Multiple Routing Configurations for Fast IP Network Recovery

Amund Kvalbein, *Member, IEEE*, Audun Fosselie Hansen, Tarik Čičić, *Member, IEEE*, Stein Gjessing, *Member, IEEE*, and Olav Lysne, *Member, IEEE*

***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). 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. 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. 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. We also show how an estimate of the traffic demands in the network can be used to improve the distribution of the recovered traffic, and thus reduce the chances of congestion when MRC is used.**

***Index Terms*—Availability, computer network reliability, communication system fault tolerance, communication system routing, protection.**

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.

Manuscript received December 21, 2006; revised July 20, 2007 and February 06, 2008; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor J. Yates. First published July 25, 2008; current version published April 15, 2009.

A. Kvalbein and A. F. Hansen are with Simula Research Laboratory, 1367 Lysaker, Norway (e-mail: amundk@simula.no).

T. Čičić was with Simula Research Laboratory, 1367 Lysaker, Norway. He is now with Media Network Services, Norway, and also with the Department of Informatics, University of Oslo, 1371 Oslo, Norway.

S. Gjessing and O. Lysne are with Simula Research Laboratory, 1367 Lysaker, Norway, and also with the University of Oslo, 0371 Oslo, Norway.

Digital Object Identifier 10.1109/TNET.2008.926507

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 [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 a *proactive* and *local* protection mechanism that allows recovery in the range of milliseconds. MRC allows packet forwarding to continue over preconfigured 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]. 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.

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 on an alternate link. MRC assumes that the network uses shortest path routing and destination based hop-by-hop forwarding.

The shifting of traffic to links bypassing the failure can lead to congestion and packet loss in parts of the network [9]. This limits the time that the proactive recovery scheme can be used to forward traffic before the global routing protocol is informed about the failure, and hence reduces the chance that a transient

failure can be handled without a full global routing re-convergence. Ideally, a proactive recovery scheme should not only guarantee connectivity after a failure, but also do so in a manner that does not cause an unacceptable load distribution. This requirement has been noted as being one of the principal challenges for precalculated IP recovery schemes [10]. With MRC, the link weights are set individually in each backup configuration. This gives great flexibility with respect to how the recovered traffic is routed. The backup configuration used after a failure is selected based on the failure instance, and thus we can choose link weights in the backup configurations that are well suited for only a subset of failure instances.

The rest of this paper is organized as follows. In Section II we describe the basic concepts and functionality of MRC. We then define MRC formally and present an algorithm used to create the needed backup configurations in Section III. In Section IV, we explain how the generated configurations can be used to forward the traffic safely to its destination in case of a failure. We present performance evaluations of the proposed method in Section V. In Section VI, we discuss how we can improve the recovery traffic distribution if we have an estimate of the demands in the network. In Section VII, we discuss related work, and finally we conclude in Section VIII.

II. MRC Overview

MRC is based on building a small set of backup routing configurations, that are used to route recovered traffic on alternate paths after a failure. The backup configurations differ from the normal routing configuration in that link weights are set so as to avoid routing traffic in certain parts of the network. We observe that if all links attached to a node are given sufficiently high link weights, traffic will never be routed through that node. The failure of that node will then only affect traffic that is sourced at or destined for the node itself. Similarly, to exclude a link (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 guarantees loop-free forwarding within one configuration. Finally, we design a forwarding process that takes advantage of the backup configurations to provide fast recovery from a component failure.

In our approach, we construct the backup configurations so that for all links and nodes in the network, there is a configuration where that link or node is not used to forward traffic. Thus, for any single link or node failure, there will exist a configuration that will route the traffic to its destination on a path that avoids the failed element. Also, the backup configurations must be constructed so that all nodes are reachable in all configurations, i.e., there is a valid path with a finite cost between each node pair. Shared Risk Groups can also be protected, by regarding such a group as a single component that must be avoided in a particular configuration. In Section III, we formally describe MRC and how to generate configurations that protect every link and node in a network.

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

When a router detects that a neighbor can no longer be reached through one of its interfaces, it does not immediately inform the rest of the network about the connectivity failure. Instead, packets that would normally be forwarded over the failed interface are marked as belonging to a backup configuration, and forwarded on an alternative interface towards its destination. The selection of the correct backup configuration, and thus also the backup next-hop, is detailed in Section 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 specific values in the DSCP field in the IP header. If this is not possible, other packet marking strategies like IPv6 extension headers or using a private address space and tunneling (as proposed in [11]) could be used.

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

If a failure lasts for more than a specified time interval, a normal re-convergence will be triggered. MRC does not interfere with this convergence process, or make it longer than normal. However, MRC gives continuous packet forwarding during the convergence, and hence makes it easier to use mechanisms that prevents *micro-loops* during convergence, at the cost of longer convergence times [12]. If a failure is deemed permanent, new configurations must be generated based on the altered topology.

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 start-up of the network, and each time a node or link is permanently added or removed. We use the notation shown in Table I.

A. Configurations Structure

MRC configurations are defined by the network topology, which is the same in all configurations, and the associated link weights, which differ among configurations. We formally represent the network topology as a graph $G = (N, A)$, with a set of

TABLE I
NOTATION

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

nodes N and a set of unidirectional links (arcs) A.[1] 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:

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

We distinguish between the normal configuration C_0 and the backup configurations C_i, $i > 0$. In the normal configuration, C_0, all links have "normal" weights $w_0(a) \in \{1, \ldots, w_{\max}\}$. We assume that C_0 is given with finite integer weights. MRC is agnostic to the setting of these weights. In the backup configurations, selected links and nodes must not carry any transit traffic. Still, traffic must be able to depart from and reach all operative nodes. These traffic regulations are imposed by assigning high weights to some links in the backup configurations:

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

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

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

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

$$\begin{aligned} &\forall (u, v) \in A, w_i(u, v) \geq w_\mathrm{r} \\ \wedge \quad &\exists (u, v) \in A, w_i(u, v) = w_\mathrm{r}. \end{aligned} \tag{1}$$

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

$$w_i(u, v) = w_\mathrm{r} \Rightarrow (u \in S_i \wedge v \in \overline{S}_i) \vee (v \in S_i \wedge u \in \overline{S}_i) \tag{2}$$

$$w_i(u, v) = \infty \Rightarrow u \in S_i \vee v \in S_i. \tag{3}$$

This means that a restricted link always connects an isolated node to a non-isolated node. An isolated link either connects an isolated node to a non-isolated node, or it connects two isolated nodes. Importantly, this means that a link is always isolated in the same configuration as at least one of its attached nodes. These two rules are required by the MRC forwarding process described in Section IV in order to give correct forwarding without knowing the root cause of failure. When we talk of a backup configuration, we refer to a configuration that adheres to (2) and (3).

