O(V+E).

2.4.3 Link Congestion Mitigation Algorithm Time Complexity

Similar to the link failure recovery algorithm, Step 2.3 in the congestion mitigation algorithm computes a path in time complexity O(V+E), where V network nodes and E edges. As Step 2.3 is executed m times, where m number of the assumed congested links that node S has. Note that m has different meaning from the other two previous algorithms. Therefore, the time complexity is O(m(V+E)) and the total time complexity of the algorithm is O(V+E) when m is very small compared to V and E.

CHAPTER 3

ALGORITHMS IMPLEMENTAION AND SIMULATION RESULTS

First, node structure and routing module in ns2 will be described. Then, describing the procedures in the class Simulator in Section 3.1 and instance procedures in the class node to access and operate on individual nodes. Section 3.2 presents the routing module in ns2 node. Then, the simulations study the load distribution in each link and the throughput for each node. Section 3.3 contains the simulation setup and the result for link failure scheme. Section 3.4 presents the simulation setup and the results of link node scheme. In Section 3.5, link failure recovery algorithm is applied for double link failure.

3.1 Node Basics

The instance node in ns2 constructs of simple classifier objects. The Node itself is a standalone class in OTcl. However, most of the components of the node are themselves TclObjects. The typical structure of a unicast node is as shown in Figure 3.1. This simple structure consists of two TclObjects: an address classifier (classifer_) and port classifier (dmux_) .These classifiers deliver incoming packets either the correct agent (e.g., Upper layer) or outgoing link (e.g., next hop).

Basic node contains the following components:

• An address or id : _,increasing by 1 (from initial value 0) across the simulation namespace as new node is created.

- A list of neighbors (neighbor_): Structure of a Unicast Node. Notice that entry_ is simply a label variable instead of a real object, e.g., the classifier_.
- A list of agents (agent_)
- A node type identifier (nodetype_)
- A routing module

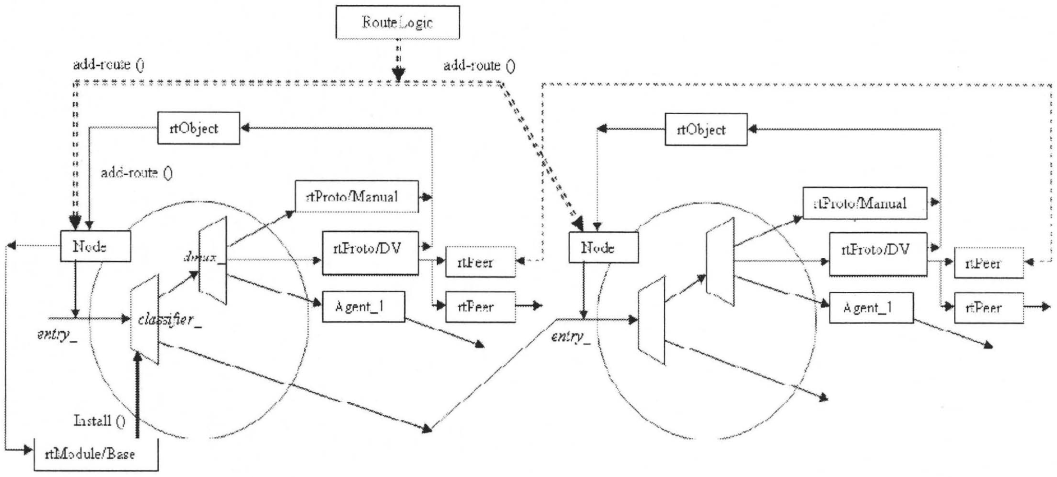


Figure 3.1 Ns2 unicast node routing structure.

3.1.1 Address and Port Number Management

The procedure $node id returns the node number of the node. This number is automatically incremented and assigned to each node at creation by the class Simulator method, $ns node. The class Simulator also stores an instance variable array1, Node_, indexed by the node id, and contains a reference to the node with that id. The procedure $node agent hporti returns the handle of the agent at the specified port. If no agent at the specified port number is available, the procedure returns the null string. The procedure alloc-port returns the next available port number. It uses an instance variable, np_, to track the next unallocated port number. *add-route and add-routes* are procedures that used by unicast routing to add routes to populate the classifier_ The usage syntax is $node add-route (destination id). dmux_ is a port demultiplexer at the node, if the destination id is the same as this node's id, it is often the head of a link to send packets for that destination. *add-routes* (destination id) used to add multiple routes to the same destination that must be used simultaneously in round robin manner to spread the bandwidth used to reach that destination across all paths (e.g., Multiple Equal Cost Path (MECP)). Finally, the procedure intf-changed{} is invoked by the network dynamics code if a link incident on the node changes state.

3.1.2 Agent Management

Given an agent, the procedure attach{} will add the agent to its list of agents_, assign a port number the agent and set its source address, set the target of the agent to be its (*i.e.*, the node's) entry{}, and add a pointer to the port demultiplexer at the node (dmux_) to the agent at the corresponding slot in the dmux_ classifier. Conversely, detach{}will remove the agent from agents_, and point the agent's target, and the entry in the node dmux_ to nullagent.

3.1.3 Tracking Neighbors

Each node keeps a list of its adjacent neighbors in its instance variable, neighbor_. The procedure add-neighbor {} adds a neighbor to the list. The procedure neighbors {} returns this list. The function of a node when it receives a packet is to examine the packet's fields, usually its destination address, and on occasion, its source address. It should then map the values to an outgoing interface object that is the next downstream recipient of this packet. In *ns*, this task is performed by a simple *classifier* object. A node in *ns* uses many different types of classifiers for different purposes. A classifier provides a way to match a packet against some logical criteria and retrieve a reference to another simulation object based on the match results. Each classifier contains a table of simulation objects indexed by *slot number*.

3.2 Routing Module and Classifier Organization

An *ns* node is essentially a collection of classifiers. The simplest node unicast contains only one address classifier and one port classifier, as shown in Figure 3.1.

3.2.1 Routing Module

In general, every routing implementation in *ns* consists of three function blocks:

- *Routing agent* exchanges routing packet with neighbors,
- *Route logic* uses the information gathered by routing agents (or the global topology database in the case of static routing) to perform the actual route computation.
- *Classifiers* sit inside a Node. They use the computed routing table to perform packet forwarding.

Notice that when implementing a new routing protocol, one does not necessarily implement all of these three blocks. For instance, when one implements a link state routing protocol, one simply implement a routing agent that exchanges information in the link state manner, and a route logic that does

Dijkstra on the resulting topology database. It can then use the same classifiers as other unicast routing protocols. When a new routing protocol implementation includes more than one function blocks, especially when it contains its own classifier, it is desirable to have another object, which is called a *routing module*, that manages all these function blocks and to interface with node to organize its classifiers. Figure 3.2 shows functional relation among these objects. Notice that routing modules may have direct relationship with route computation blocks, i.e., route logic and/or routing agents. Hover, route computation MAY not install their routes directly through a routing module.

