A NEW RECOVERY SCHEME FOR HANDLING BOTH LINK AND NODE FAILURE IN THE FAST IP NETWORKS

PONDUGALA.BHASKAR RAO*, U. NANAJI#

* Department of CSE, Saint Theresa Institute of Engg. & Technology, Garividi, Vizianagaram, (A.P.), India

# HOD, Prof., Department of CSE, Saint Theresa Institute of Engg. & Technology, Garividi, Viziaanagaram, (A.P.), India

Abstract— In order for IP to become a full-fledged carrier grade transport technology a native IP failure-recovery scheme is necessary that can correct failures in the order of milliseconds. IPFast ReRoute (IPFRR) intends to fill this gap, providing fast, local and proactive handling of failures right in the IP layer. Building on experiences and extensive measurement results collected with a prototype implementation of the prevailing IPFRR technique, not-via, in this paper we identify high address management burden and computational complexity as the major causes of why commercial IPFRR deployment still lags behind, and we present a lightweight Not-via scheme, which, according to our measurements, improves these issues.

Index Terms—**resilience, IP fast reroute, redundant trees**

1.Introduction

IP has come a long way to become a cost-effective bearingPlatform for commercial services. There is however, an important piece still missing in the puzzle a resilience scheme capable to treat failures in tens of milliseconds. In response to this challenge, the Internet Engineering Task Force has initiated the IP Fast Re route framework. To our days, many IPFRR proposals have come to existence, yet the largest industrial backing is undoubtedly behind the technique based on the notion of “Not-via addresses. This paper came into being in reaction to the vast operational experience we gathered on a Not-via-enabled IPFRR test bed deployed at BME-TMIT. We found that not via raises serious address management issues, and it poses substantial additional CPU-load on IP routers. This additional management and the extra computational cost make operators reluctant to adopt IPFRR, despite of its potential benefits. To improve the manageability of Not-via, we present a Lightweight not-via scheme. The main idea is, on the traces of to adopt the concept of node-redundant trees (simply redundant trees in the sequel) for IPFRR and apply them directly to Not-via. As shall be shown,

this modification reduces the number of not-via addresses, cuts the computational complexity down to the level of plain shortest path routing, and it removes many corner cases that plague the original not-via proposal.

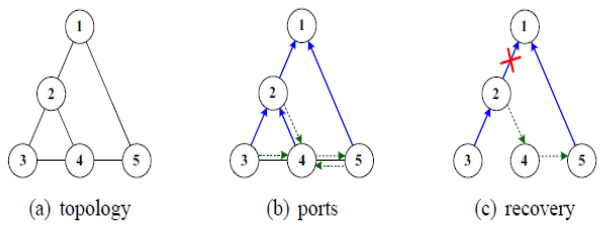
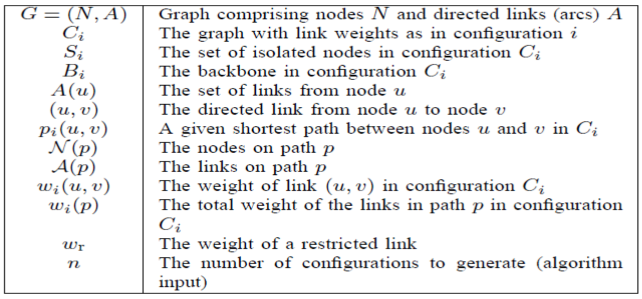


Fig.1

Example of IPFRR (solid/dashed arrows are primary/backup ports).