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

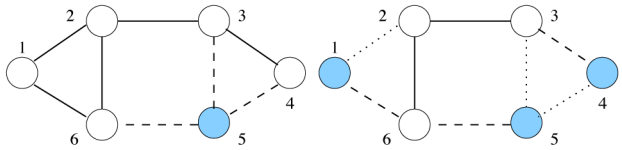


Fig. 1. Left: node 5 is isolated (shaded color) by setting a high weight on all its connected links (stapled). Only traffic to and from the isolated node will use these restricted links. Right: a configuration where nodes 1, 4 and 5, and the links 1–2, 3–5 and 4–5 are isolated (dotted).

The purpose of the restricted links is to isolate a node from routing in a specific backup configuration C_i, such as node 5 to the left in Fig. 1. In many topologies, more than a single node can be isolated simultaneously. In the example to the right in Fig. 1, three nodes and three links are isolated.

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

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

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

$$\begin{aligned} \forall u, v \in N : \mathcal{N}\left(p_i(u, v)\right) \setminus \left(\overline{S}_i \cup \{u, v\}\right) = \emptyset \\ \wedge w_i\left(p_i(u, v)\right) < \infty. \end{aligned} \tag{4}$$

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

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

$$a \in A_i \Leftrightarrow w_i(a) \leq w_{\max}. \tag{5}$$

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

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

$$\forall u, v \in B_i : a \in \mathcal{A}\left(p_i(u, v)\right) \Rightarrow w_i(a) \leq w_{\max} \tag{6}$$

An important invariant in our algorithm for creating backup configurations is that the backbone remains connected. Since all backup configurations must adhere to (2) and (3), we can

show that a backup configuration with a connected backbone is equivalent to a valid backup configuration:

Lemma 3.1: A backup configuration C_i is valid if and only if it contains a connected backbone.

Proof: We first show that a connected backbone implies that C_i is valid. For each node pair u and v, zero, one or both of u and v are in S_i. Assume $u \in S_i \wedge v \in S_i$. From the definition of an isolated node and (2), $\exists u', v' \in \overline{S}_i : w_i(u,u') = w_{\mathrm{r}} \wedge w_i(v,v') = w_{\mathrm{r}}$. From (6) $a \in \mathcal{A}(p_i(u',v')) \Rightarrow w(a) \leq w_{\max}$. Thus,

$$w_i\left(p_i(u,v)\right) \leq 2w_{\mathrm{r}} + w_i\left(p_i(u',v')\right) < \infty \tag{7}$$

$$\mathcal{N}\left(p_i(u,v)\right) \setminus \left(\overline{S}_i \cup \{u,v\}\right) = \emptyset \tag{8}$$

and (4) follows. A subset of the above is sufficient to show the same if only one, or none, of u, v is in S_i.

For the converse implication, assume $u, v \in \overline{S}_i$ and node $x \in \mathcal{N}(p_i(u,v))$. From (4), $x \in \overline{S}_i$ and $w_i(p_i(u,v)) < \infty$. Since by (2) restricted links are always connected to at least one isolated node, such links can not be part of $\mathcal{A}(p_i(u,v))$, and all links in $\mathcal{A}(p_i(u,v))$ must have normal weights. ∎

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

Proposition 3.2: Let x be a node isolated in the valid backup configuration C_i. Then, restricted link weight value

$$w_{\mathrm{r}} = |A| \cdot w_{\max} \tag{9}$$

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

Proof: Since all links attached to the isolated node x have a weight of at least w_{r}, the weight of a path through x will be at least $2 \cdot w_{\mathrm{r}} = 2 \cdot |A| \cdot w_{\max}$. From the definition of an isolated node and (2), all isolated nodes are directly connected to the configuration backbone. From (4), any shortest path in C_i will be entirely contained in B_i, except possibly the first or the last hop. A valid configuration contains a connected backbone, and the total weight of the sub-path that is within B_i will be at most $|A_i| \cdot w_{\max}$. Since $|A_i| < 2|A|$, no shortest path will include x as the transit. ∎

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

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

$$\begin{aligned} &\forall a \in A, \exists C_i \in \mathcal{C} : w_i(a) = \infty \\ \wedge \quad &\forall u \in N, \exists C_i \in \mathcal{C} : u \in S_i. \end{aligned} \tag{10}$$

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

B. Algorithm

The number and internal structure of backup configurations in a complete set for a given topology may vary depending on the construction model. If more configurations are created, fewer links and nodes need to be isolated per configuration, giving a richer (more connected) backbone in each configuration. On the other hand, if fewer configurations are constructed, the state requirement for the backup routing information storage is reduced. However, calculating the minimum number of configurations for a given topology graph is computationally demanding. One solution would be to find all valid configurations for the input consisting of the topology graph G and its associated normal link weights w_0, and then find the complete set of configurations with lowest cardinality. Finding this set would involve solving the Set Cover problem, which is known to be NP-complete [13].

Algorithm 1: Creating backup configurations.

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

Instead 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 w_0, 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 Section III-B-3. 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 1 loops through all nodes in the topology, and tries to isolate them one at a time. A link is isolated in the same iteration as one of its attached nodes. The algorithm terminates when either all nodes and links in the network are isolated in exactly one configuration, or a node that cannot be

Function `isolate`(C_i, u)

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

isolated is encountered. We now specify the algorithm in detail, using the notation shown in Table I.

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

When a node u is attempted isolated in a backup configuration C_i, it is first tested that doing so will not disconnect B_i according to definition (6). The `connected` method at line 13 decides this by testing that each of u's neighbors can reach each other without passing through u, an isolated node, or an isolated link in configuration C_i.

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

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

b) Isolating links: Along with u, as many as possible of its attached links are isolated. The algorithm runs through the links $A(u)$ attached to u (lines 2–3 in function `isolate`). It can be shown that it is an invariant in our algorithm that in line 1, all links in Q_{a} are attached to node u. The node v in the other end of the link may or may not be isolated in some configuration already (line 4). If it is, we must decide whether the link should be isolated along with u (line 7), or if it is already isolated in the configuration where v is isolated (line 11). A link must always be isolated in the same configuration as one of its end nodes. Hence, if the link was not isolated in the same configuration as v, it *must* be isolated along with node u.