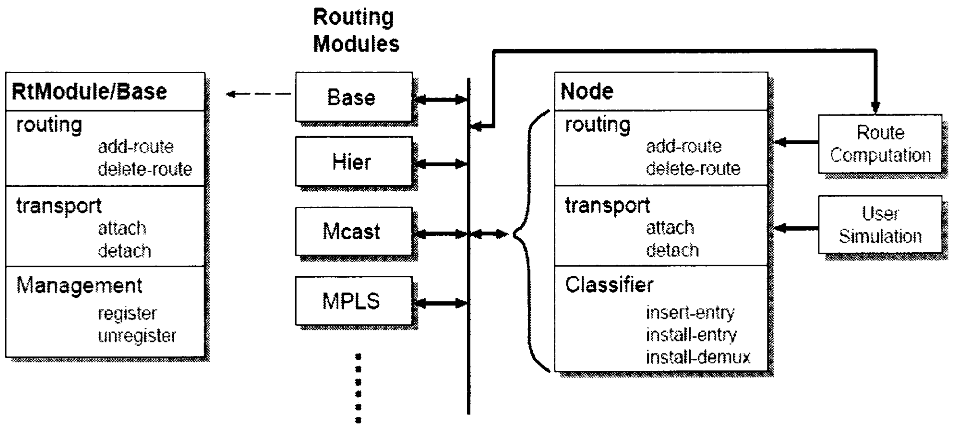


Figure 3.2 Interaction among node, routing module, and routing.

A routing module contains three major functionalities:

1- A routing module initializes its connection to a node through register {}, and tears the connection down via unregister {}. Usually, in register{} a routing module (a) tells the node whether it interests in knowing route updates and transport agent attachments, and (b) Creates its classifiers and install them in the node. In unregister {} a routing module does the exact opposite: it deletes its classifiers and removes its hooks on routing update in the node.

2- If a routing module is interested in knowing routing updates, the node will inform the module via RtModule::add-route{dst, target} and RtModule::delete-route{dst, nullagent}.

3- If a routing module is interested in learning about transport agent attachment and detachment in a node, the node will inform the module via RtModule::attach{agent, port} and RtModule::detach{agent, nullagent}.

3.2.2 Node Interface

To connect to the above interfaces of routing module, a node provides a similar set of interfaces:

In order to know which module to register during creation, the Node class keeps a list of modules as a class variable. The default value of this list contains only the base routing module. The Node class provides the following two *procedures* to manipulate this module list:

- Node::enable-module{[name]} If module RtModule/[name] exists, this proc puts [name] into the module list.
- Node::disable-module{[name]} If [name] is in the module list, remove it from the list. When a node is created, it goes through the module list of the Node class, creates all modules included in the list, and registers these modules at the node. After a node is created, one may use the following instance producers to list modules registered at the node.
- Node::list-modules{} Return a list of the handles of all registered modules. Node::get-module{[name]} Return a handle of the registered module whose name matches the given one. Notice that any routing module can only have a single instance registered at any node.

To allow routing modules register their interests of routing updates, a node object provides the following instance procedures:

- Node::route-notify{module} Add module into route update notification list.
- Node::unreg-route-notify{module} Remove module from route update notification list.

- Node::port-notify{module} Add module into agent attachment notification list.

- Node::unreg-port-notify{module} Remove module from agent attachment notification list.

Node provides the following procedures to manipulate its address and port classifiers:

- Node::insert-entry inserts classifier into the entry point of the node. It also associates the new classifier with module so that if this classifier is removed later, module will be unregistered.

- Node::install-entrydiffers from Node::insert-entry in that it deletes the existing classifier at the node entry point, unregisters any associated routing module, and installs the new classifier at that point. If hook is given, and the old classifier is connected into a classifier chain, it will connect the chain into slot hook of the new classifier.

- Node::install-demux{demux, port} places the given classifier demux as the default demultiplexer. If port is given, it plugs the existing demultiplexer into slot port of the new one. Notice that in either case it does not delete the existing demultiplexer.

3.2.3 New Features Added

New utility procedures have been added that help to implement previous proposed algorithms. These procedures as following:

- update-route {} this procedure update the routing table for node.

- get-next-hop {} return the next hop for a specified destination.

- compute-next-hop {}: compute a route from source to destination and return next hop.

- print-route {}: display the route from source to destination.

These procedures are implemented in C++ programming language. Then, the procedures linked to TCL code which is used in ns2 simulation software. Furthermore, an implementation for the forward policy has been explained in Chapter 2. Each node has a classifier which controls the forwarding process. Another constrain have to be satisfied that if the received packet is the same packet just the node has forwarded it then will update-route producer notified to enable the backup next hop. In this case, the next hop considered invalid and the node enables its backup next hop.

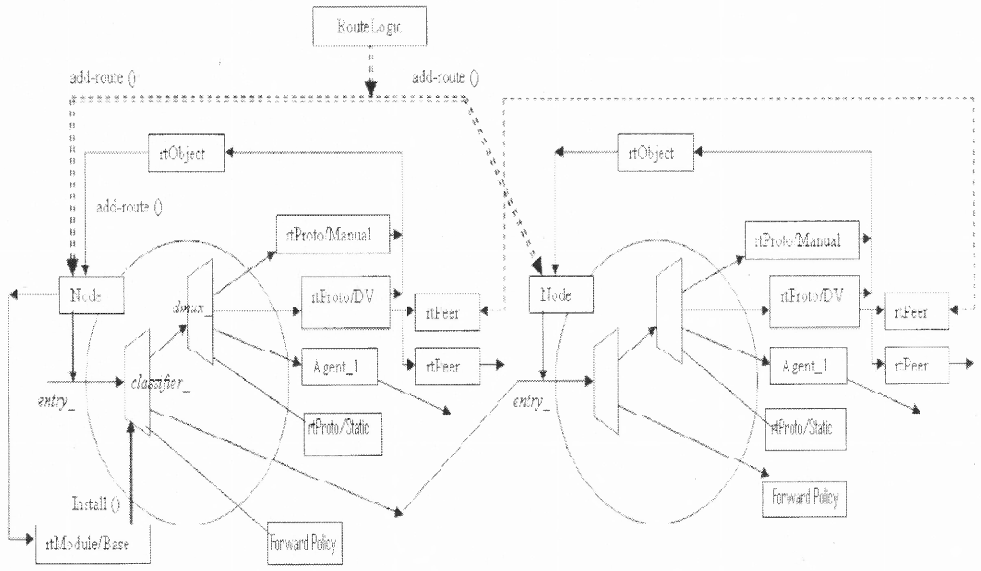


Figure 3.3 A modified ns2 unicast node routing structure.

3.3 Simulation for Link Recovery

Using ns2 network simulator was adopted to study the performance of the proposed scheme for node/link failure recovery. Comparing the proposed schemes with Link state protocols such as OSPF is extensively deployed. The topology used in simulation all randomly generated using BRITE [41] topology generator. Four random topologies with different network size range from 20 to

100 nodes are used in simulation. The traffic generated in network using CBR traffic with rate 3Mbps for each flow. generate mesh traffic among nodes in the network that means each node sends traffic to other nodes in the network. Node minimum degree is four links for each node.

