Resilient Routing Layers for Recovery in Packet Networks

Audun Fosselie Hansen[1,2] Amund Kvalbein[1] Tarik Čičić[1] Stein Gjessing[1]
Olav Lysne[1]
[1] Networks and Distributed Systems Group
Simula Research Laboratory
Fornebu, Norway
{audunh, amundk, tarikc, steing, olavly}@simula.no

[2]Telenor R&D, Fornebu, Norway

Abstract

Most existing methods for network recovery are often complex and seldom used by network administrators. In this paper we present a novel approach for global and local recovery named Resilient Routing Layers (RRL). The method is supported by algorithms, but also simple enough for a network administrator to implement by hand for reasonably sized networks. The idea in our approach is that for each node in the network there is a topology subset called a "safe layer", which can handle any traffic affected by a fault in the node itself, or any of its links.

We demonstrate that our approach performs well compared to other comparable methods in a wide range of different network topologies. Particularly, we demonstrate RRLs performance for what are assumed to be the weakest parameters for our method, i.e., backup-path lengths and state information overhead. We discuss implementation issues of RRL, and demonstrate its applicability to MPLS networks.

1 Introduction

Network resilience is an area of growing importance in communication systems research and engineering. Recent history of the Internet shows its vulnerability on all levels, from physical sabotage to failing links and routers. Numerous techniques have been developed to prevent, repair and repel the damage, and among these, network recovery techniques have received significant attention.

There is, however, a discrepancy between the network recovery in theory and practice. While the theoretical research has devised schemes of high elegance and performance like p-cycles [8] and redundant trees [15], many network engineers still use static, manually-laid protection paths. The problem is that existing sophisticated algorithms often suffer from high complexity and lack of clear control and management view for the network operators.

The need for simplicity of deployment has also been supported empirically. Through extensive network monitoring, Labovitz and others demonstrated that most communication outages in IP networks stemmed from software/hardware bugs and misconfiguration in routers [13], [12]. Complicated deployment are obvious sources of misconfiguration, thus recovery methods should not add significant complexity to the overall system. These observations have attained great impact on the design of our recovery approach.

We use recovery as a common term for *protection* and *restoration*. Protection schemes calculate the backup routes in advance, while restoration schemes calculate the backup routes upon detection of failures. Thus, protection operates in a much shorter time-scale than restoration.

Recovery schemes can also be categorized by the scope of the recovery. *Global recovery* covers link and node failures by calculating a new end-to-end path, while with *local recovery* faults are handled locally by the neighbors. Point of repair, i.e., the node initiating protection switching or restoration, is the ingress node of a path for global recovery and the neighbor node detecting the fault for local recovery. For global recovery, the recovery action is not performed before the ingress node has been notified about the failure, and hence global recovery operates on a longer time-scale than local recovery.

The success of a recovery action depends not only on the properties of the recovery scheme in use. If the failure of a node or link physically disconnects the network, traffic routed over such a node or link can not be recovered. Such nodes and links are termed articulation points.

In this paper we propose a technique we call "Resilient Routing Layers" (RRL). RRL differs from other recovery schemes by that it is designed with the systems engineering mindset. In other words, it is simple to understand and deploy, and it is made to be used by the network engineers in practice. One main feature of RRL, as will be thoroughly described later, is that it provides a network manager with simple global abstractions of the network that form a basis for routing traffic in failure situations. The simple global abstractions is what we refer to as *routing layers*. RRL offers flexibility in that the abstractions can be built differently to optimize different parameters, such as backup path lengths, state amount, and multiple fault protection.

The rest of the paper is organized as follows: In Sec. 2 we give an overview of the most relevant related work. Sec. 3 defines RRL and discusses its features. In Sec. 4 we discuss the number of layers and backup path lengths and give a general comparison with other schemes. Sec. 5 demonstrates the applicability for MPLS networks and finally we conclude and give some future research directions in Sec. 6.