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

In the case that the neighbor node v was *not* isolated in any configuration (line 12), we isolate the link along with u if there exists another link not isolated with u (line 14). If the link can not be isolated together with node u, we leave it for node v to isolate it later. To make sure that this link can be isolated along with v, we must process v next (line 17, selected at line 10 in Algorithm 1), and link (v, u) must be the first among the links originating from node v to be processed (line 18, selected at line 2).

2) Output: We show that successful execution of Algorithm 1 results in a complete set of valid backup configurations.

Proposition 3.3: If Algorithm 1 terminates successfully, the produced backup configurations adhere to (2) and (3).

Proof: Links are only given weights w_{r} or ∞ in the process of isolating one of its attached nodes, and (3) follows. For restricted links, (2) requires that only one of the attached nodes are isolated. This invariant is maintained in line 7 in function `isolate` by demanding that if a node attached to a restricted link is attempted isolated, the link must also be isolated. Hence it is impossible to isolate two neighbor nodes without also isolating their connecting link, and (2) follows. ∎

Proposition 3.4: If Algorithm 1 terminates successfully, the backup configurations set $\mathcal{C} = \{C_1, C_2, \ldots, C_n\}$ is complete, and all configurations $C_i \in \mathcal{C}$ are valid.

Proof: Initially, all links in all configurations have original link weights. Each time a new node and its connected links are isolated in a configuration C_i we verify that the backbone in that configuration remains connected. When the links are isolated, it is checked that the node has at least one neighbor not isolated in C_i (line 14 in Function `isolate`). When isolating a node, we also isolate as many as possible of the connected links. A link is always isolated in the same configuration as one of its attached nodes. If this is not possible, the node is not isolated (`isolate`, line 9). From Lemma 3.1, the altered configuration remains valid.

The algorithm runs through all nodes. If one node cannot be isolated, the algorithm aborts (line 22 in Algorithm 1). If it does terminate with success, all nodes and links are isolated in one configuration, thus the configuration set is complete. ∎

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

Proposition 3.5: Given a bi-connected graph $G = (N, A)$, there will exist $n \leq |N|$, so that Algorithm 1 will terminate successfully.

Proof: Assume $n = |N|$. The algorithm will create $|N|$ backup configurations, isolating one node in each backup configuration. In bi-connected topologies this can always be done.

Along with a node u, all attached links except one, say (u, v), can be isolated. By forcing node v to be the next node processed (`isolate` line 17), and the link (v, u) to be first among $A(v)$ (line 18), node v and link (v, u) will be isolated in the next configuration. This can be repeated until we have configurations so that every node and link is isolated. This holds also for the last node processed, since its last link will always lead to a node that is already isolated in another configuration. Since all links and nodes can be isolated, the algorithm will terminate successfully. ■

Ring topologies represent the worst-case input for our algorithm since all $|N|$ nodes have two links each and would have to be isolated in different configurations in order to close the loop described in Prop. 3.5. In bi-connected networks with higher connectivity it is often possible to reuse the configurations and terminate the algorithm with a lower n. In Section V we analyze the number of backup configurations created by Algorithm 1 for different input network topologies.

4) Complexity: The complexity of the proposed algorithm is determined by the loops and the complexity of the `connected` method. This method performs a procedure similar to determining whether a node is an articulation point in a graph, bound to worst case $\mathcal{O}(|N|+|A|)$. Additionally, for each node, we run through all adjacent links, whose number has an upper bound in the maximum node degree Δ. In the worst case, we must run through all n configurations to find a configuration where a node can be isolated. The worst case running time for the complete algorithm is then bound by $\mathcal{O}(n\Delta|N||A|)$. The running time on a standard desktop computer varies from sub-second for small topologies to a few seconds for topologies with several hundred nodes.

IV. Local Forwarding Process

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

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

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

Let $C(u)$ denote the backup configuration where node u is isolated, i.e., $C(u) = C_i \Leftrightarrow u \in S_i$. Similarly, let $C(u, v)$ denote the backup configuration where the link (u, v) is isolated, i.e., $C(u, v) = C_i \Leftrightarrow w_i(u, v) = \infty$. Assuming that node d is the egress (or the destination) in the local network domain,

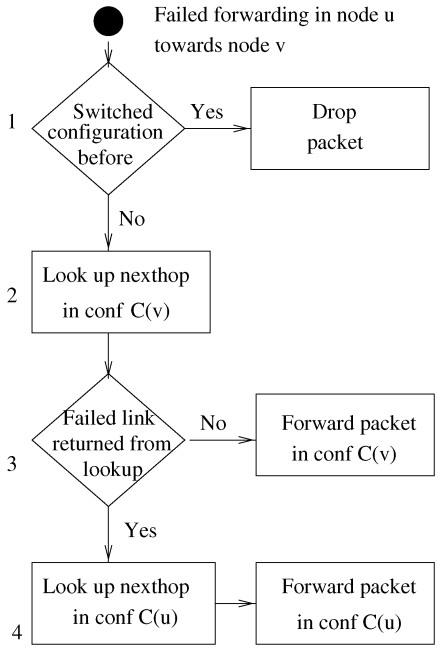


Fig. 2. Packet forwarding state diagram.

we can distinguish between two cases. If $v \neq d$, 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 C_0), and still meet a failed component on their forwarding path, must be discarded. This way packets loops are avoided, also in the case that node d indeed has failed. The steps that are taken in the forwarding process by the detecting node u are summarized in Fig. 2.

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

Proposition 4.1: Node u selects configuration C_i so that $v \notin \mathcal{N}(p_i(u, d))$, if $v \neq d$.

Proof: Node u selects $C(v)$ in step 2. Node v is isolated in $C(v)$ and will not be in the shortest path $p_i(u, d)$ according to proposition 3.2. ■

Proposition 4.2: Node u selects configuration C_i so that $(u, v) \notin \mathcal{A}(p_i(u, d))$.

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

Assume that v is the egress node for destination d. Remember that according to (3), $C(u, v) = C(u) \vee C(u, v) = C(v)$. We distinguish between three possible cases, illustrated in Fig. 3.

If $C(u) = C_i$ and $C(v) = C_i$ as in Fig. 3(a), then $C(u, v) = C_i$ according to the definition of an isolated node and (2). Forwarding step 2 will select $C(v) = C_i$ and $\mathcal{A}(p_i(u, v))$ does not contain (u, v).

If $C(u) = C_i$, $C(v) = C_j$, $i \neq j$, and $C(u, v) = C_j$ as in Fig. 3(b), forwarding step 2 will select $C(v) = C_j$ and $\mathcal{A}(p_j(u, v))$ does not contain (u, v).