3.3.1 Load Distribution for Link

The simulation focus on Traffic load in each link after a random selected link has been failed. Then, the traffic load in each link for our proposed recovery scheme was compared with link state protocol. The link index arranged according to BRITE software order each link is bidirectional. For example, link (1,2) has two indices one for direction 1-to-2 and the other for 2-to-1. Figure 3.3 depicts the link load for link state protocol in blue points and our proposed fast recovery in purple for a network size 20 nodes. This graph shows how algorithms can recovery the failure in short time and with accuracy near the link state protocol. Where link state protocol waits after a failure occurs and recalculate new path to resume service.

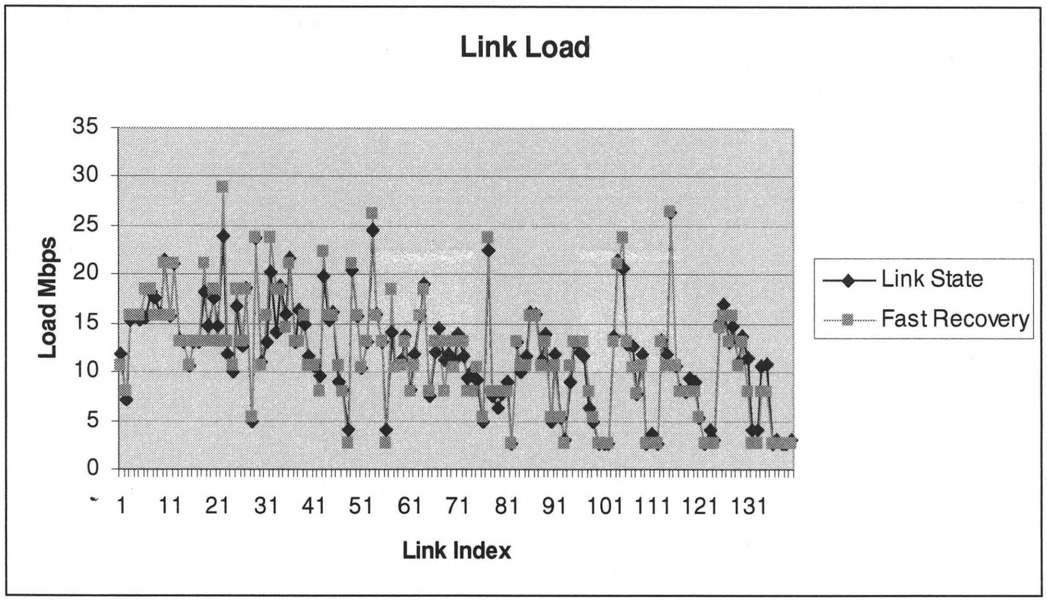


Figure 3.4 Link load for each link in the network size 20 nodes.

Then, Figures 3.5, 3.6, 3.7 show the load in each link. In all figures, the link load for Fast Recovery Algorithm close to the link state protocol or recalculated path. In addition, noticing that our recovery algorithm can distribute the traffic among links near optimal. For example, Figure 3.5 shows that link utilization for Fast Recovery less than the link utilization when use link state protocol. In the same time, Figure 3.8 shows that the throughput for all nodes is better than link state. Why could this be happen? Referring to Equation 1.1 the service interruption time the term $T_{\text{notfication}}$ and T_{recomute} these values are reduced in Fast Recovery. Consequently, the service time increases that mean the node receive more packets. In other words, the failed link, which is selected randomly, is far from the source that make the service interruption time for link state is longer than the Fast recovery. Furthermore, the location of link failure is significant so there have been many studies conducted to identify the critical link in network. For instance, the location of link failures has been addressed in [42][43].

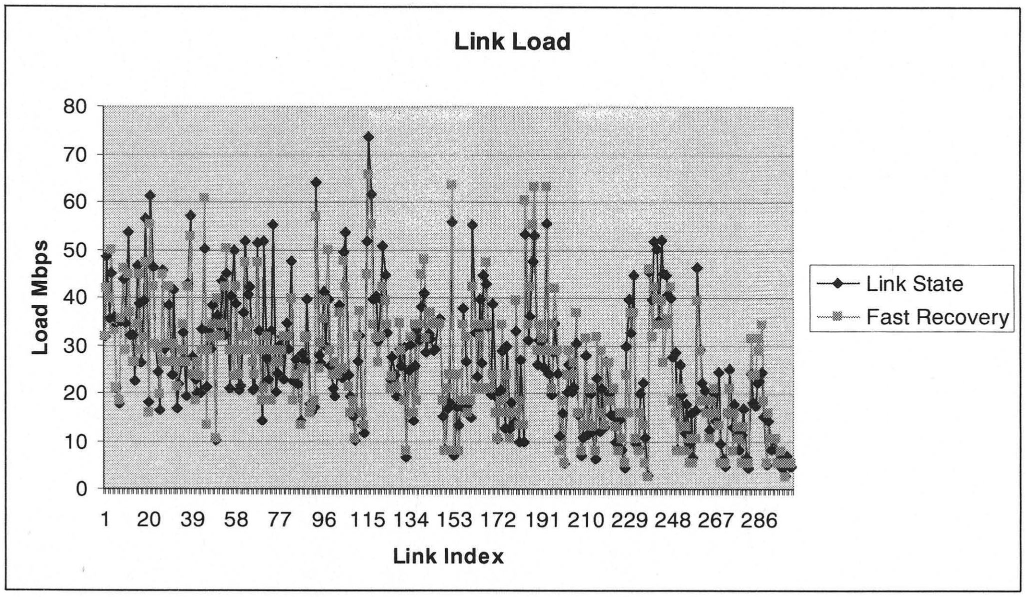


Figure 3.5 Link load for each link in the network size 40 nodes.

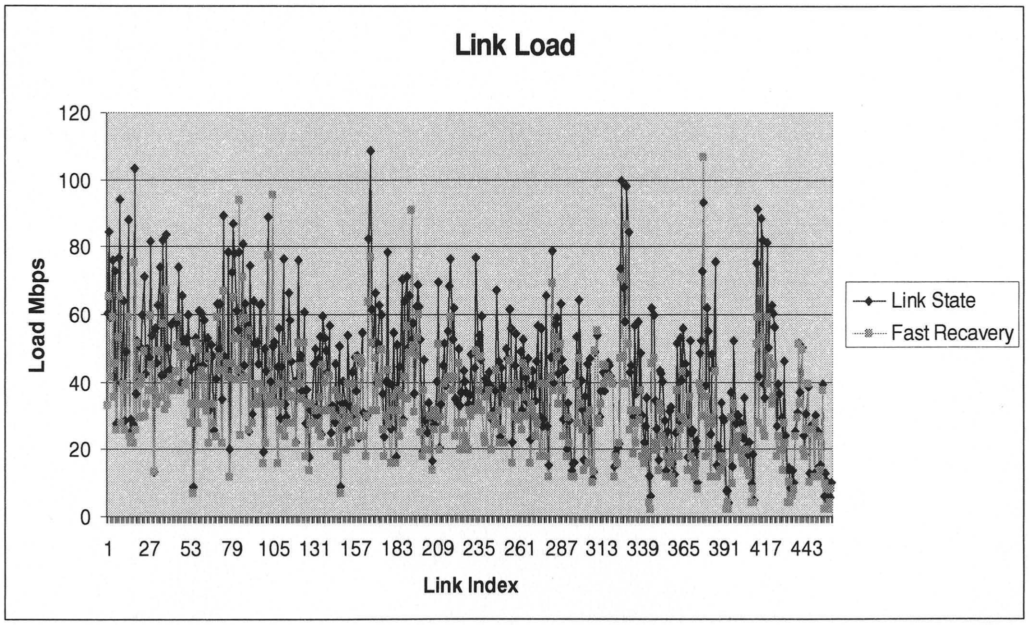


Figure 3.6 Link load for each link in the network size 60 nodes.