2. Multiple Routing Configurations (MRC)

MRC is based on building a small set of backup routing configurations that are used to route recovered traffic on alternate paths after a failure. The backup configurations differ from the normal routing configuration in that link weights are set so as to avoid routing traffic in certain parts of the network. We observe that if all links attached to a node are Given sufficiently high link weights, traffic will never be routed through that node. The failure of that node will then only affect traffic that is sourced at or destined for the node itself .Similarly, to exclude a link (or a group of links) from taking part in the routing, we give it infinite weight. The link can then fail without any consequences for the traffic. Our MRC approach is threefold. First, we create a set of backup configurations, so that every network component is excluded from packet forwarding in one configuration. Second, for each configuration, a standard routing algorithm like OSPF is used to calculate configuration specific shortest paths and create forwarding tables in each router, based on the configurations. The use of a standard routing algorithm guarantee sloop-free forwarding within one configuration. Finally, we design a forwarding process that takes advantage of the backup configurations to provide fast recovery from a component failure.



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

Fig 2.1 Table Notation

2.1 MRC Configuration Structure

MRC configurations are defined by the network topology, which is the same in all configurations, and the associated-link weights, which differ among configurations. We formally represent the network topology as a graph G = (N,A), with a set of nodes N and a set of unidirectional links (arcs) In order to guarantee single-fault tolerance, the topology graph G must be bi-connected. A configuration is defined by this topology graph and the associated link weight function .The following figure 2.2

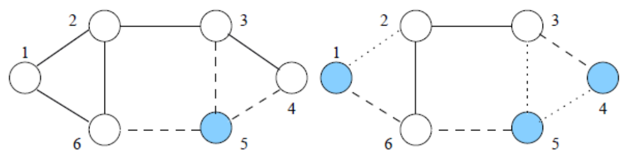


Fig 2.2

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

2.2 Algorithm

The number and internal structure of backup configurations in a complete set for a given topology may vary depending on the construction model.If more configurations clinks and nodes need to be isolated per configuration, giving a richer (more connected) backbone in each configuration. On the other hand, if fewer configurations are constructed, the state requirement for the backup routing information storage is reduced. However, calculating the minimum umber of configurations for a given topology graph is computationally demanding. One solution would be to find all valid configurations for the input consisting of the topology graph G and its associated normal link weights w0, and then find the complete set of configurations with lowest cardinality. Finding this set would involve solving the Set Cover problem, which is known to be NP-complete [13]. Instea we present a heuristic algorithm that attempts to make all nodes and links in an arbitrary bi-connected topology isolated. Our algorithm takes as input the directed graph G and the number n of backup configurations that is intended created. If the algorithm terminates successfully, its output is a complete set of valid backup configurations. The algorithm is agnostic to the original link weights w0, and assigns new link weights only to restricted and isolated links in the backup configurations. For a sufficiently high n, the algorithm will always terminate successfully, as will be further discussed in This algorithm isolates all nodes in the network, and hence requires a bi-connected as input. Topologies where the failure of a single node disconnects the network can be processed by simply ignoring such nodes, which are then left unprotected. The algorithm can be implemented either in a network management system, or in the routers. As long as all routers have the same view of the network topology, they will compute the same set of backup configurations.

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

Algorithm 1: Creating backup configurations.

```
1  for $i \in \{1 \dots n\}$ do
2      $C_i \leftarrow (G, w_0)$
3      $S_i \leftarrow \emptyset$
4      $B_i \leftarrow C_i$
5  end
6  $Q_n \leftarrow N$
7  $Q_a \leftarrow \emptyset$
8  $i \leftarrow 1$
9  while $Q_n \neq \emptyset$ do
10     $u \leftarrow$ first $(Q_n)$
11     $j \leftarrow i$
12     repeat
13         if connected$(B_i \setminus (\{u\}, A(u)))$ then
14             $C_{tmp} \leftarrow$ isolate$(C_i, u)$
15             if $C_{tmp} \neq$ null then
16                 $C_i \leftarrow C_{tmp}$
17                 $S_i \leftarrow S_i \cup \{u\}$
18                 $B_i \leftarrow B_i \setminus (\{u\}, A(u))$
19         $i \leftarrow (i \bmod n) + 1$
20     until $u \in S_i$ or $i$=$j$
21     if $u \notin S_i$ then
22         Give up and abort
23 end
```

a) Main loop: n backup configurations are created as copies of the normal configuration. A queue of nodes (Qn) and a queue of links (Qa) are initiated. The node queue contains all nodes in an arbitrary sequence. The link queue is initially empty, but all links in the network will have to pass through it. Method first returns the first item in the queue, removing it from the queue. When a node u attempted isolated in a backup configuration Ci, it is first tested that doing so will not disconnect Bi according to definition If the connectivity test is positive, function isolate is called, which attempts to find a valid assignment of isolated and restricted links for node u .