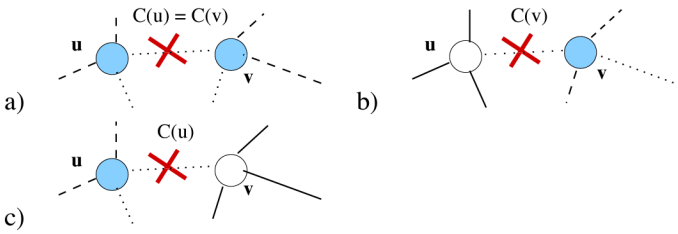


Fig. 3. When there is an error in the last hop $u \rightarrow v$, a packet must be forwarded in the configuration where the connecting link is isolated. The figure shows isolated nodes (shaded color), restricted links (dashed), and isolated links (dotted). In cases (a) and (b), $C(u,v) = C(v)$, and the forwarding will be done in $C(v)$. In case (c), $C(u,v) \neq C(v)$, and the forwarding will be done in $C(u)$.

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

A. Implementation Issues

The forwarding process can be implemented in the routing equipment as presented above, requiring the detecting node u to know the backup configuration $C(v)$ for each of its neighbors. Node u would then perform at most two additional next-hop 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 the DSCP field of the IP header to identify the configuration. As we will see shortly, a very limited number of backup configurations are needed to guarantee recovery from all single link or node failures, and hence the number of needed values would be small. A more implicit approach would be to assign a distinct local IP address space for each backup configuration. Each node in the IGP cloud would get a separate address in each configuration. The detecting node could then encapsulate recovered packets and tunnel them shortest path in the selected backup configuration to the egress node. The packets would then be decapsulated at the egress and forwarded from there as normal towards the final destination. The drawback with this method is the additional processing and bandwidth resource usage associated with tunneling.

Recent IETF standardization work on Multi Topology routing mechanisms [14], [15] provides a useful framework for MRC implementation. These IETF drafts specify how routers can exchange information about the link weights used in several logical topologies, and build topology specific forwarding tables. Use of these drafts for providing proactive recovery is sketched in [16].

V. Performance Evaluation

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

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

A. Evaluation Setup

We have implemented the algorithm described in Section III-B and created configurations for a wide range of bi-connected synthetic and real topologies.[2] The synthetic topologies are 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. The real topologies are taken from the Rocketfuel topology database [20].

For each topology, we measure the minimum number of backup configurations needed by our algorithm to isolate every node and link in the network. Recall from Section III-B that our algorithm for creating backup configurations only takes the network topology as input, and is not influenced by the link weights. Hence, the number of configurations needed is valid irrespective of the link weight settings used. For the Rocketfuel topologies, we also measure the number of configurations needed if we exclude the nodes that can be covered by *Loop-Free Alternates* (LFA) [21]. LFA is a cheaper fast reroute technique that exploits the fact that for many destinations, there exists an alternate next-hop that will not lead to a forwarding loop. If such alternate paths exist for *all* traffic that is routed through a node, we can rely on LFA instead of protecting the node using MRC.

Based on the created configurations, we measure the backup path lengths (hop count) achieved by our scheme after a node failure. For a selected class of topologies, we evaluate how the backup path lengths depend on the number of backup configurations.

The shifting of traffic from the normal path to a recovery path changes the load distribution in the network, and can in some cases lead to congestion and packet loss. We therefore test the effect our scheme has on the load distribution after a failure. To do this, we have performed simulations of the European COST239 network [22] shown in Fig. 4, designed to connect major cities across Europe. All links in the network have

[2]Our simulation software is available at http://www.simula.no/research/networks/software/.

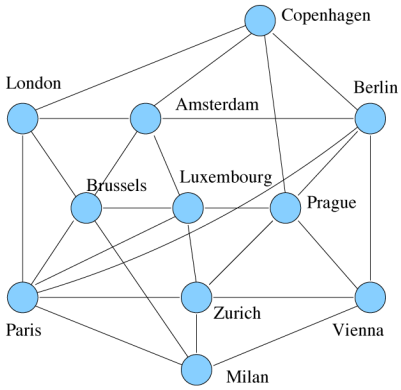


Fig. 4. The COST239 network.

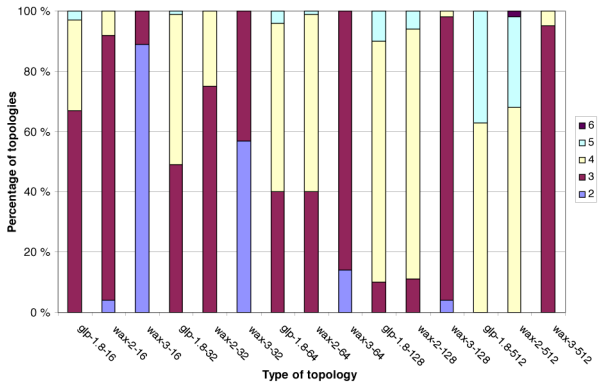


Fig. 5. The number of backup configurations required for a wide range of BRITE generated topologies. As an example the bar name wax-2-16 denotes that the Waxman model is used with a links-to-node ratio of 2, and with 16 nodes.

equal capacity. To achieve a good load distribution and minimize the chances of congestion in the failure-free case, we adopt the link weight optimization heuristic introduced in [23]. They define a piecewise linear cost function Φ that is dependent on the load $l(a)$ on each of the links a in the network. Φ is convex and resembles an exponentially growing function. They then introduce a local search heuristic that tries to minimize the value of Φ by randomly perturbing the link weights. This local search heuristic has been shown to give performance that is close to the optimal solution that can be achieved by a connection oriented technology like MPLS.

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

For our load evaluations, we use a gravity-style traffic matrix where the traffic between two destinations is based on the population of the countries they represent [22]. For simplicity, we look at constant packet streams between each node pair. The traffic matrix has been scaled so that the load on the most utilized link in the network is about 2/3 of the capacity. We use shortest path routing with equal splitting of traffic if there exists several equal cost paths towards a destination.