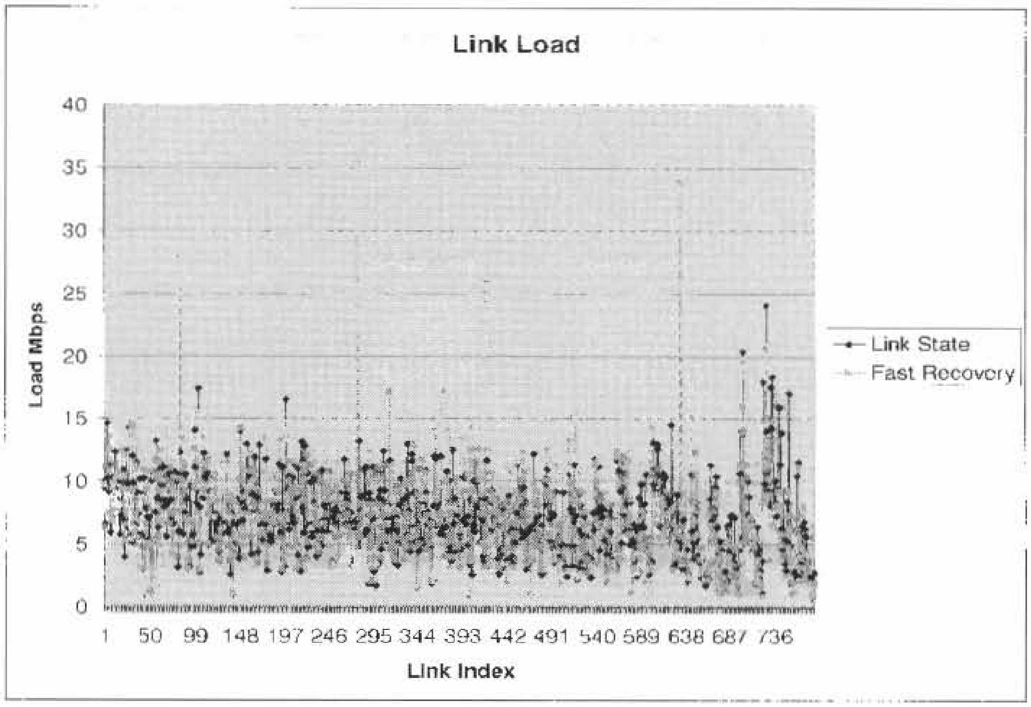


Figure 3.7 Link load for each link in the network size 100 nodes.

3.3.2 Throughput for Node

Throughput is number of received and acknowledged packets for a destination. Here, throughput is the mount of received traffic in Mbps. In Figure 3.6, the throughput for link state and fast recovery are the same. Again referring to service interruption time equation T_{intrr}, T_{intrr} is the same for link sate and Fast Recovery.

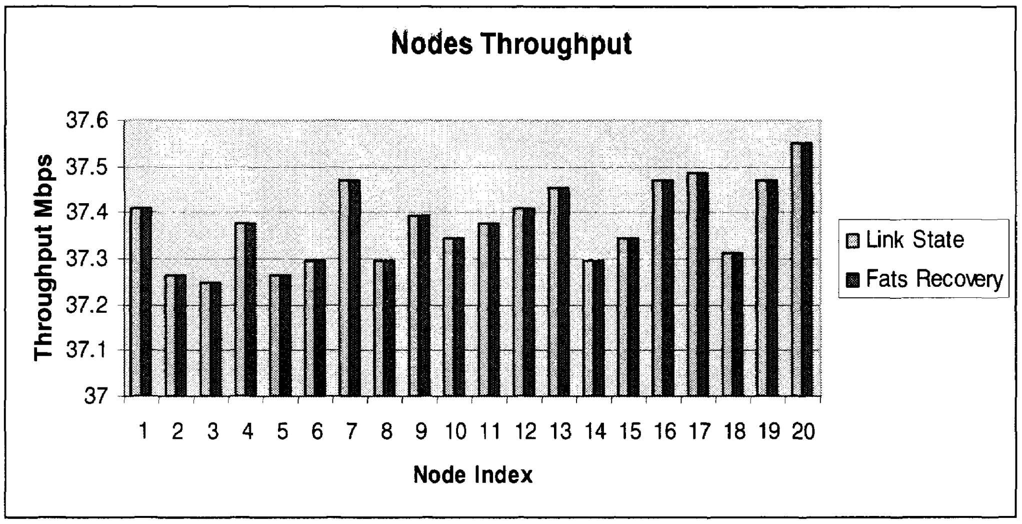


Figure 3.8 Throughput for each node in the network size 20 nodes.

On other hand, Figure 3.6 shows that the throughput for Fast Recovery is better than link state for all nodes.

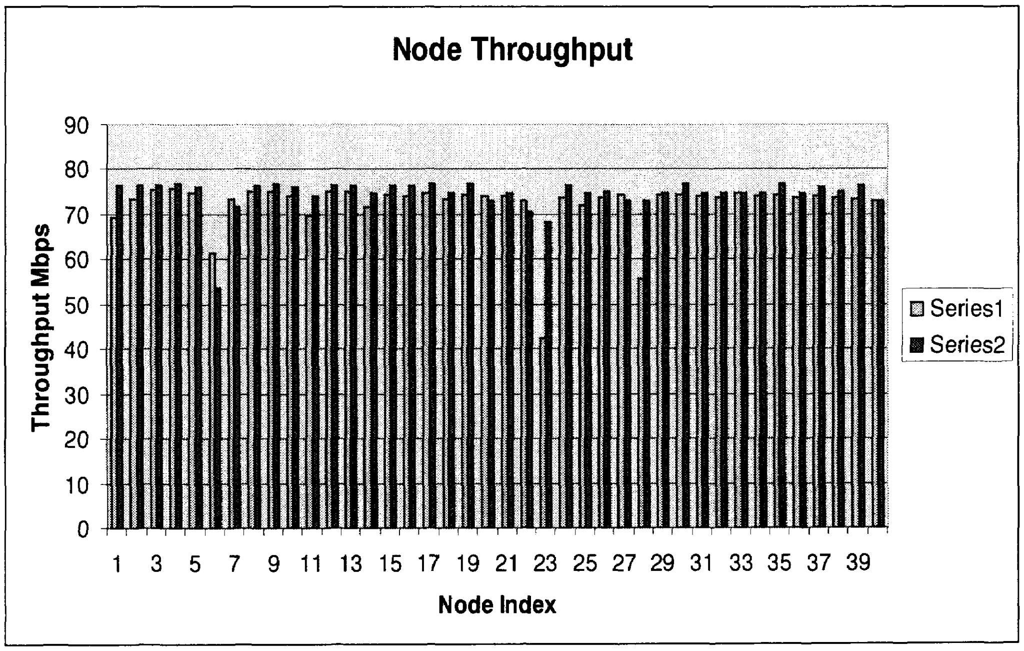


Figure 3.9 Throughput for each node in the network size 40 nodes.