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

```
Function isolate($C_i, u$)
1  $Q_a \leftarrow Q_a + (u,v), \forall(u,v) \in A(u)$
2  while $Q_a \neq \emptyset$ do
3      $(u,v) \leftarrow$ first $(Q_a)$
4      if $\exists j : v \in S_j$ then
5          if $w_j(u,v) = w_r$ then
6              if $\exists(u,x) \in A(u) \setminus (u,v) : w_i(u,x) \neq \infty$ then
7                  $w_i(u,v) \leftarrow w_i(v,u) \leftarrow \infty$
8              else
9                  return null
10         else if $w_j(u,v) = \infty$ and $i \neq j$ then
11             $w_i(u,v) \leftarrow w_i(v,u) \leftarrow w_r$
12     else
13         if $\exists(u,x) \in A(u) \setminus (u,v) : w_i(u,x) \neq \infty$ then
14             $w_i(u,v) \leftarrow w_i(v,u) \leftarrow \infty$
15         else
16             $w_i(u,v) \leftarrow w_i(v,u) \leftarrow w_r$
17             $Q_n \leftarrow v + (Q_n \setminus v)$
18             $Q_a \leftarrow (v,u)$
19 end
20 return $C_i$
```

3. Alternative Path Setup for Load Balancing

In Pro-active Failure Recovery (PFR), routers try to detect and recover failures locally rather than relying on network wide routing convergence. Generally, a link failure will trigger the physical detection, which reports the event to the IP layer immediately. A router can locally find multiple paths forwarding the affected traffic on the failed link(s) via its neighbors. Figure 3.1(a) illustrates this by a simple example. Node *u*'s next hop on its shortest path towards *d* is *f*. Node *u* also maintains alternative paths to reach *d* through other neighbors such as *a*1, *a*2, *a*n. When a local link *u* → *f* fails, *u* will shift its affected traffic to alternative paths, e.g., via *a*1, without waiting for the completion of routing convergence. The diverted packets are marked (e.g., using the Type Of Service (TOS) field in the packet), so that the downstream routers will know that the packet is diverted and can make appropriate forwarding decisions make appropriate forwarding decisions PFR works mainly in intra-domain routing such as OSPF(Open Shortest Path First) and IS-IS(Intermediate System-Intermediate System). It can handle single-link failures and multiple-link failures that do not affect each other. In other words, as long as the multiple failures do not disable all the alternative paths and the packets diverted to the alternative paths do not encounter another failure again, PFR works fine. Our work is within the same intra-domain routing scope and inheres the same limitation regarding multiple link failures. Our contribution is to set up and utilize multiple alternative paths (e.g., paths via *a*1, *a*2, . *a*n in Figure 1(a)) efficiently

On one hand, given a destination, there can be a huge number of alternative paths from a router to reach the destination. It is not only impractical but also unnecessary to find all the possible alternative paths and store them. On the other hand, the number of alternative paths cannot be too small, otherwise they will not be able to serve the purpose of failure recovery and load balancing. Therefore, we need to find ***adequate*** number of ***loop-free*** alternative paths without significant computation overhead.