B. Number of Backup Configurations

Fig. 5 shows the minimum number of backup configurations that Algorithm 1 could produce in a wide range of synthetic topologies. Each bar in the figure represents 100 different topologies given by the type of generation model used, the links-to-node ratio, and the number of nodes in the topology. Table II shows the minimum number of configurations Algorithm 1 could produce for selected real-world topologies of varying size. For the Sprint US network, we show results for both the POP-level and router level topologies. The table also shows how many nodes that are covered by LFAs, and the number of configurations needed when MRC is used in combination with LFAs. Since some nodes and links are completely covered by LFAs, MRC needs to isolate fewer components, and hence the number of configurations decreases

TABLE II
NUMBER OF BACKUP CONFIGURATIONS FOR SELECTED REAL-WORLD NETWORKS

Network	Nodes	Links	Confs	LFA	Confs
Sprint US (POP)	32	64	4	17	4
Sprint US (R)	284	1882	5	186	5
Geant	19	30	5	10	4
COST239	11	26	3	10	2
German Telecom	10	17	3	10	-
DFN	13	37	2	13	-

for some topologies. We see that for the COST239 network, all nodes except one is covered by LFAs. However, we still need two backup configurations to cover this single node, because isolating all the attached links in a single configuration would leave the node unreachable.

The results show that the number of backup configurations needed is usually modest; 3 or 4 is typically enough to isolate every element in a topology. No topology required more than six configurations. In other words, Algorithm 1 performs very well even in large topologies. The algorithm fails only if it meets a node that if isolated disconnects the backbone in each of the n backup configurations. The algorithm often goes through all network nodes without meeting this situation even if n is low, and is more successful in topologies with a higher average node degree. The running time of our algorithm is modest; about 5 seconds for the router level Sprint US network.

In Section III-B we stated the problem of finding a minimal complete set of valid configurations can be transformed to the Set Covering problem. It has long been known that heuristic algorithms can efficiently approximate an optimal solution to this problem [24], which makes the good performance of Algorithm 1 less surprising.

This modest number of backup configurations shows that our method is implementable without requiring a prohibitively high amount of state information.

C. Backup Path Lengths

Fig. 6 shows path length distribution of the recovery paths after a node failure. The numbers are based on 100 different synthetic Waxman topologies with 32 nodes and 64 links. All the topologies have unit weight links, in order to focus more

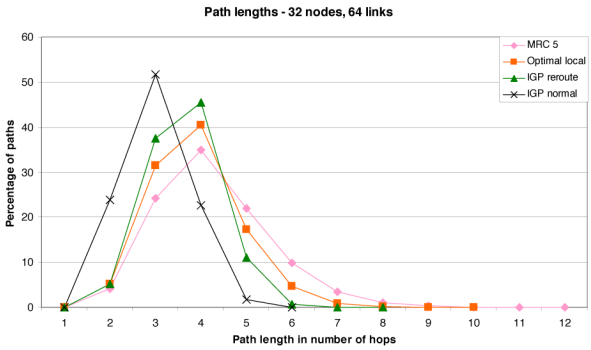


Fig. 6. Backup path lengths in the case of a node failure.

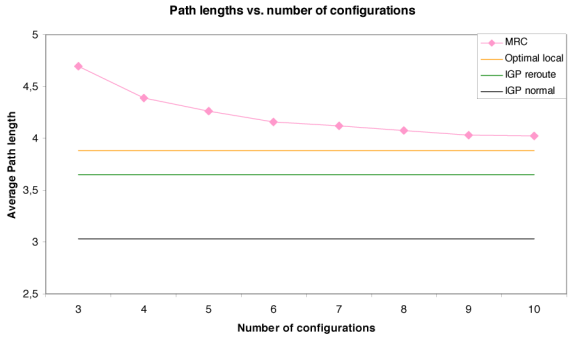


Fig. 7. Average backup path lengths in the case of a node failure as a function of the number of backup configurations.

on the topological characteristics than on a specific link weight configuration. Results for link failures show the same tendency and are not presented.

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

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

Algorithm 1 yields richer backup configurations as their number increases. In Fig. 7 we have plotted the average backup path lengths for the 75 of the 100 input topologies that could be covered using 3 backup configurations. The figure shows that the average recovery path length decreases as the number of backup configurations increases.

D. Load on Individual Links

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

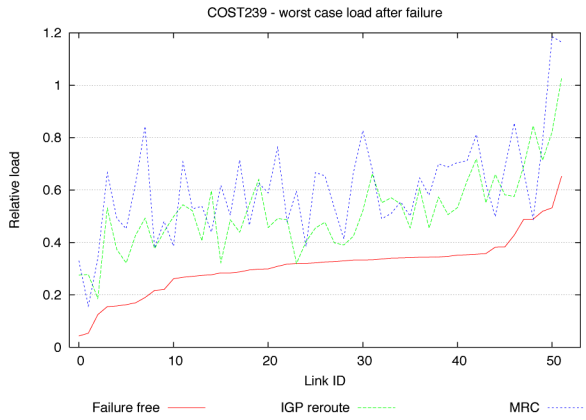


Fig. 8. Load on all unidirectional links in the failure free case, after IGP re-convergence, and when MRC is used to recover traffic. Shows each individual links worst case scenario.

Fig. 8 shows the maximum load on all links, which are indexed from the least loaded to the most loaded in the failure-free case. The results indicate that the restricted routing in the backup topologies result in a worst case load distribution that is comparable to what is achieved after a complete IGP rerouting process.

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

VI. Recovery Load Distribution

MRC recovery is local, and the recovered traffic is routed in a backup configuration from the point of failure to the egress node. This shifting of traffic from the original path to a backup path affects the load distribution in the network, and might lead to congestion. In our experience, the effect a failure has on the load distribution when MRC is used is highly variable. Occasionally the load added on a link can be significant, as we saw in Fig. 8. In this section, we describe an approach for minimizing the impact of the MRC recovery process on the post failure load distribution.

If MRC is used for fast recovery, the load distribution in the network during the failure depends on three factors:

(a) The link weight assignment used in the normal configuration C_0,
(b) The structure of the backup configurations, i.e., which links and nodes are isolated in each $C_i \in \{C_1, \ldots, C_n\}$,
(c) The link weight assignments used in the backbones $B_1, \ldots, B_n$ of the backup configurations.

The link weights in the normal configuration (a) are important since MRC uses backup configurations only for the traffic affected by the failure, and all non-affected traffic is distributed according to them. The backup configuration structure (b) dictates which links can be used used in the recovery paths for each failure. The backup configuration link weight assignments (c) determine which among the available backup paths are actually used.

Algorithm 3: Load-aware backup configurations.