Figures 3.8 and 3.9 show the same thing. All nodes have better throughput when using Fast Recovery instead of link state. Figure 3.11 shows the throughput for first fifty nodes (0-49) in the network and Figure 3.12 presents the throughput for the other fifty nodes (50-99).

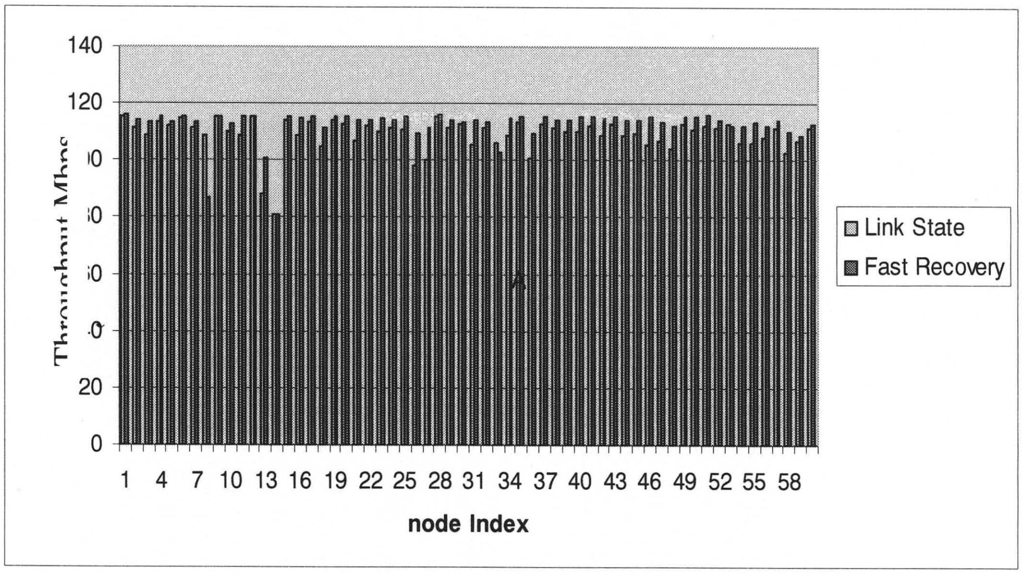


Figure 3.10 Throughput for each node in the network size 60 nodes.

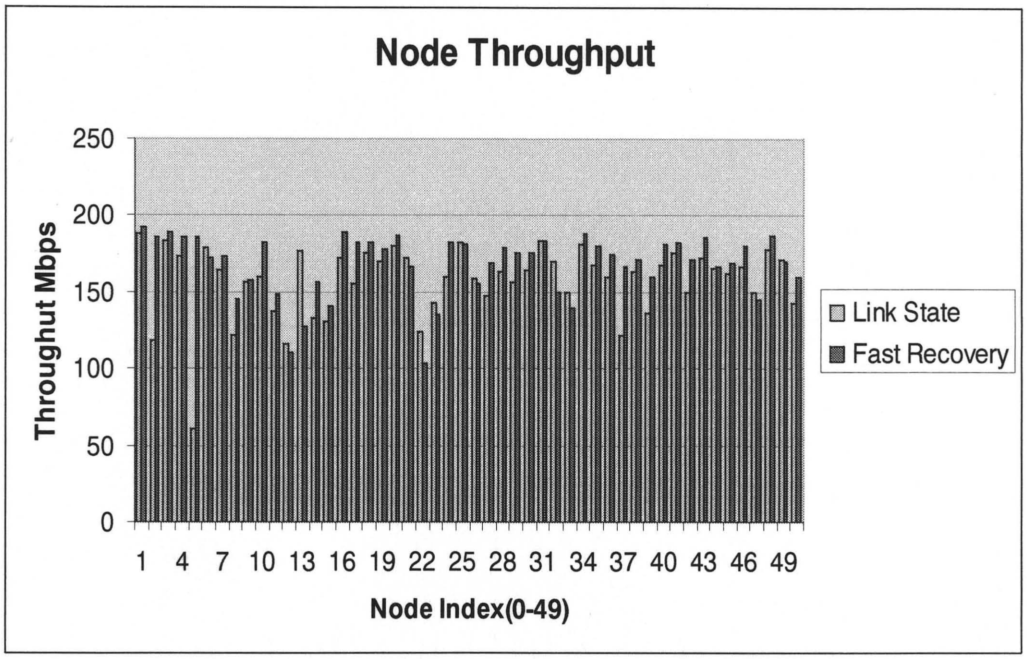


Figure 3.11 Throughput for each node in the network size 100 nodes (0-49).

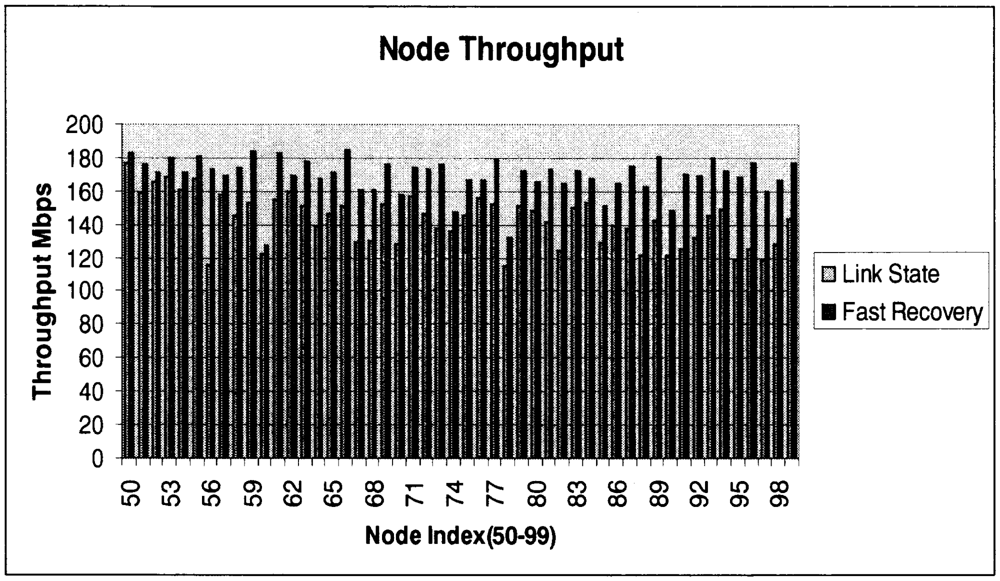


Figure 3.12 Throughput for each node in the network size 100 nodes (50-99).