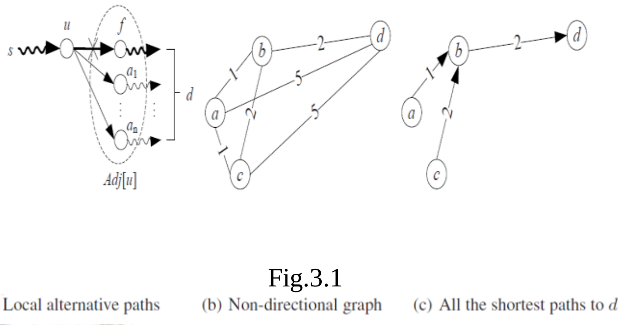


Fig.3.1
(a) Local alternative paths (b) Non-directional graph (c) All the shortest paths to *d*

4. Packet Forwarding Process in Network

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

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

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

Similarly, let C(u, v) denote the backup configuration where the link (u, v) is isolated, i.e., $C(u,v) = C_i \Leftrightarrow w_i(u,v) = \infty$ Assuming that node d is the egress (or the destination) in the local network domain, we can distinguish between two cases. If v 6= d, forwarding can be done in configuration C(v), where both v and (u, v) will be avoided. In the other case, v = d, the challenge is to provide recovery for the failure of link (u, v) when node v is operative. Our strategy is to forward the packet using a path to v that does not contain (u, v).

Furthermore, packets that have changed configuration before (their configuration ID is different than the one used in C0), and still meet a failed component on their forwarding path, must be discarded. This way packets loops are avoided, also in the case that node d indeed has failed. The steps that are taken in the forwarding process by the detecting node u are

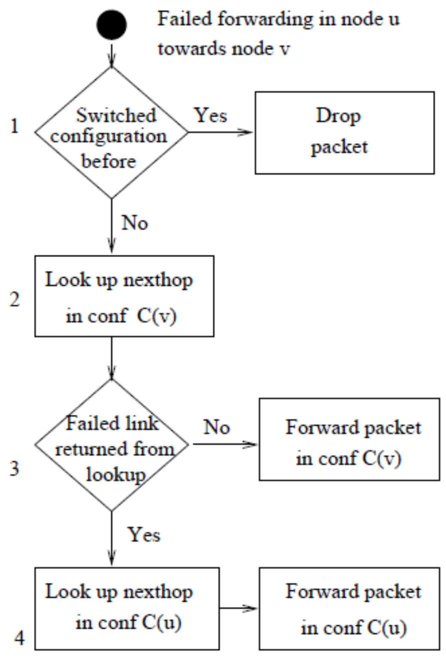

Failed forwarding in node u towards node v
1
Switched configuration before
Yes
Drop packet
No
2
Look up nexthop in conf C(v)
3
Failed link returned from lookup
No
Forward packet in conf C(v)
Yes
4
Look up nexthop in conf C(u)
Forward packet in conf C(u)

Fig. 4.1. Packet forwarding state diagram.

5. Implementation Process

The forwarding process can be implemented in the routing Equipment as presented above, requiring the detecting node u to know the backup configuration C(v) for each of its neighbors. Node u would then perform at most two additional next-hop look-ups in the case of a failure. However, all nodes in the network have full knowledge of the structure of all backup configurations. Hence, node u can determine in advance the correct backup configuration to use if the normal next hop for a destination d has failed. This way the forwarding decision at the point of failure can be simplified at the cost of storing the identifier of the correct backup configuration to use for each destination and failing neighbor. For the routers to make a correct forwarding decision, each packet must carry information about which configuration it belongs to. This information can be either explicit or implicit. An explicit approach could be to use a distinct value in theDSCP field of the IP header to identify the configuration. As we will see shortly, a very limited number of backup configurations are needed to guarantee recovery from all single link or node failures, and hence the number of needed values would be small. A more implicit approach would be to assign a distinct local IP address space for each backup configuration. Each node in the IGP cloud would get a separate address in each configuration. The detecting node could then encapsulate.

6. EMRC Approach

Enhanced Multiple Routing Configuration (EMRC) is a threefold approach. First, a set of backup configurations are created, such that every network component is excluded from packet forwarding in one configuration. Second, for each configuration, a routing algorithm like OSPF is used to calculate configuration specific shortest paths and create forwarding tables in each router. Third, a forwarding process is designed which uses the backup configurations to provide fast recovery from a component failure.