2 Related work

From a graph-theoretical point of view, methods for fault tolerance rely on a graph property described by Menger (Menger's theorem) [17]: *a nontrivial graph G is k-connected if and only if for each pair (u, v) of distinct vertices there are at least k internally disjoint $u - v$ paths in G.* A bi-connected graph will then provide each u, v pair with two internally disjoint paths. In general a graph is said to be k-connected with respect to vertices if removal of any $k - 1$ vertices leaves the graph connected. The same applies with respect to edges. To guarantee one fault tolerance for every vertex, a network must be at least biconnected with respect to vertices.

These properties have served as foundation for most work regarding recovery. One of the most studied approaches has been algorithms for finding disjoint paths between sources and destinations in a network. Suurballe presents an algorithm for finding k vertex-disjoint shortest paths [25]. The algorithmic run-time is $\mathcal{O}(|V|^2 \cdot log|V|)$, where V is the number of vertices. Later on, other algorithms have been proposed to minimize the complexity. Some of them have assumed certain assumptions and short-cuts like maximally disjoint paths instead of totally disjoint paths, and edge-disjoint paths instead of vertex-disjoint paths [30], [26], [14].

Recovery by end-to-end disjoint paths relies on notification about failures to the ingress nodes. To avoid such notification, a local variant could be implemented. Each node should then initiate establishment of backup paths to cover each possible failure in its neighbor links and nodes [19]. So, from each node, to each potential egress node a backup path should be established for every possible failure in the neighborhood. Such a strategy would provide a network manager with a unsurmountable number of paths to overview. Figure 1 gives an example of how it would look for a simple network with three nodes and three links.

Restoration serves as an alternative to such precalculations. One main parameter of optimization for restoration schemes is the time used to calculate a new path upon detection of a failure. Afek and others propose using k+1 original shortest paths to recover k edge failures [5]. MPLS label stacking is used to implement this approach. No path calculation is needed after detection, only the decision on what path to use remains. The authors demonstrate that the path table sizes decrease, and that their method doesn't add considerable length to the backup paths compared to standard methods. Otel presents an incremental Dijkstra algorithm offering fast local backup path calculation for MPLS rerouting [18]. The algorithm takes as input the existing outdated shortest path three rooted at the local node. The algorithm has a complexity close to $\mathcal{O}(|V|)$, while standard Dijkstra has $\mathcal{O}(|V|^2)$ complexity, where $|V|$ is the number of nodes in the network.

Network recovery management is difficult if the only offered view of the network is a collection of unstructured backup paths. The literature provides however some alternatives for more structured recovery. Such schemes are based on building a set of subtopologies of the network, serving as a more intuitive abstraction of the recovery paths. These schemes can serve as input to restoration and protection, both global and local.

Itai and Rodeh generalize and structure the disjoint path approach to spanning trees [11]. To overcome the failure of less than k edges, they present a communication protocol which uses k spanning trees having the property that for every vertex v, the k paths from v to the root are edge-disjoint. In this way one common calculation provides the network with disjoint paths between one source and several destinations or between several sources and one destination. They show how their algorithmic run-time is proportional to the number of edges. Medard and others use the multi-tree approach to generate both edge and vertex redundant trees for arbitrary biconnected networks [15]. These trees, named red and blue, are such that any node is connected to the common root of the trees by at least one of the trees in case of a vertex or edge failure. They prove that the algorithmic complexity is $\mathcal{O}(|V|^3)$. Xue and others optimize this algorithm for generating trees based on QoS constraints like cost and delay [31].

Grover and Stamatelakis introduce another concept providing fault tolerance called protection cycles [8], [9]. Their goal is to provide circuit oriented mesh networks with the fast recovery speed normally offered in rings. Similar approaches using Hamiltonian cycles are presented in [10] and [21]. Their method calculates one or more cycles visiting all nodes in the network. The method is optimized to cover link failures and also to minimize the over-provisioning ratio on any link in the network. When a link fails, the traffic is locally switched to be routed according to the cycle instead of the original shortest path. Following a cycle, avoiding the a failure, will probably add considerable path length compared to shortest path, however the authors do not comment on that.