3.4 Simulation for Node Recovery

In node recovery simulation, three random topologies were used with different network size range from 20 to 60 nodes. The traffic generated is as in link recovery simulation 3Mbps for each flow. Node minimum degree is two links for each node.

Figure 3.13 Link load for each link in the network size 20 nodes for node failure.

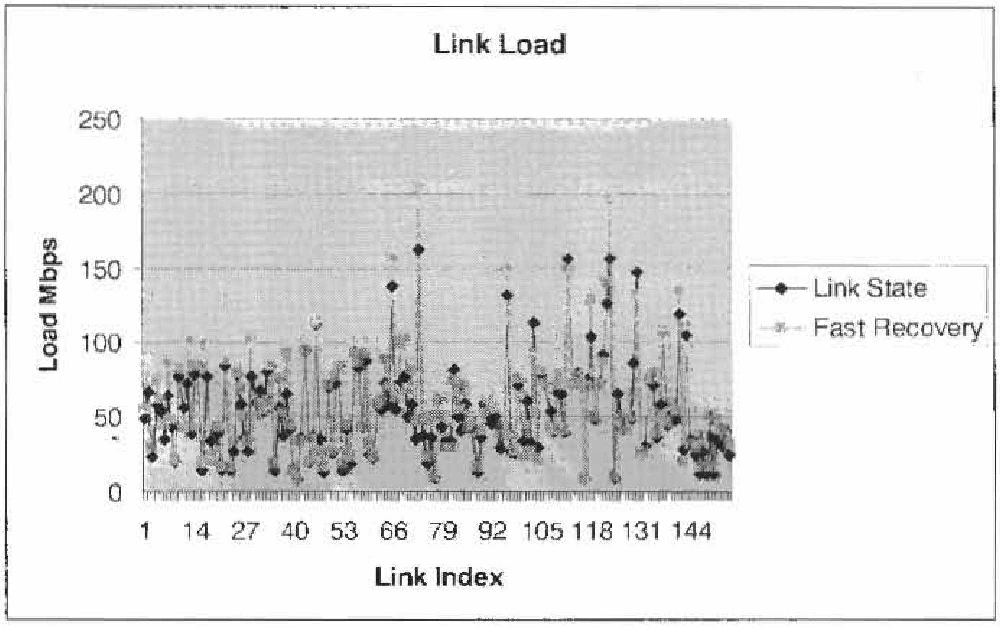


Figure 3.14 Link load for each link in the network size 40 nodes for node failure.

3.4.2 Throughput for Nodes

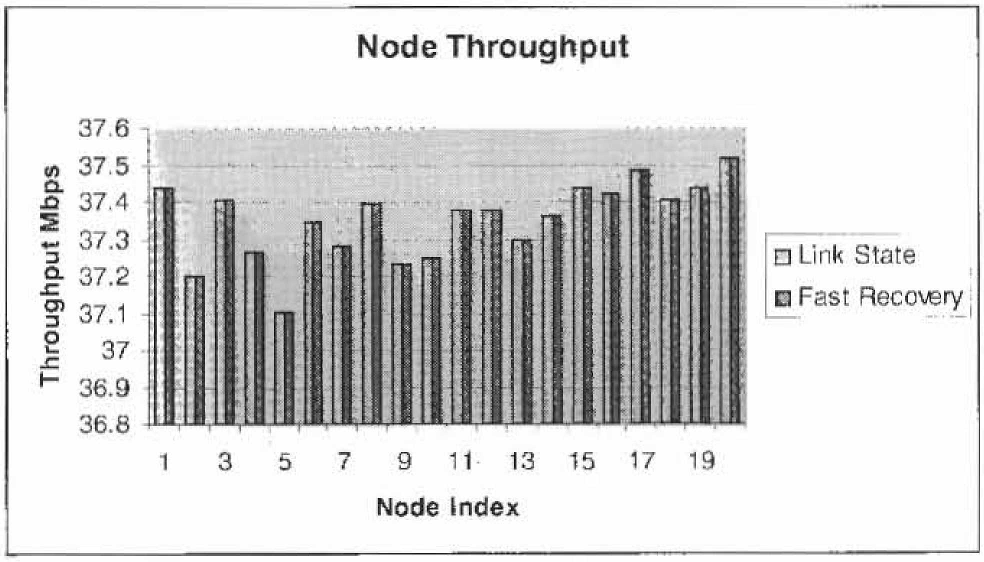


Figure 3.15 Throughput for each node in the network size 20 nodes for node failure.

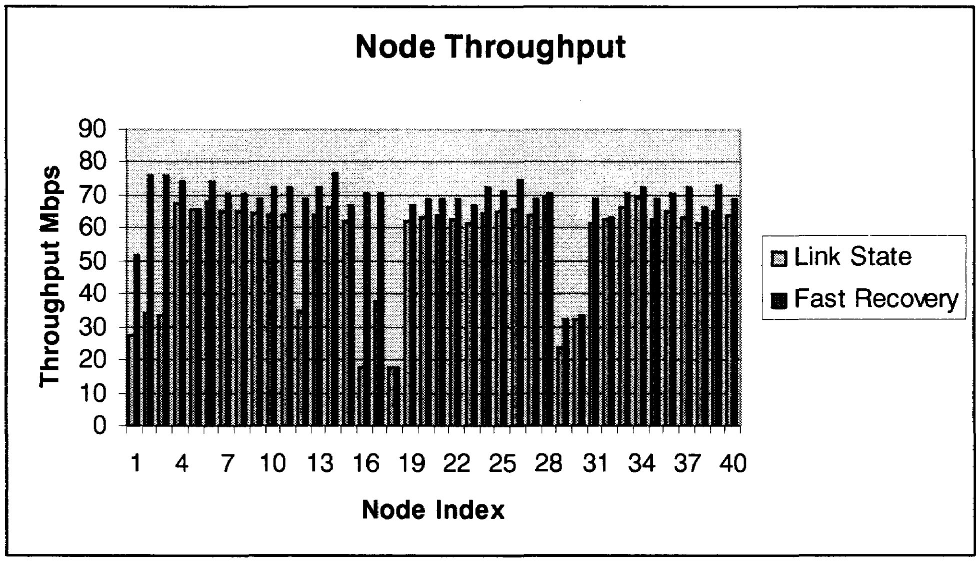


Figure 3.16 Throughput for each node in the network size 40 nodes double link failure

3.5 Simulation for Double Link Recovery

In this section apply recovery scheme in double link failure. As show in figure 3.17 our scheme can recovery from two link failure occurs in the same time with a good load distribution. Figure 3.18 shows that can also gain the same throughput for each node as link state protocol.