6.1 Forwarding Procedure for EMRC

When we want to transmit any data from source to destination in the network, first we identify the source node and destination node, after that we look at the shortest path in between them in the original routing table and the data packets are transmitted by using that shortest route.

When a data packet reaches a point of failure, the node adjacent to the failure, called the detecting node stops the transmission. At that time, the detecting node gives the timeslot to failure recovery before shifting to the backup route.

Within the timeslot, if the failure is recovered then data is transmitted by using the original route only and if the failure is not recovered, then the detecting node is responsible for finding a backup configuration where the failed component is isolated. The detecting node marks the packet as belonging to this configuration, and forwards the packet. From the packet marking, all transit routers identify the packet with the selected backup configuration, and forward it to the egress node avoiding the failed component. Packet marking is most easily done by using specific values in the DSCP field in the IP header. If this is not possible, other packet marking strategies like IPv6 extension headers or using a private address space and tunneling could be used.

During the backup route transmission, the detecting node sends the probing signals for failure recovery and if failure is recovered, then backup route transmission is stopped and the data packets are transmitted by reusing the original route. By reusing the original route we can improve the fastness of routing, since the backup route is longer than the original route.

If a failure lasts for more than a specified time interval, a normal reconvergence will be triggered. EMRC does not interfere with this convergence process, or make it longer than normal. However, EMRC gives continuous packet forwarding during the convergence, and hence makes it easier to use mechanisms that prevent micro-loops during convergence, at the cost of longer convergence times. If a failure is deemed permanent, new configurations must be generated based on the altered topology.

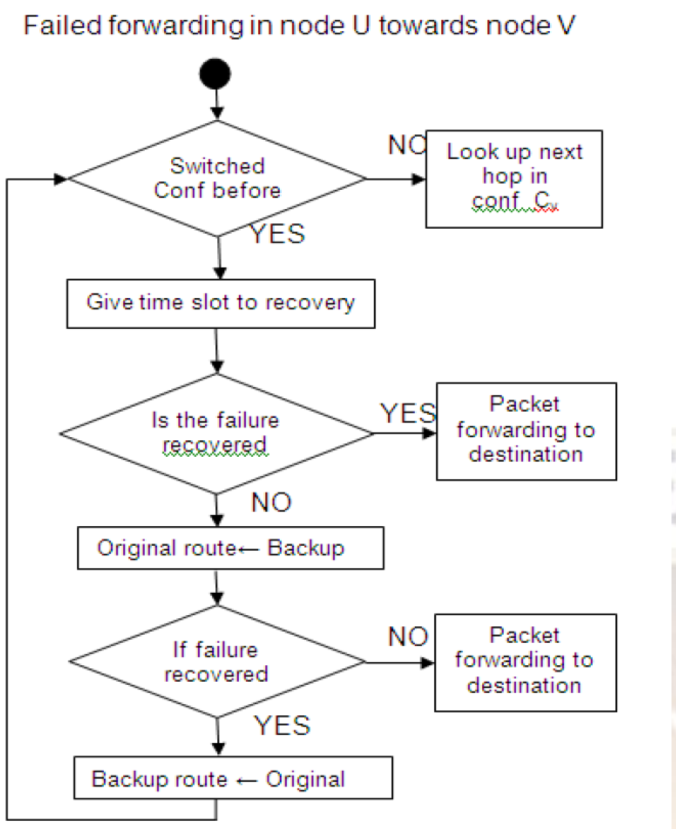

Failed forwarding in node U towards node V
Switched Conf before
NO
Look up next hop in conf. Cv
YES
Give time slot to recovery
Is the failure recovered
YES
Packet forwarding to destination
NO
Original route← Backup
If failure recovered
NO
Packet forwarding to destination
YES
Backup route ← Original

Fig 6.1

6.2 Comparison of MRC and EMRC

EMRC is developed from MRC. So, all the processes in EMRC such as backup route finding, shortest path finding and forwarding is same as the MRC. These all are explained and compared by using the following in FIGURE 3.