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

Network operators often plan and configure their network based on an estimate of the traffic demands from each ingress node to each egress node. Clearly, the knowledge of such demand matrix D provides the opportunity to construct the backup configurations in a way that gives better load balancing and avoids congestion after a failure. We propose a procedure to do this by constructing a complete set of valid configurations in three phases. First, the link weights in the normal configuration are optimized for the given demand matrix D while only taking the failure free situation into account. Second, we take advantage of the load distribution in the failure free case to construct the MRC backup configurations in an intelligent manner. Finally, we optimize the link weights in the backbones of the backup configurations to get a good load distribution after any link failure.

Since the optimization procedure described in this section uses random search and relies on an estimate of the traffic matrix, it can only be implemented by a central network management unit. The link weight optimization is a computing intensive process, and should be conducted as a background refinement process.

The improvement of this proposal compared to the MRC backup configuration generation described in Section III is the congestion avoidance during MRC recovery. The functionality with respect to single link or node failure recovery guarantees is unchanged. Since the backup configurations are only used to route recovered traffic, we can optimize the structure and link weights used in these configurations for the different failure scenarios without sacrificing performance in the failure free case. We restrict ourselves to link failures in the optimization phase in order to limit the calculation complexity. A more detailed description of this procedure can be found in [25].

A. Algorithm

We first optimize the link weights in the normal configuration C_0, so that the cost of routing the demands D is minimized. For this purpose we use the link weight optimization heuristic and the cost function Φ introduced in [23], as described in the evaluation method in Section V.

Using the optimized link weights in C_0, we calculate the load $l(a)$ on each unidirectional link in the network in the failure-free case. In Algorithm 3 this information is used to construct backup configurations $C_1, \ldots, C_n$. The intuition behind our algorithm is that we want the amount of traffic that is potentially recovered in each backup configuration to be approximately equal. We want to avoid that the failure of heavily loaded links results in large amounts of traffic being recovered in backup configurations with a sparse backbone. Instead, this traffic should be routed in a rich (well connected) backbone, where we have a better chance of distributing it over less loaded links by setting appropriate link weights.

To implement this load-aware algorithm we calculate the *potential* of each node in the network and the potential of each backup configuration:

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

$$\gamma(u) = \sum_{v \in N} \left(l(u,v) + l(v,u) \right). \tag{11}$$

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

$$\gamma_i = \sum_{u \in S_i} \gamma(u). \tag{12}$$

Our modified backup configuration construction method is defined in Algorithm 3. As in Algorithm 1, the input to our algorithm for generating backup configurations is the normal configuration C_0, and the number n of backup configurations we want to create. We start our configuration generation algorithm by ordering all nodes with respect to their potential and assigning each node to a tentative backup configuration $C_{\mathrm{T}}(u)$ (line 6 in Algorithm 3), so that the potential γ_i of each backup configuration is approximately equal:

$$\gamma_i \approx \gamma_j, \quad i,j \in \{1,\ldots,n\}. \tag{13}$$

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

We then go through all nodes in the network, and attempt to isolate each node u in its tentative backup configuration (line 10). For some nodes, this might not be possible without breaking the definition of a connected backbone given in (5). This node is then attempted isolated in backup configuration C_{i+1}, C_{i+2} and so on (line 20), until all backup configurations are tried. If a node can not be isolated in any of the backup configurations, we give up and abort. Note that when nodes can not be isolated in the backup configuration it was assigned to, this will disturb the desired property of equalizing γ_i among the backup configurations. However, in our experience this typically only happens for a very limited number of nodes, and the consequences are not severe.

The outcome of this algorithm is dependent on the network topology and the traffic demand matrix D. If the load is close to equally distributed on the links before a failure, we end up with approximately the same number of nodes isolated in each backup configuration. If the traffic distribution is more skewed, the algorithm typically ends up with isolating many nodes with a small potential in C_1, while only very few nodes, with a high potential, are isolated in backup configuration C_n. This is in accordance with the goal of having a rich backbone in which to reroute traffic after the failure of heavily loaded links.

B. Weight Optimization

When the backup configurations $C_1, \ldots, C_n$ are created, our third phase consists of optimizing the link weights used in these configurations so that the recovered traffic is distributed over less utilized links. We use a weight search heuristic similar to the one used in the failure free case, and we adopt the cost function Φ introduced in [23]. Our goal is to find a weight function w_i for each configuration, so that the cost Φ of routing the demands through the network is as small as possible after any link failure. However, evaluating the Φ for a given weight setting is a complex task because it involves recalculating the load on all links in the network. In order to get a method that scales to large networks, we identify a limited set of *critical* link failures. We then optimize the link weights taking only the critical link failures into account.

Let Φ^a denote the cost of routing the demands through the network when link a has failed. We define the critical link set L_C as the k links that give the highest value of Φ^a upon failure, i.e., L_C is the set of links with cardinality k so that $\forall a \in L_C, b \notin L_C : \Phi^a \geq \Phi^b$. The setting of k can be used to balance the result precision with the computation complexity. Note that the initial calculation of L_C is performed after we have optimized w_0, but before we have optimized $w_1, \ldots, w_n$. To capture possible changes caused by the altered link weights in the backup configurations, the set of critical link failures is periodically updated during the link weight search.

With MRC, there is a dependency between a particular link failure and the backup configurations that are used to recover the traffic, as explained in Section IV. This means that for the failure of a particular link (u, v), the distribution of recovered traffic is determined only by the link weights in the backup configurations where nodes u and v are isolated. We take advantage of this to further reduce the number of cost evaluations performed in our local search heuristic. For each backup configuration C_i, we define $L_i \subseteq L_C$ as the set of critical links whose failure results in recovered traffic being routed according to C_i:

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

$$L_i = \{a \in L_C | a \notin B_i\}. \tag{14}$$

Given the set of critical link failures for each backup configuration, we run the local search heuristic with the objective of minimizing the sum $\sum_{a \in L_C} \Phi^a$. Each time we change a link weight $w_i(a)$ in backup configuration i, we only need to evaluate the resulting cost Φ^a after the failure of each link a in L_i. For all other link failures, this cost will be the same.