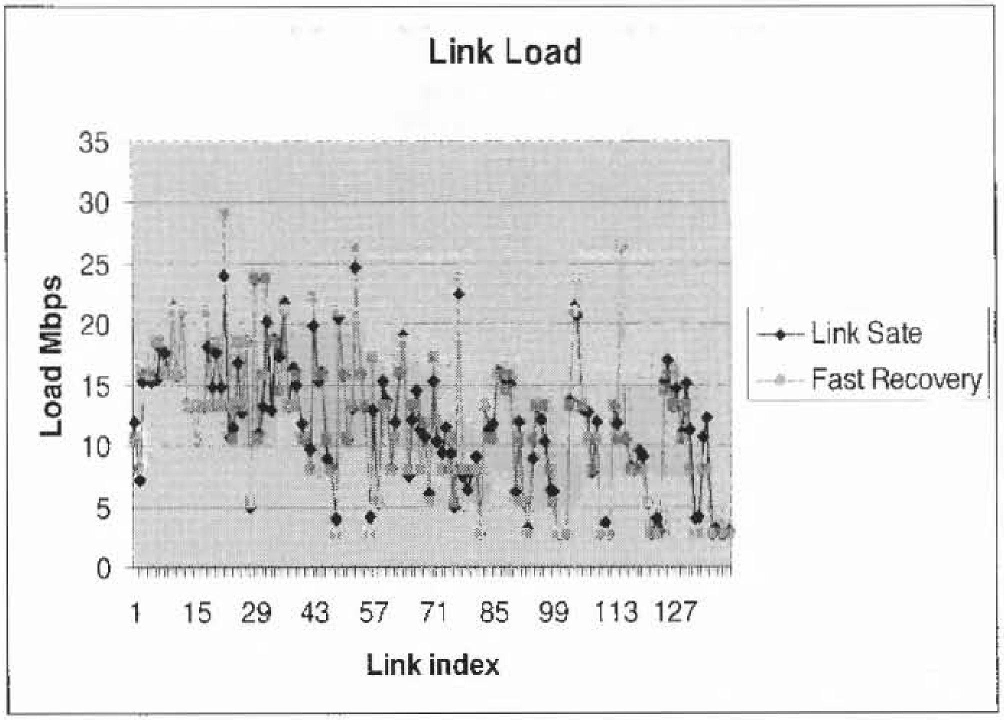


Figure 3.17 Load for each link in the network size 20 nodes for node failure double link failure.

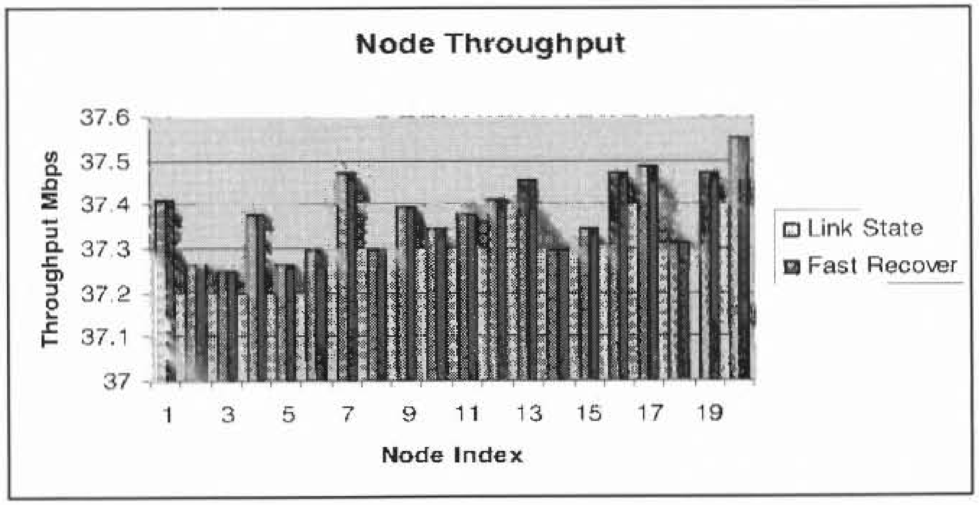


Figure 3.18 Throughput for each node in the network size 20 nodes double link failure.

CHAPTER 4

CONCLUSION AND FUTURE WORK

Link/node failure occurs frequently as has been shown in Figure 1.1, and these failures are the cause of service disruption network systems. To avoid long time service interruption, many techniques and approaches have been developed. Hardware protection works in physical layer. It needs frequently maintenance which is very costly. There are many Network layer recovery solutions have been proposed. For instance, IP Fast ReRouting (IPFRR) is a scheme that ties to reroute the traffic from the failed part of the network to other working parts without using extra hardware. Hover, the hardware solution protection recovers faster than the other solution when failure occurs. Indeed, IPFRR is a preplanned backup path to reroute traffic on a link/node failure and can be implemented by modifying the exiting router software. A tradeoff between convergence time and cost always exits.

4.1 Contributions

The goals of the thesis are as following. First, reduces the service interruption time as possible as can. Second, recover all the flows without causing congestion in other part of the network. Three algorithms re proposed that recover the node failure, link failure and congestion. These schemes depend on the local recovery and local reroute that reduce the notification and dissemination time. Due to that, the service interruption time is reduced and recovery convergence becomes fast.

A Comparison between the proposed schemes results and OSPF protocol are conducted.

Chapter 2 presents the proposed schemes. First, link failure recovery where every node considers primary next hop fail, then it computes a path. This path does not cause congestion in other links while rerouting traffic in it. Second, node failure recovery scheme is more critical than link failure because each node has many links. Similarly to link failure recovery, node failure recovery pre-computes a feasible path that does not cause congestion in other links. Third, congestion mitigation scheme tries to reduce the like utilization but may increase the over all network utilization. To clarify the difference between link utilization and network utilization, for example, maybe the computed path longer than the primary path that increase the over all network utilization. On the other hand, there is a short path and many flows share one or more links in this path, so more flows take shared link more these link utilized till causing congestion in one or more links. Finally, the time complexity analysis of schemes is presented. Complexity time for all algorithms was liner.

In Chapter 3, the simulation and results is presented. Basic components in ns2 will be explained that re used to implement the schemes. Then, a simulation was used to show the performance of the proposed schemes. The simulation was focused on two network factor. Firstly, the link utilization, the results show that the proposed techniques less link utilization than the OSPF protocol. Consequently, reduce the probability of congestion occurs. Secondly, throughput is a significant factor in any network. The results show that proposed schemes throughput per node always better than OSPF. That result from reducing the service interruption time. However, when network size increases the convergence time between the proposed schemes and OSPF grows. Therefore, the throughput grows due to increasing in service time.

4.2 Future Work

This thesis focuses only in a single failure either node or link. Hover, applying the scheme on double link failure occurs in the same time, it gives similar link utilization for re-calculated path by OSPF. But what could happen if two flows have the same backup path. Nodes share this path when they pre-computed their next hops individually without any co-oration among them. Now, when the failure occurs each node will reroute the traffic to backup path that will cause congestion in some links in pre-computed path. Extending the work to solve like this problem for multiple link/node failure required a lot of work. Furthermore, to keep cooperation among nodes may cause exchange information overhead in the network.

Finally, implementing our schemes in Linux kernel to study the exact performance of the proposed schemes and monitor the changes in the traffic when failures occur. Implementing these algorithms in kernel will help to design such a batch file could install it in routers to update their operating system instead of replacing the route itself.

REFERENCES

[1] I. Chlamtac, A. Ganz, and G. Karmi, “Lightpath communications: an approach to high bandwidth optical WAN’s,” IEEE Transactions on Communications, Vol. 40, pp. 1171-1182, 1992.