The three main categories of schemes presented above, i.e., disjoint paths, redundant trees and protection cycles, have first been introduced for recovery in circuit switched networks. As technologies for connection-oriented packet switched networks have evolved they have also been adopted for such networks. One such technology is Multiprotocol label switching (MPLS) that provides a flexible framework for traffic engineering in general, where recovery is an important part [20]. IETF is standardizing a framework and ancillary protocols for MPLS recovery [22], [19]. IETF is not detailing algorithms on how to find backup resources, only how to signal and represent them.

Algorithms for finding disjoint paths can be directly deployed for giving input to the signaling mechanisms of MPLS, while redundant trees and protection cycles need some adjustments.

Barthos and Raman demonstrate the applicability of Medards dual-tree approach [15] for MPLS recovery [3], [4]. Their method differs from [15] in using the egress node, i.e., destination, as root. In addition, they calculate optimal primary paths, using the blue and red trees only for recovery. So, in addition to a primary path for each pair of ingress-egress nodes, they calculate red-blue trees for every egress. The authors demonstrate that the approach requires few labels and that the backup path lengths are not considerable longer than for MPLS fast reroute.

Grover and Stamatelakis also adapt the concept of protection cycles to IP/MPLS networks [23], [24]. They optimize for link failures and oversubscription ratio.

In the following we will describe how RRL provides a simple topology abstraction, i.e., routing layers, for the network manager to administer the recovery of the network. The work most closely related to our approach is [27] which treats interconnection networks for computer clusters. Here a layer-based approach is used to obtain deadlock-free and fault-tolerant routing in irregular cluster networks based on a routing strategy called Up*/Down*. RRL is not hampered by deadlock-considerations necessary in interconnection networks. It extends the concept of layers, and shows its applicability to general packet networks and routing strategies.

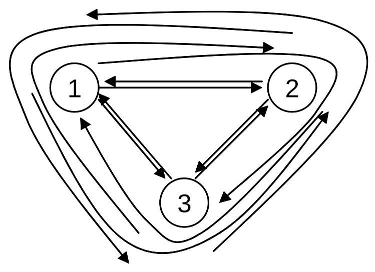

Figure 1. All-to-all backup paths in a 3-node cycle.

3 Resilient Routing Layers (RRL)

3.1 Overview of RRL

RRL is based on calculating redundant subsets of the network topology that we call *layers*. Each layer contains all nodes but only a subset of the links in the network. We say that a *node is safe* in a layer if only one of its links is contained in that layer. We will use the term *safe layer for a node* to denote a layer in which the node is safe.

The layers are used as input to routing or path-finding algorithms, calculating a routing table or path table for each layer. We assume that for each layer an algorithm is used to find loop-free paths between all pairs of source and destination. Therefore all pairs of nodes can reach each other in any layer.

We observe the following:

1. In a safe layer for a given node, this node will not experience transit traffic.

2. If a node fails, any safe layer for that node keeps an intact path between all pairs of sources and destinations that are distinct from the node itself.

3. If a node fails, traffic sourced by or destined for the failed node will be lost under any circumstance.

In order to use RRL as a basis for a complete method for recovery, we need to generate layers in such a way that all nodes that are not articulation points are safe in at least one layer. As we shall demonstrate later on, this can be achieved with relatively few layers.

The concept above can be used both for global and local recovery. In global recovery the packets that should have traversed the failed node are made to use the node's safe layer from the source. In local recovery, the node upstream of the failed node transfers the packet to the safe layer of the failed node when the packets arrive.

RRL handles link-faults as follows. First, we define a leaf link as the link connecting a safe node to the network. A *safe layer of a link* can be defined as the safe layer of its downstream node n as long as n is not the final destination and its leaf link is not the failed link.