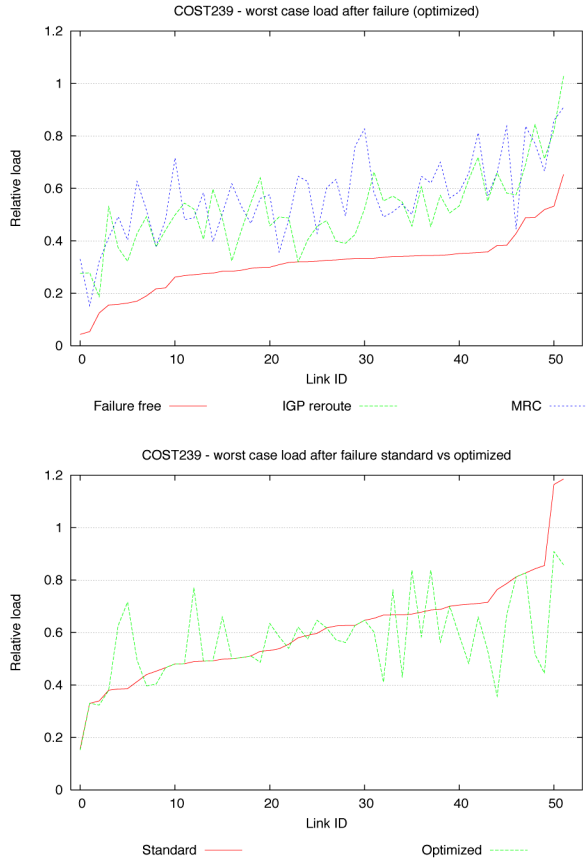


Fig. 9. Load on all unidirectional links in the COST239 network after the worst case link failure. Top: Optimized MRC versus complete IGP rerouting. Bottom: Standard versus optimized MRC.

The local search heuristic terminates and returns the weight setting that gives the lowest value of the objective function after a given number of iterations.

C. 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 Section 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 highlighted 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

TABLE III
CONCEPTUAL COMPARISON OF DIFFERENT APPROACHES FOR FAST IP RECOVERY

Scheme	Guaranteed in bi-connected	Node faults	Link faults	Pre-configured	Connec-tionless	Failure agnostic	Last hop
MRC	yes	yes	yes	yes	yes	yes	yes
Not-via tunneling [11]	yes	yes	yes	yes	yes	yes	yes
Local rerouting [28]	no	no	yes	no	yes	N/A	N/A
FIR [8]	yes	no	yes	yes	yes	N/A	N/A
FIFR [29]	yes	yes	yes	yes	yes	yes	no
LFA [21]	no	yes	yes	yes	yes	yes	yes
MPLS FRR [26]	yes	yes	yes	yes	no	no	N/A
Rerouting (OSPF)	yes	yes	yes	no	yes	yes	yes

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 in Section 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.

VII. RELATED WORK

Much work has lately been done to improve robustness against component failures in IP networks [10]. In this section, we focus on the most important contributions aimed at restoring connectivity without a global re-convergence. Table III summarizes important features of the different approaches. We indicate whether each mechanism guarantees one-fault tolerance in an arbitrary bi-connected network, for both link and node failures, independent of the root cause of failure (failure agnostic). We also indicate whether they solve the "last hop problem".

Network layer recovery in the timescale of milliseconds has traditionally only been available for networks using MPLS with its fast reroute extensions [26]. In the discussion below, we focus mainly on solutions for connectionless destination-based IP routing. A related survey can be found in [27].

IETF has recently drafted a framework called IP fast reroute [30] where they point at Loop-Free Alternates (LFAs) [21] as a technique to partly solve IP fast reroute. From a node detecting a failure, a next hop is defined as an LFA if this next hop will not loop the packets back to the detecting node or to the failure. Since LFAs do not provide full coverage, IETF is also drafting a tunneling approach based on so called "Not-via" addresses to guarantee recovery from all single link and node failures [11]. Not-via is the connectionless version of MPLS fast reroute [26] where packets are detoured around the failure to the next-next hop. 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 tunneling approach may give less optimal backup paths, and less flexibility with regards to post failure load balancing.

Narvaez *et al.* [28] 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 [31]. 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 bi-connected 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.

Nelakuditi *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. In another paper, they have also proposed a similar method, named Failure Inferencing based Fast Rerouting (FIFR), for handling node failures [29]. 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". FIFR guarantees one-fault-tolerance in any bi-connected network, it is connectionless, pre-configured and it does not affect the original failure-free routing.

Our main inspiration for using multiple routing functions to achieve failure recovery 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 $\mathrm{Up}*/\mathrm{Down}*$ [32]. 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 [33], [34]. 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. In this paper we refine these ideas and adapt them to an IP setting.

None of the proactive recovery mechanisms discussed above take any measures towards a good load distribution in the network in the period when traffic is routed on the recovery paths. Existing work on load distribution in connectionless IGP networks has either focused on the failure free case [23], [35], [36], or on finding link weights that work well both in the normal case and when the routing protocol has converged after a single link failure [37]–[39].

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.

VIII. Conclusion

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 bi-connected 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 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 thus achieves fast recovery with a very limited performance penalty.

References

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

[2] A. Basu and J. G. Riecke, "Stability issues in OSPF routing," in *Proc. ACM SIGCOMM*, San Diego, CA, Aug. 2001, pp. 225–236.

[3] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian, "Delayed internet routing convergence," *IEEE/ACM Trans. Networking*, vol. 9, no. 3, pp. 293–306, Jun. 2001.

[4] C. Boutremans, G. Iannaccone, and C. Diot, "Impact of link failures on VoIP performance," in *Proc. Int. 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 *Proc. 23rd Int. Conf. Distributed Computing Systems (ICDCS'03)*, Washington, DC, 2003, pp. 204–213, IEEE Computer Society.

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

[7] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C.-N. Chuah, and C. Diot, "Characterization of failures in an IP backbone network," in *Proc. IEEE INFOCOM*, Mar. 2004, vol. 4, pp. 2307–2317.

[8] S. Nelakuditi, S. Lee, Y. Yu, Z.-L. Zhang, and C.-N. Chuah, "Fast local rerouting for handling transient link failures," *IEEE/ACM Trans. Networking*, vol. 15, no. 2, pp. 359–372, Apr. 2007.

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

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

[11] S. Bryant, M. Shand, and S. Previdi, "IP fast reroute using not-via addresses," Internet Draft (work in progress), draft-ietf-rtgwg-ipfrr-notvia-addresses-01, Jun. 2007.

[12] P. Francois, M. Shand, and O. Bonaventure, "Disruption free topology reconfiguration in OSPF networks," in *Proc. IEEE INFOCOM*, Anchorage, AK, May 2007, pp. 89–97.

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

[14] 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), draft-ietf-ospf-mt-07.txt, Nov. 2006.

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

[16] M. Menth and R. Martin, "Network resilience through multi-topology routing," in *Proc. 5th Int. Workshop on Design of Reliable Communication Networks (DRCN)*, Oct. 2005, pp. 271–277.

