Fast IP Network Recovery using Multiple Routing Configurations

Amund Kvalbein*, Audun Fosselie Hansen*†, Tarik Čičić*, Stein Gjessing* and Olav Lysne*
*Simula Research Laboratory, Oslo, Norway
† Telenor R&D, Oslo, Norway
Email: {amundk, audunh, tarikc, steing, olavly}@simula.no

***Abstract*— As the Internet takes an increasingly central role in our communications infrastructure, the slow convergence of routing protocols after a network failure becomes a growing problem. To assure fast recovery from link and node failures in IP networks, we present a new recovery scheme called Multiple Routing Configurations (MRC). MRC is based on keeping additional routing information in the routers, and allows packet forwarding to continue on an alternative output link immediately after the detection of a failure. Our proposed scheme guarantees recovery in all single failure scenarios, using a single mechanism to handle both link and node failures, and without knowing the root cause of the failure. MRC is strictly connectionless, and assumes only destination based hop-by-hop forwarding. It can be implemented with only minor changes to existing solutions. In this paper we present MRC, and analyze its performance with respect to scalability, backup path lengths, and load distribution after a failure.**

I. Introduction

In recent years the Internet has been transformed from a special purpose network to an ubiquitous platform for a wide range of everyday communication services. The demands on Internet reliability and availability have increased accordingly. A disruption of a link in central parts of a network has the potential to affect hundreds of thousands of phone conversations or TCP connections, with obvious adverse effects.

The ability to recover from failures has always been a central design goal in the Internet [1]. IP networks are intrinsically robust, since IGP routing protocols like OSPF are designed to update the forwarding information based on the changed topology after a failure. This re-convergence assumes full distribution of the new link state to all routers in the network domain. When the new state information is distributed, each router individually calculates new valid routing tables.

This network-wide IP re-convergence is a time consuming process, and a link or node failure is typically followed by a period of routing instability. During this period, packets may be dropped due to invalid routes. This phenomenon has been studied in both IGP [2] and BGP context [3], and has an adverse effect on real-time applications [4]. Events leading to a re-convergence have been shown to occur frequently, and are often triggered by external routing protocols [5].

Much effort has been devoted to optimizing the different steps of the convergence of IP routing, i.e., detection, dissemination of information and shortest path calculation, but the convergence time is still too large for applications with real time demands [6]. A key problem is that since most network failures are short lived [7], too rapid triggering of the re-convergence process can cause route flapping and increased network instability [2].

The IGP convergence process is slow because it is *reactive* and *global*. It reacts to a failure after it has happened, and it involves all the routers in the domain. In this paper we present a new scheme for handling link and node failures in IP networks. Multiple Routing Configurations (MRC) is *proactive* and *local*, which allows recovery in the range of milliseconds. MRC allows packet forwarding to continue over pre-configured alternative next-hops immediately after the detection of the failure. Using MRC as a first line of defense against network failures, the normal IP convergence process can be put on hold. This process is then initiated only as a consequence of non-transient failures. Since no global re-routing is performed, fast failure detection mechanisms like fast hellos or hardware alerts can be used to trigger MRC without compromising network stability [8]. MRC guarantees recovery from any single link or node failure, which constitutes a large majority of the failures experienced in a network [7].

The main idea of MRC is to use the network graph and the associated link weights to produce a small set of backup network configurations. The link weights in these backup configurations are manipulated so that for each link and node failure, and regardless of whether it is a link or node failure, the node that detects the failure can safely forward the incoming packets towards the destination. MRC assumes that the network uses shortest path routing and destination based hop-by-hop forwarding.

In the literature, it is sometimes claimed that the node failure recovery implicitly addresses link failures too, as the adjacent links of the failed node can be avoided. This is true for intermediate nodes, but the destination node in a network path must be reachable if operative ("The last hop problem", [9]). MRC solves the last hop problem by strategic assignment of link weights between the backup configurations.

MRC has a range of attractive features:

- It gives almost continuous forwarding of packets in the case of a failure. The router that detects the failure initiates a local rerouting immediately, without communicating with the surrounding neighbors.
- MRC helps improve network availability through sup-

pression of the re-convergence process. Delaying this process is useful to address transient failures, and pays off under many scenarios [8]. Suppression of the re-convergence process is further actualized by the evidence that a large proportion of network failures is short-lived, often lasting less than a minute [7].

- MRC uses a single mechanism to handle both link and node failures. Failures are handled locally by the detecting node, and MRC always finds a route to the destination (if operational).
- MRC makes no assumptions with respect to the *root cause of failure*, e.g., whether the packet forwarding is disrupted due to a failed link or a failed router. Regardless of this, MRC guarantees that there exists a valid, pre-configured next-hop to the destination.
- An MRC implementation can be made without major modifications to existing IGP routing standards. IETF recently initiated specifications of multi-topology routing for OSPF and IS-IS, and this approach seems well suited to implement our proposed backup configurations [10], [11], [12].

The concept of multiple routing configurations and its application to network recovery is not new. Our main inspiration has been a layer-based approach used to obtain deadlock-free and fault-tolerant routing in irregular cluster networks based on a routing strategy called Up*/Down* [13]. General packet networks are not hampered by deadlock considerations necessary in interconnection networks, and hence we generalized the concept in a technology independent manner and named it Resilient Routing Layers [14][15]. In the graph-theoretical context, RRL is based on calculating spanning sub topologies of the network, called layers. Each layer contains all nodes but only a subset of the links in the network.

The work described in this paper differs substantially from RRL in that we do not alter topologies by removing links, but rather manipulate link weights to meet goals of handling both node and link failures without needing to know the root cause of the failure. In MRC, all links remain in the topology, but in some configurations, some links will not be selected by shortest path routing mechanisms due to high weights.