If the downstream node n is the final destination and its leaf link is the failed link, we have two options:

1. We use the safe layer of the detecting node (upstream node) as long as its leaf link is not the failed link.

2. If so is, the safe layer of the upstream node is still the safe layer of the link, but the upstream node deflects the traffic to another link. The safe layer of the detecting node will route no traffic through this node, and hence the traffic will not loop back to the failure.

Since link failures are handled using the safe layer of a node, most of the following examples and evaluations will focus on node failures.

3.1.1 An Example

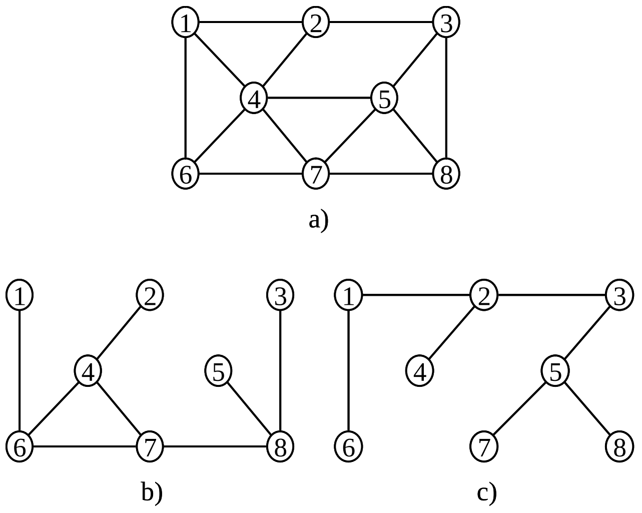

Figure 2. a): An example network with 8 nodes and 14 links. b): layer 1 (L_1) generated based on a). c): layer 2 (L_2) generated based on a).

We demonstrate a method for generating safe layers for all nodes by the following example, Fig. 2a being the starting point. This network has no original articulation points. The resulting layers are presented in Fig. 2b and Fig. 2c. The first layer (L_1) will be calculated starting with node 1 as a candidate safe node. Since node 1 is not an articulation point, i.e., its removal does not disconnect the network, we remove links and make node 1 a safe node. Node 2 is then analyzed and found eligible as a safe node in the same layer as node 1. The same is the case with node 3, but node 4 has become an articulation point, so node 4 is not safe i L_1. Finally, layer 1 (L_1) will be the safe layer of the nodes 1, 2, 3 and 5. Note that a layer may contain cycles as seen in layer 1.

When all remaining nodes are articulation points in L_1, layer L_2 is calculated. It starts with e.g., node 4, and covers nodes 4, 6, 7 and 8. In other words, our example network can be covered with only two layers.

For reasonably sized networks, generation of layers could easily be done manually by a network manager.

3.1.2 Implementation Considerations

To take advantage of the resilient routing layers, a packet network implementation must fulfill certain requirements. These requirements depend on whether the network operates in a connectionless or a connection-oriented manner.

For a connectionless network, e.g., IP or Ethernet, each packet must be marked according to what layer is currently valid. If n is the maximum number of layers, $log_2(n)$ bits in the packet header should identify the currently valid layer. The node that moves a packet to another layer, marks the packet header with the global identification of the new layer. In the case of failures, only traffic transiting the failed node should be moved to another layer. All packets not affected by the fault will still be routed based on the full topology. The node moving traffic to another layer must therefore know if a failed node is in the route of a packet. For local recovery, this is obviously fulfilled since it is the next hop that is failing. For global recovery, we must ensure that the ingress node is aware of the entire route for the traffic.

For a connection-oriented network, e.g., MPLS, marking packets with a global layer identification is not necessary. Path signaling is performed as normal. For each layer a new set of paths must be signaled. As for connectionless networks, the node moving traffic from original paths to recovery paths must know what paths are affected by the failure.