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

[18] B. M. Waxman, "Routing of multipoint connections," *IEEE J. Sel. Areas Commun.*, vol. 6, no. 9, pp. 1617–1622, Dec. 1988.

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

[20] Rocketfuel Topology Mapping. [Online]. Available: http://www.cs.washington.edu

[21] A. Atlas and A. Zinin, "Basic specification for IP fast-reroute: loop-free alternates," IETF Internet Draft (work in progress), draft-ietf-rtgwg-ipfrr-spec-base-06, Mar. 2007.

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

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

[24] D. S. Johnson, "Approximation algorithms for combinatorial problems," in *Proc. 5th Annu. ACM Symp. Theory of Computing*, 1973, pp. 38–49.

[25] A. Kvalbein, T. Čičić, and S. Gjessing, "Post-failure routing performance with multiple routing configurations," in *Proc. IEEE INFOCOM*, May 2007, pp. 98–106.

[26] P. Pan, G. Swallow, and A. Atlas, "Fast reroute extensions to RSVP-TE for LSP tunnels," RFC 4090, May 2005.

[27] A. Raja and O. C. Ibe, "A survey of IP and multiprotocol label switching fast reroute schemes," *Comput. Netw.*, vol. 51, no. 8, pp. 1882–1907, Jun. 2007.

[28] P. Narvaez, K.-Y. Siu, and H.-Y. Tzeng, "Local restoration algorithms for link-state routing protocols," in *Proc. IEEE Int. Conf. Computer Communications and Networks (ICCCN'99)*, Oct. 1999, pp. 352–357.

[29] 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 *Proc. IEEE INFOCOM*, Mar. 2005, vol. 4, pp. 2859–2863.

[30] M. Shand and S. Bryant, "IP fast reroute framework," IETF Internet Draft (work in progress), draft-ietf-rtgwg-ipfrr-framework-07, Jun. 2007.

[31] R. Rabbat and K.-Y. Siu, "Restoration methods for traffic engineered networks for loop-free routing guarantees," in *Proc. IEEE Int. Conf. Communications (ICC'01)*, Helsinki, Finland, Jun. 2001, vol. 5, pp. 1566–1570.

[32] I. Theiss and O. Lysne, "FRoots, a fault tolerant and topology agnostic routing technique," *IEEE Trans. Parallel Distrib. Syst.*, vol. 17, no. 10, pp. 1136–1150, Oct. 2006.
[33] A. F. Hansen, T. Čičić, S. Gjessing, A. Kvalbein, and O. Lysne, "Resilient routing layers for recovery in packet networks," in *Proc. Int. Conf. Dependable Systems and Networks (DSN 2005)*, Jun. 2005, pp. 238–247.
[34] A. Kvalbein, A. F. Hansen, T. Čičić, S. Gjessing, and O. Lysne, "Fast recovery from link failures using resilient routing layers," in *Proc. 10th IEEE Symp. Computers and Communications (ISCC 2005)*, Jun. 2005, pp. 554–560.
[35] Y. Wang, Z. Wang, and L. Zhang, "Internet traffic engineering without full mesh overlaying," in *Proc. IEEE INFOCOM*, Apr. 2001, pp. 565–571.
[36] A. Sridharan, R. Guerin, and C. Diot, "Achieving near-optimal traffic engineering solutions for current OSPF/IS-IS networks," *IEEE/ACM Trans. Networking*, vol. 13, no. 2, pp. 234–247, Apr. 2005.
[37] A. Nucci, B. Schroeder, S. Bhattacharyya, N. Taft, and C. Diot, "IGP link weight assignment for transient link failures," in *Proc. 18th Int. Teletraffic Congress*, Berlin, Germany, Aug. 2003.
[38] B. Fortz and M. Thorup, "Robust optimization of OSPF/IS-IS weights," in *Proc. INOC*, Oct. 2003, pp. 225–230.
[39] A. Sridharan and R. Guerin, "Making IGP routing robust to link failures," in *Proc. Networking*, Waterloo, Canada, 2005.

Amund Kvalbein (M'05) received the M.Sc. and Ph.D. degrees from the University of Oslo, Oslo, Norway, in 2003 and 2007, respectively. The main focus of his Ph.D. thesis was fast recovery mechanisms.

He is now a Postdoctoral Fellow at Simula Research Laboratory, Lysaker, Norway, where he is leading a project on network resilience. His main research interests are network layer protocols, in particular issues related to fault tolerance, scalability and robustness under unexpected operational environments.

Audun Fosselie Hansen received the M.Sc. and Ph.D. degrees from the University of Oslo, Oslo, Norway, in 2001 and 2007, respectively. His thesis focused on fast reroute in IP networks.

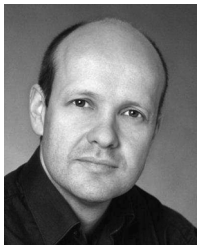
He is currently working for Telenor R&I and Simula Research Laboratory as a Project Manager for the joint SimTel project. The focus of this project is increased service availability and quality to heterogeneous mobile terminals in heterogeneous environments.

Tarik Čičić (M'00) received the M.S. and Ph.D. degrees from the University of Oslo, Oslo, Norway, in 1997 and 2002, respectively.

He was a Postdoctoral Fellow at Simula Research Laboratory, Norway, from 2003 to 2007. Currently he is the CTO of Media Network Services, Norway, and an Associate Professor at the Department of Informatics, University of Oslo. His research interests include network architectures, communication protocols modeling, network measurements, and network resilience.

Stein Gjessing (M'91) received the Dr. Philos. degree in 1985.

He is currently a Professor of computer science in the Department of Informatics, University of Oslo, Oslo, Norway, and an adjunct researcher at Simula Research Laboratory, Norway. His original work was in the field of object-oriented concurrent programming. He has worked with computer interconnects [Scalable Coherent Interface (IEEE Std. 1596) and LAN/MANs (Resilient Packet Ring, IEEE Std. 802. 17)]. His main research interests are currently within network resilience, including sensor networks, Internet-like networks and optical networks.



Olav Lysne (M'92) received the Masters and the Dr. Scient. degrees from the University of Oslo, Oslo, Norway, in 1988 and 1992, respectively.

He is now Research Director at Simula Research Laboratory, and a Professor in computer science at the University of Oslo. His early research contributions were in the field of algebraic specification and term rewriting. However, his most important scientific contributions are in interconnection networks, focusing on problems like effective routing, fault tolerance and quality of service. In this area, he has participated in numerous program committees, and he served as general chair of ICPP 2005.