The rest of this paper is organized as follows. In Sec. II we describe the basic concepts and functionality of MRC. An algorithm used to create the needed backup configurations is presented in Sec. III. Then, in Sec. IV, we explain how the generated configurations can be used to forward the traffic safely to its destination in case of a failure. In Sec. V, we present performance evaluations of the proposed method, and in Sec. VI, we discuss related work. Finally, in Sec. VII, we conclude and give some prospects for future work.

II. MRC Overview

MRC is based on using a small set of backup routing configurations, where each of them is resistant to failures of certain nodes and links. Given the original network topology, a *configuration* is defined as a set of associated link weights. In a configuration that is resistant to the failure of a particular node n, link weights are assigned so that traffic routed according to this configuration is never routed through node n. The failure of node n then only affects traffic that is sent from or destined to n. Similarly, in a configuration that is resistant to failure of a link l, traffic routed in this configuration is never routed over this link, hence no traffic routed in this configuration is lost if l fails. In MRC, node n and link l are called *isolated* in a configuration, when, as described above, no traffic routed according to this configuration is routed through n or l.

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

Fig. 1a illustrates a configuration where node 5 is isolated. In this configuration, the weight of the stapled links is set so high that only traffic sourced by or destined for node 5 will be routed over these links, which we denote *restricted* links.

Node failures can be handled through blocking the node from transiting traffic. This node-blocking will normally also protect the attached links. But a link failure in the last hop of a path can obviously not be recovered by blocking the downstream node (ref. “the last hop problem”). Hence, we must make sure that, in one of the backup configurations, there exists a valid path to the last hop node, without using the failed link. A link is isolated by setting the weight to infinity, so that any other path would be selected before one including that link. Fig. 1b shows the same configuration as before, except now link 3-5 has been isolated (dotted). No traffic is routed over the isolated link in this configuration; traffic to and from node 5 can only use the restricted links.

In Fig. 1c, we see how several nodes and links can be isolated in the same configuration. In a backup configuration like this, packets will *never* be routed over the isolated (dotted) links, and *only in the first or the last hop* be routed over the restricted (dashed) links.

Some important properties of a backup configuration are worth pointing out. First, all non-isolated nodes are internally connected by a sub-graph that does not contain any isolated or restricted links. We denote this sub-graph as the *backbone* of the configuration. In the backup configuration shown in Fig. 1c, nodes 6, 2 and 3 with their connecting links constitute this backbone. Second, all links attached to an isolated node are either isolated or restricted, but an isolated node is always directly connected to the backbone with at least one restricted link. These are important properties of all backup configurations, that are further discussed in Sec. III, where we explain how backup configurations can be constructed.

Using a standard shortest path calculation, each router creates a set of configuration-specific forwarding tables. For

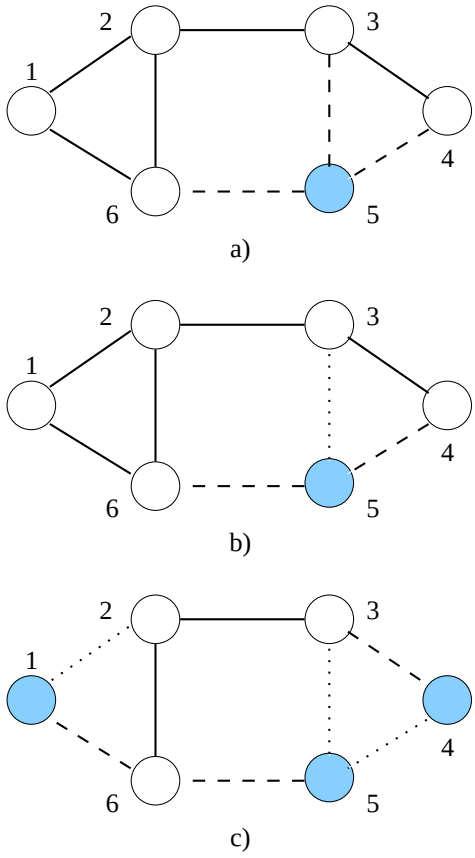

Fig. 1. a) Node 5 is isolated (shaded color) by setting a high weight on all its connected links (stapled). Only traffic to and from the isolated node will use these restricted links. b) The link from node 3 to node 5 is isolated by setting its weight to infinity, so it is never used for traffic forwarding (dotted). c) A configuration where nodes 1, 4 and 5, and the links 1-2, 3-5 and 4-5 are isolated.

simplicity, we say that a packet is forwarded according to a configuration, meaning that it is forwarded using the forwarding table calculated based on that configuration.

When a router detects that a neighbor can no longer be reached through one of its interfaces, it does not immediately inform the rest of the network about the connectivity failure. Instead, packets that would normally be forwarded over the failed interface are marked as belonging to a backup configuration, and forwarded on an alternative interface towards its destination. The selection of the correct backup configuration, and thus also the backup next-hop, is detailed in Sec. IV. The packets must be marked with a configuration identifier, so the routers along the path know which configuration to use. Packet marking is most easily done by using 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 tunnelling (as proposed in [16]) can be imagined.

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

III. Generating Backup Configurations

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

A. Configuration Constraints

To guarantee single-failure tolerance and consistent routing, the backup configurations used in MRC must adhere to the following requirements:

1) A node must not carry any transit traffic in the configuration where it is isolated. Still, traffic must be able to depart from and reach an isolated node.
2) A link must not carry any traffic at all in the configuration where it is isolated.
3) In each configuration, all node pairs must be connected by a path that does not pass through an isolated node or an isolated link.
4) Every node and every link must be isolated in at least one backup configuration.

The first requirement decides what weights must be put on the restricted links attached to an isolated node. To guarantee that no path will go through an isolated node, it suffices that the restricted links have a weight W of at least the sum of all link weights w in the original configuration:

$$W > \sum_{e_{i,j} \in \mathcal{E}} w_{i,j} \tag{1}$$

This guarantees that any other path between two nodes in the network will be chosen by a shortest path algorithm before one passing through the isolated node. Only packets sourced by or destined for the isolated node itself will traverse a restricted link with weight W, as they have no shorter path. With our current algorithm, restricted and isolated links are given the same weight in both directions in the backup configurations, i.e., we treat them as undirected links. However, this does not prevent the use of independent link weights in each direction in the default configuration.