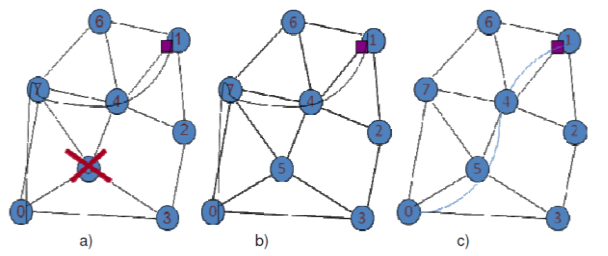


Fig 6.2

Selection of routes in MRC and EMRC a) At the time of failure occurrence in MRC and EMRC.b) After the failure recovery in MRC c) After the failure recovery in EMRC.

7. Experimental Evaluation

To evaluate our load aware construction algorithm, we compute the worst case load on each link after a link failure, and compare it to the results achieved by the original algorithm. We reuse the evaluation framework from Sec. V, and set the critical link set size k to 20.In the top panel in Fig. 9, we show the worst case link loads for the load aware MRC ("Optimized MRC") and after a full IGP re-convergence on the new topology. The links are sorted by the load in the failure-free case. The top panel in Fig. 9 is directly comparable to Fig. 8. We see that the worst case load peaks for the optimized MRC are somewhat reduced compared to the standard MRC. The maximum link load after the worst case link failure has been reduced from 118% to 91%, which is better than what is achieved after a full IGP re-convergence. This is possible since the re-converged network will choose the shortest available path, while MRC in this case manages to route the recovered traffic over less utilized links. The effect of the proposed recovery load balancing is High lighted in the bottom panel of Fig. 9, where the optimized and standard MRC are directly compared. Here, the links are sorted by their load after a worst case failure using standard MRC.

We see how the optimized MRC often manages to route traffic over less utilized links after the failure of a heavily loaded link. Note that the optimizations described here will only have an effect if the network topology allows more than one possible backup path after a failure. We have also run our optimizations on less connected networks than COST239, without achieving any significant improvements over the method described inSec. III. For small networks like COST239, our link weight optimization is performed in seconds or minutes. For larger networks the optimizations can take several hours, and should be conducted as a background refinement process. Note that updating the link weights in the backup configurations can be done without consequences for the traffic, since no traffic is routed there during normal operation.