[2] T. Frisanco, ”Optimal spare capacity design for various protection switching methods in ATM networks,” IEEE ICC, pp. 293-298, 1997.

[3] D. K. Hsing, B. C. Cheng, G. Goncu and L. Kant,” A restoration methodology based on pre-planned source routing in ATM networks,” ICC, pp. 277-182, 1997.

[4] A. Itai and M. Rodeh, “The multi-tree approach to reliability in distributed networks, Information and Computation,” Vol. 79, pp. 43-59, 1998.

[5] M. M´edard, S. G. Finn and R.A. Barry, ”A novel approach to automatic protection switching using trees,” ICC, pp. 272-276, 1997.

[6] M. M´edard, S. G. Finn and R.A. Barry, “WDM loop-back recovery in mesh networks,” OFC, pp. 298-299, 1998.

[7] M. M´edard, S. G. Finn, R.A. Barry and R.G. Gallager, “Redundant trees for preplanned recovery in arbitrary vertex-redundant or edge-redundant graphs,” IEEE/ACM Transactions on Networking, Vol. 7, pp. 641-652, 1999.

[8] S. Ramamurthy and B. Mukherjee, “Survivable WDM mesh networks Part I-Protection,” IEEE INFOCOM, pp. 744-751, 1999.

[9] S. Ramamurthy and B. Mukherjee, “Survivable WDM mesh networks Part II-Restoration,” IEEE ICC, pp. 2023-2030, 1999.

[10] G. D. Signorelli, M. Gryseels and P. M. Demeester, “SDH over WDM: interworking and planning aspects,” Proceedings of the SPIE, Vol. 3408, pp. 235-246, 1998.

[11] N. Wauters, C. Ocakoglu, K. Struyve and F. P. Falcao, “Survivability in a new pan-European carriers’ carrier network based on WDM and SDH technology: current implementation and future requirements,” IEEE Communications Magazine, Vol. 37, No. 8, pp. 63-69, 1999.

[12] T. H. Wu and R. C. Lau, “A class of self-healing ring architectures for SONET network applications,” IEEE Trans. Communications, Vol. 40, pp. 1746-1756, Nov. 1992.

[13] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C.-N. Chuah, and C. Diot, “Characterization of failure in an IP backbone,” in IEEE INFOCOM, Mar. 2004.

[14] A. Leon-Garcia, and I. Widjaja,” Communication Network Fundamental Concepts and Key Architechtures”, 2nd edition

[15] S. Bellcore, “Automatic Protection Switching for SONET, Issue1,” Oct. 1990.

[16] www.cs.virginia.edu/~mngroup/projects/mpls/documents/thesis, Retrieved Aug. 2, 2007 from the World Wide Web.

[17] ANSI. Fiber Distributed Data Interface (FDDI) { Token Ring Media Access Control (MAC), ANSI X3.139-1987, 1987.

[18] G. Malkin. IETF RFC 2453: RIP version 2, Nov. 1998.

[19] C. L. Hedrick. IETF RFC 1058: Routing Information Protocol, Jun. 1988.

[20] J. Moy. IETF RFC 2328: OSPF version 2, Apr. 1998.

[21] D. Oran. IETF RFC 1142: OSI IS-IS intra-domain routing protocol, Feb. 1990.

[22] C. Alaettinoglu, V. Jacobson, and H. Yu.”Toward millisecond IGP convergence,” NANOG 20, Washington, D.C., USA, Oct. 2000.

[23] “IP Fast Reroute Overview and Things we are struggling to solve,” http://bgp.nu/~dward/IPFRR/IPFRR_overview_NANOG.pdf, Retrieved Aug. 2, 2007 from the World Wide Web.

[24] N. Ansari, G.Cheng, R. N. Krishnan, “Efficient and Reliable Link State Information Dissemination”,in IEEE communication letters, Vol. 8, No. 5, May 2004.

[25] T. Korkmaz, M. Krunz, “Hybrid Flooding and Tree-based Broadcasting for Reliable and Efficient Link-state Dissemination,” Global Telecommunications Conference. GLOBECOM '02. IEEE, Vol 3, pp. 2400-2404, Nov. 2002.

[26] G. Cheng, N. Ansari, “An Information Theory Based Framework for Optimal Link State Update”, IEEE COMMUNICATIONS LETTERS, Vol. 8, No. 11, Nov. 2004.

[27] Y. Jia, I. Nikolaidis, P. Gburzynski, “Alternative Paths vs. Inaccurate Link State Information in Realistic Network Topologies,“ Proc. Int. Symp. Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2002), Soc. for Modeling & Simulation International, pp. 162-169, Jul. 2002.

[28] Y. Jia, I. Nikolaidis, P. Gburzynski,"Multiple Path Routing in Networks with Inaccurate Link State Informat ion", Communications, 2001. ICC 2001. IEEE International Conference, Vol. 8, pp. 2583-2587, 2001.

[29] G. Cheng, N. Ansari, "Minimizing the Impact of Stale Link State Information on QoS Routing", in proceedings IEEE Communications Society subject matter experts for publication in the IEEE GLOBECOM 2005.

[30] A. Al-Fuqaha, G. Chaudhry, C. Beard, M. Guizani, I. Habib,"Link-State Update Policies for All-Optical DWDM Transport Networks", Communications, 2004 IEEE International Conference on , Vol 3, pp. 1831-1835, Jun. 2004.

[31] M. Shand and S. Bryant, "IP fast reroute framework," Internet-Draft, Oct. 2005. [Online]. Available: http://www.ietf.org/internet-drafts/draftietf-rtgwg-ipfrr-framework-04.txt

[32] A. Iselt, A. Kirstdter, A. Pardigon, and T. Schwabe, "Resilient routing using ecmp and mpls," in IEEE High Performance Switching and Routing (HPSR), Apr. 2004.

[33] Cornelis Hoogendoom, Karl Schrodi, Manfred Huber, Christian Winkler and Joachim Charinski," Towards Carrier-Grade Next Generation Networks," Proceedings of ICCT2003.

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

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

[36] A. Atlas, "Basic specification for IP fast-reroute: loopfree alternates," Internet-Draft, Feb. 2005.Available:http://www3.ietf.org/proceedings/05mar/IDs/draft-ietf-rtgwg-ipfrrspec-base-03.tx

[37] S. Bryant, M. Shand, and S. Previdi, "IP fast reroute using not-via addresses," Internet-Draft, Oct. 2005.Available: http://www.ietf.org/internet-drafts/draft-bryant-shand-ipfrrnotvia- addresses-01.txt

[38] A. Kvalbein et al., "Fast IP network recovery using multiple routing configurations, in IEEE INFOCOM, Apr. 2006.

[39] "NS2", Retrieved Aug. 2, 2007 from the World Wide Web: http://www.isi.edu/nsnam/ns/doc/index.html

[40] “Breadth First Search,” Retrieved Jun.10, 2007 from the World Wide Web: http://en.wikipedia.org/wiki/Breadth-first_search

[41] “Random Topology Generator” Retrieved Jun. 20, 2007 from the World Wide Web: http://www.cs.bu.edu/brite/