The second requirement implies that the weight of an isolated link must be set so that traffic will *never* be routed over it. Such links are given infinite weight.

Given these restrictions on the link weights, we now move on to show how we can construct backup configurations that adhere to the last two requirements stated above.

B. Algorithm

We now present an algorithm, designed to make all nodes and links in a arbitrary biconnected graph isolated. Our algorithm takes as input the undirected weighted graph $\mathcal{G}$, and the number n of backup configurations that is intended created.

TABLE I

NOTATION

$\mathcal{G}(\mathcal{V},\mathcal{E})$	Graph with nodes $\mathcal{V}$ and undirected links $\mathcal{E}$
$\mathcal{G}_p$	The graph with link weights as in configuration p
$\mathcal{S}_p$	Isolated nodes in configuration p
$\mathcal{E}_i$	All links from node i
$e_{i,j}$	Undirected link from node i to node j ($e_{i,j} = e_{j,i}$)
$w^p_{i,j}$	Weight of link $e_{i,j}$ in configuration p
n	Number of configurations to generate (input)
W	Weight of restricted links

It 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. With our algorithm, all nodes and links in the network are isolated in exactly one configuration.

The third property above results in the following two invariants for our algorithm, which must be evaluated each time a new node and its connected links are isolated in a configuration:

1) A configuration must contain a backbone
2) All isolated nodes in a configuration must be directly connected to the backbone through at least one restricted link.

The first invariant means that when a new node is isolated, we must make sure that the sub-graph of non-isolated nodes is not divided. If making a node isolated breaks any of these two requirements, then the node cannot be isolated in that configuration.

When isolating a node, we also isolate as many as possible of the connected links, without breaking the second invariant above. A link is always isolated in the same configuration as one of its attached nodes. This is an important property of the produced configurations, which is taken advantage of in the forwarding process described in Sec. IV.

Now we specify the configuration generation algorithm in detail, using the notation shown in Tab. I.

When a node v_i is attempted isolated in a backup configuration p, it is first tested that doing so will not break the first invariant above. The `div` method (for "divide") at line 11 decides this by testing that each of v_i's neighbors can reach each other without passing through v_i, an isolated node, or an isolated link in configuration p.

Along with v_i, as many as possible of its attached links are isolated. We run through all the attached links (line 13). The node v_j in the other end of the link may or may not be isolated in some configuration already (line 15). If it is, we must decide whether the link should be isolated along with v_i (line 20), or if it is already isolated in the configuration where v_j is isolated (line 27). A link must always be isolated in the same configuration as one of its end nodes. Hence, if the link was not isolated in the same configuration as v_j, it *must* be isolated along with node v_i.

Before we can isolate the link along with v_i, we must test (line 18) that v_i will still have an attached non-isolated link, according to the second invariant above. If this is not the case, v_i can not be isolated in the present configuration (line 23). Giving up the node in the present configuration means restarting the outer loop (line 9). It is important to note that this also involves resetting all changes that has been made in configuration p trying to isolate v_i.

In the case that the neighbor node v_j was *not* isolated in any configuration (line 29), we isolate the link along with v_i if possible (line 34). If the link can not be isolated (due to the second invariant), we leave it for node v_j to isolate it later. To make sure that this link can be isolated along with v_j, we must process v_j next (line 39, selected at line 7), and link $e_{j,i}$ must be the first among $\mathcal{E}_j$ to be processed (line 41, selected at line 13). This is discussed further in Sec. III-C below.

If v_i was successfully isolated, we move on to the next node. Otherwise, we keep trying to isolate v_i in every configuration, until all configurations are tried (line 9). If v_i could not be isolated in any configuration, requirement four in Sec. III-A could not be fulfilled. The algorithm will then terminate with an unsuccessful result (line 48). This means that our algorithm could not isolate all network elements using the required number of configurations, and a higher number of configurations must be tried. Note also that our heuristic algorithm does not necessarily produce the theoretical minimum number of

Algorithm 1: Creating Backup Configurations

```
1  for p ∈ {0 ... n − 1} do
2      G_p ⇐ G
3      S_p ⇐ ∅
4  end
5  p ⇐ 0
7  forall v_i ∈ V do
9      while v_i ∉ S_p and not all configurations
       tried do
11         if !div(v_i, G_p) then
13             forall e_{i,j} ∈ E_i do
15                 if ∃q : v_j ∈ S_q then
16                     if w^q_{i,j} = W then
18                         if ∃e_{i,k} ∈ E_i \ e_{i,j} : w^p_{i,k} ≠ ∞ then
20                             w^p_{i,j} ⇐ ∞
21                         else
23                             break 9
24                     else if w^q_{i,j} = ∞ and p ≠ q then
27                         w^p_{i,j} ⇐ W
29
30                 else
32                     if ∃e_{i,k} ∈ E_i \ e_{i,j} : w^p_{i,k} ≠ ∞ then
34                         w^p_{i,j} ⇐ ∞
35                     else
37                         w^p_{i,j} ⇐ W
39                         firstInNodeQ(v_j)
41                         firstInLinkQ(v_j, e_{j,i})
43             commit edge weight changes
44             S_p ⇐ S_p ∪ v_i
45         p++ mod n
46     if v_i ∉ S_p then
48         Give up and abort
49 end
```

backup configurations.

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

C. Termination

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

For an intuitive proof of this, look at a situation where the number of configurations created is $|\mathcal{V}|$. In this case, the algorithm will only isolate one node in each backup configuration. In biconnected topologies any node can be removed, i.e. isolated, without disconnecting the network, and hence invariant 1 above is not violated. Along with a node v_i, all attached links except one ($e_{i,j}$) can be isolated. By forcing node v_j to be the next node processed (line 39), and the link $e_{j,i}$ to be first among $\mathcal{E}_j$ (line 41), we guarantee that $e_{j,i}$ and v_j can be isolated in the next configuration. This can be repeated until we have configurations so that every node and link is isolated. This holds also for the last node processed, since its last link will always lead to a node that is already isolated in another configuration.