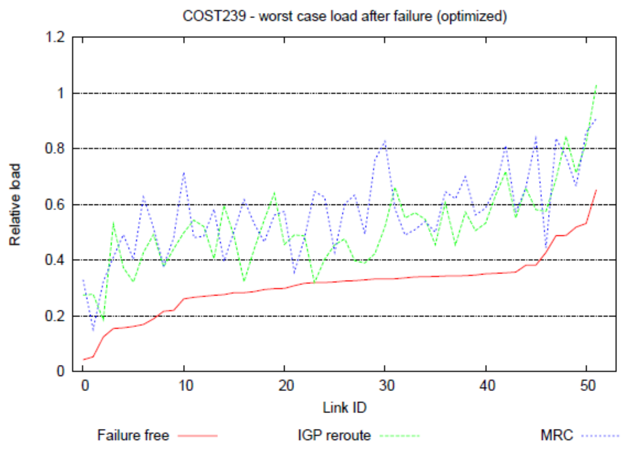


Fig.7.1

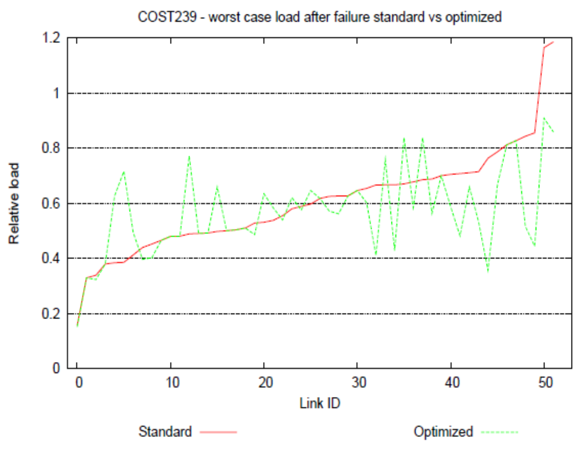


Fig 7.2
Load on all unidirectional links in the COST239 network after the worst case link failure. a)Optimized MRC vs. complete IGP rerouting)Standard vs. optimized MRC.

8. Conclusion

We have presented Multiple Routing Configurations as anapproach to achieve fast recovery in IP networks. MRC isbased on providing the routers with additional routing configurations, allowing them to forward packets along routes that avoid a failed component. MRC guarantees recovery fromany single node or link failure in an arbitrary bi-connectednetwork. By calculating backup configurations in advance, and operating based on locally available information only, MRC can act promptly after failure discovery.MRC operates without knowing the root cause of failure,i.e., whether the forwarding disruption is caused by a node or link failure. This is achieved by using careful link weight assignment according to the rules we have described. The link weight assignment rules also provide basis for the specification of a forwarding procedure that successfully solves the last hop problem.The performance of the algorithm and the forwarding mechanism has been evaluated using simulations.We Have shown that MRC scales well: 3 or 4 backup configurations is typically enough to isolate all links and nodes in our test topologies. MRC backup path lengths are comparable to the optimal backup path lengths—MRC backup paths are typically zero to two hops longer. We have evaluated the effect MRC has on the load distribution in the network while traffic is routed in the backup configurations, and we have proposed a method that minimizes the risk of congestion after a link failure if we have an estimate of the demand matrix. In the COST239 network, this approach gave a maximum link load after the worst case link failure that was even lower than after a full IGP re-convergence on the altered topology. MRC thusachieves fast recovery with a very limited performance penalty.

9. References

[1] D. D. Clark, "The design philosophy of the DARPA internet protocols," ***SIGCOMM, Computer Communications Review*, vol. 18, no. 4, pp. 114, Aug. 1988.**
[2] A. Basu and J. G. Riecke, "Stability issues in OSPF routing," in *Proceedings of SIGCOMM*, San Diego, California, USA, Aug. 2001, pp. 225–236.
[3] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian, "Delayed Internet Routing Convergence," *IEEE/ACM Transactions on Networking*, vol. 9, no. 3, pp. 293–306, June 2001.
[4] C. Boutremans, G. Iannaccone, and C. Diot, "Impact of link failures on VoIP performance," in *Proceedings of International Workshop on Network and Operating System Support for Digital Audio and Video*, 2002, pp. 63–71.
[5] D. Watson, F. Jahanian, and C. Labovitz, "Experiences with monitoring OSPF on a regional service provider network," in *ICDCS '03: Proceedings of the 23rd International Conference on Distributed Computing Systems*. Washington, DC, USA: IEEE Computer Society, 2003,

P.Bhaskar Rao received the MCA Degree from Nagarjuna Uninersity Guntur in 2005 and He is currently pursuing M.Tech in the Department Of Computer Science and Engineering in Saint Theresa Institute Of Engg & Tech Garividi, Vizianagaram, Of JNTUK Affiliation. His research interests include Computer Networks and Network Security.

Uppe.Nanaji received the B. Tech degree from JNTU, Hyderabad, India and the M. Tech degree in Computer Science Technology from GITAM College Of Engg Of Andhra University Affiliation in Vishakhapatnam in 2003, and he is currently pursuing the Ph. D in Computer Networks from Andhra University Visakhapatnam..His research interests include Computer Networks & Data Ware Housing