A ring topology is a worst-case example of a topology that would need $|\mathcal{V}|$ backup configurations to isolate all network elements.

IV. Local Forwarding Process

The algorithm presented in Sec. III creates a set of backup configurations. Based on these, a standard shortest path algorithm is used in each configuration, to calculate configuration specific forwarding tables. In this section, we describe how these forwarding tables are used to avoid a failed component.

When a packet reaches a point of failure, the node adjacent to the failure, called the *detecting node*, is responsible for finding the configuration where the failed component is isolated, and to forward the packet according to this configuration. With our proposal, the detecting node must find the correct configuration without knowing the root cause of failure.

A node must know in which configuration the downstream node of each of its network interfaces is isolated. Also, it must know in which configuration it is isolated itself. This information is distributed to the nodes in advance, during the configuration generation process.

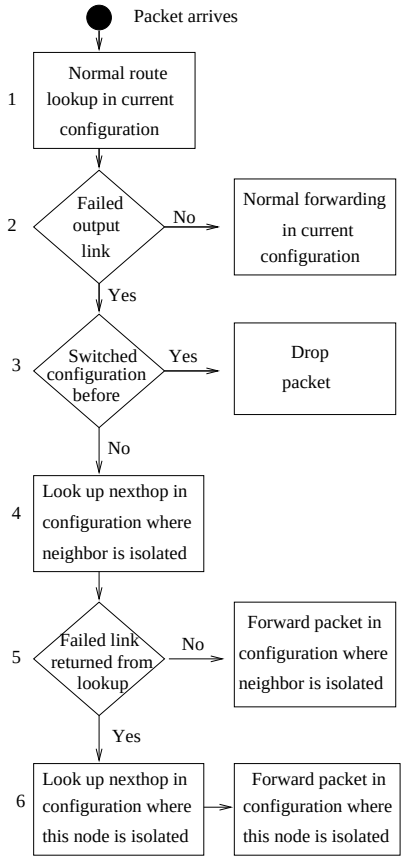

Fig. 2. State diagram for a node's packet forwarding.

The flow diagram in Fig. 2 shows the steps that are taken in a node's forwarding process. First, packets that are not affected by the failure, are forwarded as normal (step 2). Special measures are only taken for packets that would normally be forwarded through a broken interface.

In step 3, packets that are already routed according to a backup configuration, i.e., they have been marked with a backup configuration identifier by another node, are discarded. Reaching a point of failure for the second time, means either that the egress node has failed, or the network contains multiple failed elements. To avoid looping between configurations, a packet is allowed to switch configuration only once. To allow protection against multiple failures, we could imagine a scheme where packets are allowed to switch configurations more than once. A separate mechanism would then be needed to keep packets from looping between two configurations, e.g. only allowing packets to be switched to a configuration with a higher ID.

We then make a next hop lookup in the configuration where the neighbor is isolated, in step 4. If the same broken link is not returned from this lookup, we mark the packet with the correct configuration identifier, and forward the packet in this configuration (step 5). The packet is then guaranteed to reach its egress node, without being routed through the point of failure again. Only if the neighbor is the egress node for

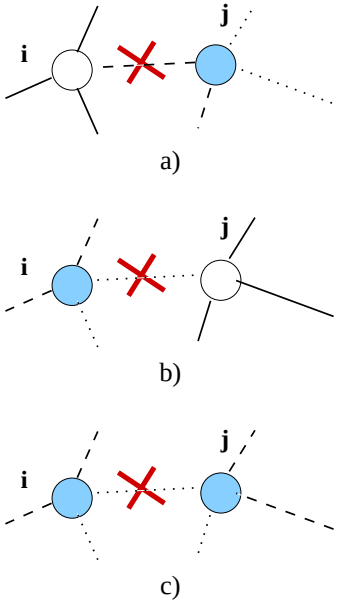

Fig. 3. Isolated nodes are given a shaded color. When there is an error in the last hop, a packet must be forwarded in the configuration where the connecting link is isolated (the link is then dotted).

the packet, and the neighbor is indeed dead, will the packet reach a dead interface for the second time (in a single failure scenario). It will then be discarded in another node.

If, however, the dead link *is* returned from the lookup in the configuration where the neighbor is isolated, we know that the neighbor node must be the egress node for the packet, since packets are never routed through an isolated node. In this case, a lookup in the configuration where the detecting node itself is isolated must be made (step 6). Remember that a link is always isolated in the same configuration as one of its attached nodes. Hence, the dead link can never be returned from this lookup. Again, if the neighbor (egress) node is indeed dead, the packet will be discarded in another node upon reaching a dead interface for the second time.

A. Last Hop Failure Example

For an example of how packet forwarding is done in the case of a failure in the last hop, consider the situation depicted in Fig. 3, where a packet reaches a dead interface in flight from node i to egress node j.

In the last hop, packets will be forwarded in the configuration where either node i or node j is isolated, depending on where the link between them is isolated. In Fig. 3a, the link is not isolated in the same configuration as node j. A route lookup in this configuration will return the same broken link. Hence, a lookup must be made in the configuration where node i is isolated, shown in Fig. 3b.

Note that if nodes i and j are isolated in the same configuration, the link connecting them is also isolated in that configuration, as shown in Fig. 3c. Packets will then always reach the egress in that configuration, even if it is the last hop link that fails, unless, of course, the egress node itself has failed.

B. Implementation issues

While the backup configurations can be generated off line, and information can be represented in the network using Multi Topology routing mechanisms [10], [11], the described forwarding process needs additional software functionality in the routers. The described forwarding process consists however of simple tests and next-hop lookups only, and should be easy to implement.

The routers will need a mapping between each interface and a specific backup configuration. This mapping can be built when the configurations are created.

V. Performance Evaluation

MRC is a local, proactive recovery scheme that resumes packet forwarding immediately after the failure is detected, and hence provides fast recovery. State requirements and the influence on network traffic are other important metrics, which will be evaluated in this section.

MRC requires the routers to store additional routing configurations. The amount of state required in the routers, is related to the number of such backup configurations. Since routing in a backup configuration is restricted, MRC will potentially give backup paths that are longer than the optimal paths. Longer backup paths will affect the total network load and also the end-to-end delay. We use a routing simulator to evaluate these metrics on a wide range of synthetic topologies. We also use a packet simulator to study the effect of failures on the network traffic in one selected topology.

Shortest path routing or “OSPF normal” in the full topology is chosen as a benchmark for comparison throughout the evaluation. The new routing resulting from full OSPF re-convergence after a single component failure is denoted “OSPF rerouting”. It must be noted that MRC yields the shown performance immediately after a failure, while IP re-convergence can take seconds to complete. Our goal is to see how close MRC can approach the performance of global OSPF re-convergence.

A. Method

1) Routing simulation: We have developed a Java software model that is used to create configurations as described by the algorithm in Sec. III-B. The configurations are created for a wide range of topologies, obtained from the BRITE topology generation tool [17] using the Waxman [18] and the Generalized Linear Preference (GLP) [19] models. The number of nodes is varied between 16 and 512 to demonstrate the scalability. To explore the effect of network density, the average node degree is 4 or 6 for Waxman topologies and 3.6 for GLP topologies. For all synthetic topologies, the links are given unit weight.

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

2) Traffic simulation: To test the effects our scheme has on the load distribution after a failure, we have implemented our scheme in a discrete-event packet simulator based on the J-Sim framework [20][1]. Simulations are performed on the European COST239 network [21] shown in Fig. 4, connecting major cities across Europe. All links have been given a common base weight (dominant), plus an individual addition based on their propagation delay.

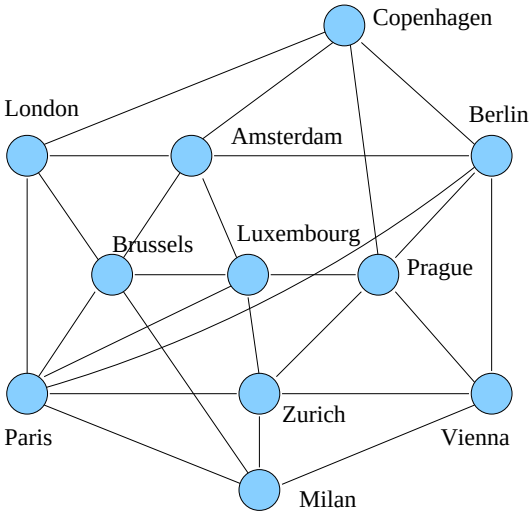

Fig. 4. The COST239 network

For our experiments, we use a traffic matrix where the traffic between two destinations is based on the population of the countries they represent [21]. For simplicity, we look at constant packet streams between each node pair. Since the purpose of the simulations is to measure how the traffic load is distributed in the network, the link capacity is set so that we never experience packet loss due to congestion. For all simulations, three backup configurations were used with MRC. We evaluate link loads before the failure, and after recovery using OSPF or MRC.

B. Routing Results

1) Minimum number of backup configurations: Figure 5 shows the minimum number of backup configurations that are needed to make all links and nodes isolated in a wide range of synthetic topologies. Each bar in the figure represents 100 different topologies given by the type of generation model used, the links-to-node ratio, and the number of nodes in the topology.

Tab. II shows the minimum number of required backup configurations needed for five real world topologies.

The results show that the number of backup configurations needed is usually modest; 3 or 4 is typically enough to isolate every element in a topology. The number of configurations needed decreases with increasing network connectivity. We never needed more than six configurations in our experiments. This modest number of backup configurations shows that our method is implementable without requiring a significant amount of state information.

[1]Our J-Sim extensions, together with our routing simulation software, is available at http://www.simula.no

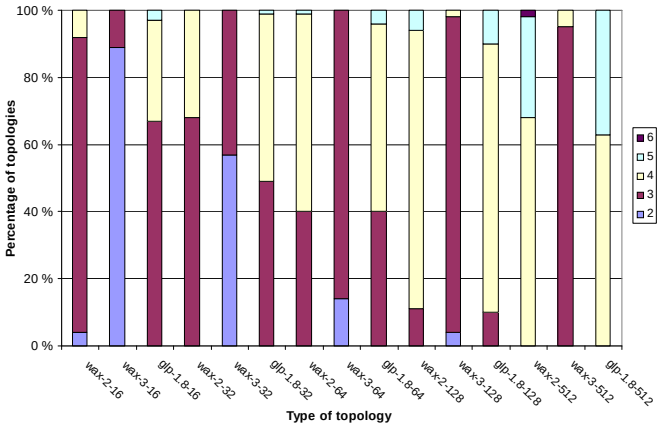

Fig. 5. The number of backup configurations required for a wide range of Brite generated topologies. As an example the bar name wax-2-16 denotes that the Waxman model is used with a links-to-node ratio of 2, and with 16 nodes.

TABLE II
NUMBER OF BACKUP CONFIGURATIONS FOR SELECTED REAL WORLD NETWORKS

Network	Nodes	Links	Configurations
Sprint US	32	64	4
German Tel	10	29	3
DFN	13	64	2
Geant	19	30	5
Cost239	11	26	3

2) Backup path lengths: Fig. 6 shows path length distribution for node failures. The numbers are based on 100 different topologies with 32 nodes and 64 links. Results for link failures and other network properties show the same tendency.

For reference, we show the path length distribution in the failure-free case (“OSPF normal”), for all paths with at least two hops. For an original path we let every intermediate node fail, and calculate the resulting backup path lengths using global OSPF rerouting, local rerouting based on the full topology except the failed component (“Optimal local”), as well as MRC with 3 and 7 backup configurations. We then select the median from these samples, and repeat for all paths in the network.

We see that MRC gives backup path lengths close to those achieved after a full OSPF re-convergence, and that the difference decreases further if we allow the use of more configurations. This means that the affected traffic will not suffer from unacceptably long backup paths in the period when it is forwarded according to an MRC backup configuration.

C. Traffic Results

1) Total network load: This metric is related to the backup path length and represents the total traffic load in the network

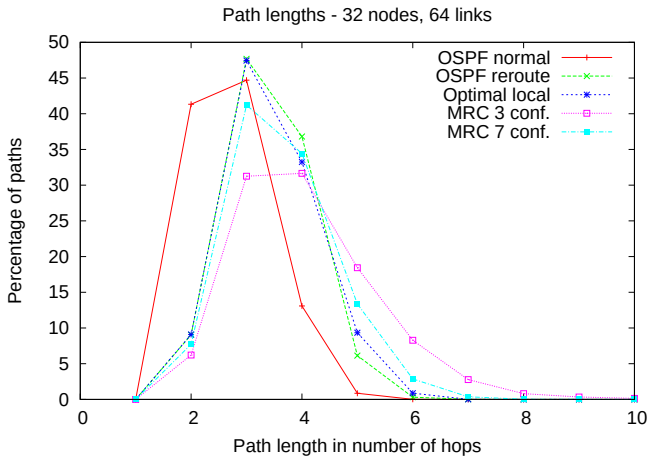

Fig. 6. Backup path lengths in the case of a node failure.

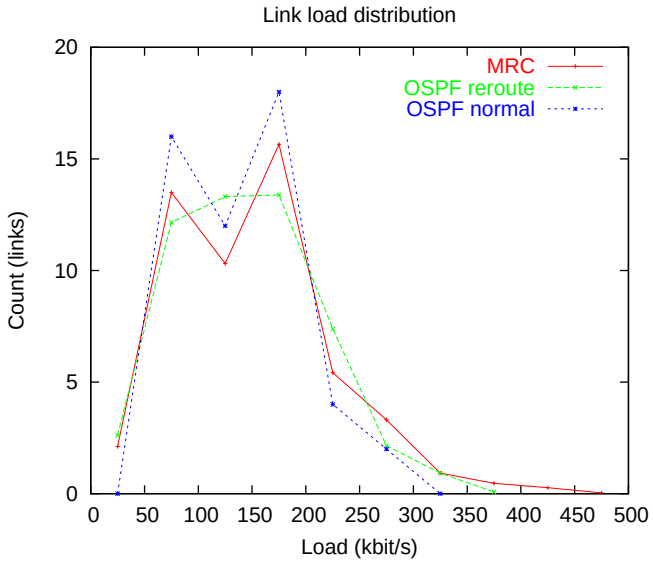

Fig. 8. Distribution of link loads in the network in the normal case, using MRC, and after OSPF rerouting.

after a failure. The sub-optimal backup paths given by MRC should result in an increased load in the network. Fig. 7 shows the aggregate throughput of all the links in the COST239 network after a link failure. The link index on the x-axis shows which of the 26 bidirectional links has failed. The relative increase in the load compared to the failure-free case is given on the y-axis.

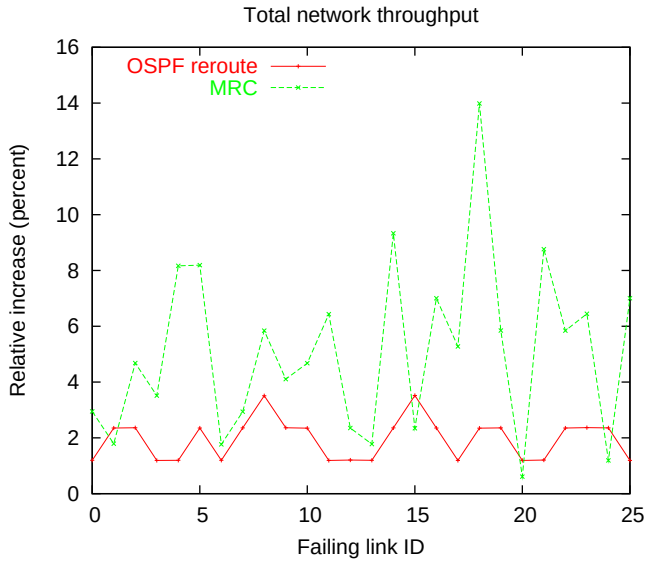

Fig. 7. Network load after link failure.

The simulations show that the load in the network increases about 5% on average after a failure when using MRC with 3 backup configurations, compared to a 2% increase with OSPF rerouting. All traffic is recovered in this scenario, so the increased network load is solely caused by the longer paths experienced by the rerouted traffic.

2) Link load distribution: Fig. 8 shows the link load distribution in the COST239 network. Again, results for the failure free situation and for OSPF rerouting are given. Results for OSPF rerouting and MRC using 3 backup configurations are averages for all 26 possible link failures.

The simulations suggest that the link load distribution in the network is similar when using MRC and after complete OSPF re-convergence.

3) Load on individual links: Fig. 9 shows the load on every unidirectional link in the network in the failure-free case, and after a link failure. The links are indexed from the least loaded to the most loaded in the failure-free case. Results are shown for MRC, and after the OSPF rerouting process has terminated.

We measure the throughput on each link for every possible link failure. Fig. 9a shows the average for all link failures, while Fig. 9b shows the worst case for each individual link. The results show that both for the average and the worst case, MRC gives a post-failure load on each link comparable to the one achieved after a full OSPF re-convergence.

In our simulations, we have kept the link weights from the original full topology in the backbone part of the backup topologies. However, we believe there is a great potential for improved load balancing after a failure by optimizing the link weights in the backup topologies.

VI. Related Work

Much work has lately been done to improve robustness against component failures in IP networks [22]. In this section, we focus on some important contributions aimed at restoring connectivity without a global re-convergence. Tab. III summarizes important features of the different approaches. We indicate whether each mechanism guarantees one-fault tolerance in an arbitrary biconnected network, for both link and node failures, independent of the root cause of failure (failure agnostic). We also indicate whether the approaches

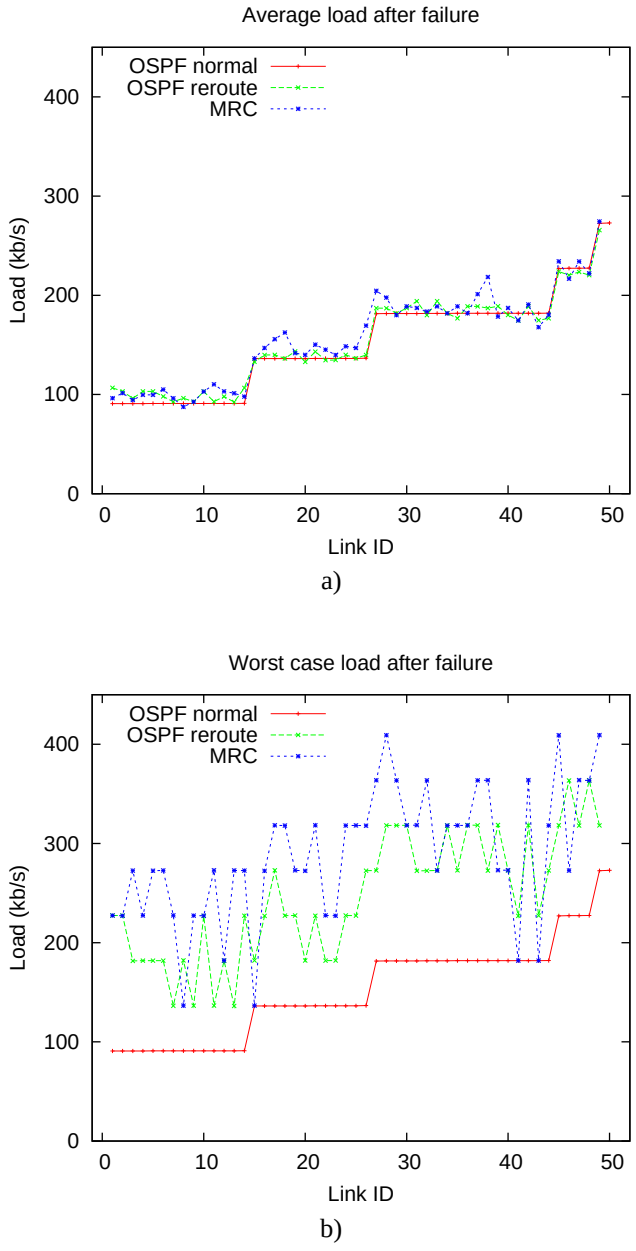

Fig. 9. Load on all unidirectional links before and after failure. a) Average for all link failures. b) Each individual link's worst case scenario.

guarantees shortest path routing in the failure-free case, and whether they solve the ”last hop problem”.

IETF has recently drafted a framework called IP fast reroute [23]. Within this framework, they propose the use of a tunnelling approach based on so called “Not-via” addresses to handle link and node failures [16]. To protect against the failure of a component P, a special Not-via address is created for this component at each of P's neighbors. Forwarding tables are then calculated for these addresses without using the protected component. This way, all nodes get a path to each of P's neighbors, without passing through (“Not-via”) P. The Not-via approach is similar to MRC in that loop free backup next-hops are found by doing shortest path calculations on a subset of the network. It also covers against link and node failures using the same mechanism, and is strictly pre-configured. However, the tunnelling approach may give less optimal backup paths, and less flexibility with regards to post failure load balancing.

Iselt et al. [24] emulate Equal Cost Multi-Path (ECMP) by using MPLS to set up virtual links where needed to make equal cost paths to a destination. This makes it possible to use one ECMP path as backup when another fails. Their method uses separate mechanisms to protect against link and node failures. Their scheme is strictly pre-configured, and it is not fully connectionless as it introduces connection-oriented emulation. As a consequence of the ECMP emulation, one hop as viewed from the routing function would often correspond to several original hops, and hence this scheme can not guarantee shortest path failure-free routing. Their scheme is not failure agnostic, i.e., they specify separate methods for link and node failures, and therefore the ”last hop problem” is avoided.

Narvaez et al. [25] propose a method relying on multi-hop repair paths. They propose to do a local re-convergence upon detection of a failure, i.e., notify and send updates only to the nodes necessary to avoid loops. A similar approach also considering dynamic traffic engineering is presented in [26]. We call these approaches *local rerouting*. They are designed only for link failures, and therefore avoid the problems of root cause of failure and the last hop. Their method does not guarantee one-fault-tolerance in arbitrary biconnected networks. It is obviously connectionless. However, it is not strictly pre-configured, and can hence not recover traffic in the same short time-scale as a strictly pre-configured scheme.

Reichert et al. [27] propose a routing scheme named O2, where all routers have at least two valid loop-free next hops to any destination. To obtain two valid next hops, the biconnected network topology must fulfil certain requirements and the normal failure-free routes may not be the shortest. Their scheme is strictly pre-configured and connectionless. It covers both node and link failures independent of the root cause of failure, and it also solves the ”last hop problem”.

Lee et al. [8] propose using interface specific forwarding to provide loop-free backup next hops to recover from link failures. Their approach is called failure insensitive routing (FIR). The idea behind FIR is to let a router infer link failures based on the interface packets are coming from. When a link fails, the attached nodes locally reroute packets to the affected destinations, while all other nodes forward packets according to their pre-computed interface specific forwarding tables without being explicitly aware of the failure. Later, they have also proposed a similar method, named failure inferencing based fast rerouting (FIFR), for handling node failures [28]. This method will also cover link failures, and hence it operates independent of the root cause of failure. However, their method will not guarantee this for the last hop, i.e. they do not solve the ”last hop problem”. Regarding other properties, FIFR guarantees one-fault-tolerance in any

biconnected network, it is connectionless, pre-configured and it does not affect the original failure-free routing.

Many of the approaches listed provide elegant and efficient solutions to fast network recovery, however MRC and Not-via tunneling seems to be the only two covering all evaluated requirements. However, we argue that MRC offers the same functionality with a simpler and more intuitive approach, and leaves more room for optimization with respect to load balancing.

VII. Conclusion and future work

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

MRC operates without knowing the root cause of failure, i.e., whether the forwarding disruption is caused by a node or link failure. 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 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. In the selected COST239 network, this added path length gives a network load that is marginally higher than the load with optimal backup paths. MRC thus achieves fast recovery with a very limited performance penalty.

There are several possible directions for future work. The use of MRC gives a changed traffic pattern in the network after a failure. We believe that the risk of congestion after a failure can be reduced by doing traffic engineering through intelligent link weight assignment in each configuration.

Since MRC can isolate several nodes and links in a single configuration, it can be successfully used in case of multiple component failures. For instance, MRC might be well suited for Shared Risk Groups by making sure that all elements in such a group are isolated in the same configuration.

Keeping backup configurations in the network makes protection of multicast traffic much easier. Protecting multicast traffic from node failures is a challenging task, since state information for a whole multicast subtree is lost. By maintaining a separate multicast tree in each backup configuration, we believe that very fast recovery from both link and node failures can be achieved.

Acknowledgments

The authors are grateful to Prof. David Hutchison of Lancaster University and Prof. Constantine Dovrolis of Georgia Tech and the anonymous reviewers for their valuable input to this work.

References

[1] D. D. Clark, "The Design Philosophy of the DARPA Internet Protocols," *SIGCOMM, Computer Communications Review*, vol. 18, no. 4, pp. 106–114, Aug. 1988.

[2] A. Basu and J. G. Riecke, "Stability Issues in OSPF Routing," in *Proceedings of SIGCOMM 2001*, August 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.

[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*. IEEE Computer Society, 2003, pp. 204–213.

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

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

[8] S. Lee, Y. Yu, S. Nelakuditi, Z.-L. Zhang, and C.-N. Chuah, "Proactive vs. reactive approaches to failure resilient routing," in *Proceedings IEEE INFOCOM'04*, Mar. 2004.

[9] S. Iyer, S. Bhattacharyya, N. Taft, and C. Diot, "An approach to alleviate link overload as observed on an IP backbone," in *Proceedings of INFOCOM'03*, Mar. 2003, pp. 406–416.

[10] P. Psenak, S. Mirtorabi, A. Roy, L. Nguen, and P. Pillay-Esnault, "MT-OSPF: Multi topology (MT) routing in OSPF," IETF Internet Draft (work in progress), Apr. 2005, draft-ietf-ospf-mt-04.txt.

[11] T. Przygienda, N. Shen, and N. Sheth, "M-ISIS: Multi topology (MT) routing in IS-IS," Internet Draft (work in progress), May 2005, draft-ietf-isis-wg-multi-topology-10.txt.

[12] M. Menth and R. Martin, "Network resilience through multi-topology routing," University of Wurzburg, Institute of Computer Science, Tech. Rep. 335, May 2004.

[13] I. Theiss and O. Lysne, "FROOTS - fault handling in up*/down* routed networks with multiple roots," in *Proceedings of the International Conference on High Performance Computing (HiPC)*, 2003.

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

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

[16] S. Bryant, M. Shand, and S. Previdi, "IP fast reroute using not-via addresses," Internet Draft (work in progress), Oct. 2005, draft-bryant-shand-IPFRR-notvia-addresses-01.txt.

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

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

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

[20] H.-Y. Tyan, "Design, realization and evaluation of a component-based compositional software architecture for network simulation," Ph.D. dissertation, Ohio State University, 2002.

TABLE III

CONCEPTUAL COMPARISON OF DIFFERENT APPROACHES FOR FAST IP RECOVERY

Scheme	**Guaranteed in biconnected**	**Node faults**	**Link faults**	**Pre-configured**	**Connec-tionless**	**Shortest path normal**	**Failure agnostic**	**Last hop**
MRC	yes	yes	yes	yes	yes	yes	yes	yes
Not-via tunnelling [16]	yes	yes	yes	yes	yes	yes	yes	yes
ECMP-MPLS [24]	yes	yes	yes	yes	no	no	no	N/A
Local rerouting [25]	no	no	yes	no	yes	yes	N/A	N/A
O2 [27]	no	yes	yes	yes	yes	no	yes	yes
FIR [8]	yes	no	yes	yes	yes	yes	N/A	N/A
FIFR [28]	yes	yes	yes	yes	yes	yes	yes	no
Rerouting (OSPF)	yes	yes	yes	no	yes	yes	yes	yes

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

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

[23] M. Shand, "IP Fast Reroute Framework," IETF Internet Draft (work in progress), June 2005, draft-ietf-rtgwg-ipfrr-framework-03.txt.

[24] A. Iselt, A. Kirstdter, A. Pardigon, and T. Schwabe, "Resilient routing using ECMP and MPLS," in *Proceedings of HPSR 2004*, Apr. 2004.

[25] P. Narvaez, K.-Y. Siu, and H.-Y. Tzeng, "Local restoration algorithms for link-state routing protocols," in *Proceedings of IEEE International Conference on Computer Communications and Networks*, Oct. 1999.

[26] R. Rabbat and K.-Y. Siu, "Restoration methods for traffic engineered networks for loop-free routing guarantees," in *Proceedings of ICC*, June 2001.

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

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