Fig. 3 gives an example of how traffic is switched between layers when node 5 is failing. The dotted links are not available in layer 1, i.e., the safe layer of node 5. Before node 5 fails, all traffic may use the full topology, e.g., traffic from node 6 to node 3 may follow the path 6-4-5-3. When node 5 fails, traffic transiting node 5 must be routed according to layer 1 (removing the dotted links.), while other traffic can still be routed according to the full topology. In the case of local recovery, traffic is routed from node 6 to 4 according to the full topology. Node 4 detects the failure, and switch traffic to layer 1. The path for traffic between node 6 and node 3 will then be 6-4-7-8-3. If node 6 is notified about the failure (global recovery) of node 5, the transition to layer 1 could be done by node 6. The path would then be 6-7-8-3. Even if node 5 has failed, our method can still handle failures of nodes 1, 2 and 3.

If a failure is defined as permanent, new layers must be calculated based on the full topology without out the failed component.

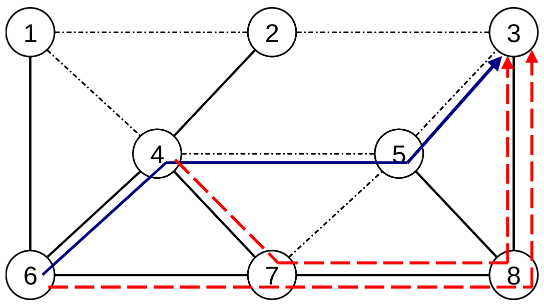

Figure 3. Example of how affected traffic are switched to layer 1 (L_1) when node 5 is failing.

3.2 An Example Algorithm

There are numerous ways for choosing which nodes should be safe in which layer. RRL can be made to be optimized on many different criteria, thus it displays significant flexibility. One alternative could be to have an algorithm generating a preferred fixed number of layers. For simplicity this section presents an algorithm making choices that in a very straightforward way attempts to minimize the number of layers[1]. The algorithm calculates layers in the topology $G = (V, E)$, where V is the set of nodes and E is the set of links.

[1]An implementation of this algorithm, together with the whole evaluation framework used in this paper can be retrieved from *http://www.simula.no*

```
(1) $S = \text{artPoints}(G)$;
      while $(S \neq V)$
          $L_i = G; V' = V \backslash S$;
          foreach $n \in V'$
              if $(n \notin \text{artPoints}(L_i))$
(2)                 $E' = \text{links}(n, L_i)$;
(3)                 $L_i = L_i \backslash \{l_j \in E' \mid 1 \leq j < |E'|\}$;
                    $S = S \cup \{n\}$;
              endif
          endfor
          store layer $L_i$;
          $i = i + 1$;
      endwhile

(4) balanceLayers();
```

Steps (1)-(4) deserve some comments. (1): Set S keeps track of the processed nodes, i.e., nodes that are either articulation points or safe nodes in an already computed layer. Initially, all articulation points in G are added to set S. artPoints(G) finds all articulation points in G. (2): We find all the adjacent links of the node ($E' = \text{links}(n, L_i)$), and then we (3): remove all adjacent links but one from the current topology of the layer. (4): So far, the algorithm attempts to make as many nodes as possible safe in the first layers. The first layers will then contain a majority of safe nodes. A typical distribution of safe nodes for a topology with 128 nodes could be 80, 36, and 12 for the resulting three layers. The layers with most safe nodes will contain fewer links and therefore offer more inefficient routing than other layers. To attain more equal routing performance in the layers, we do some postprocessing of the layers to balance the number of safe nodes. This is done by moving safe nodes from the layers with high degree of safe nodes to the layers with low degree of safe nodes, with the requirement that they are not articulation points in the new candidate layer. In addition, we must assure that we do not accidentally make an unintended safe node unsafe when adding links to the layer of high degree of safe nodes.

The authors are not aware of any algorithm running in polynomial time that finds a guaranteed minimum number of layers. The proposed heuristic algorithm performs well, but does not provide any guarantees that the number of layers will indeed be minimal.

For an arbitrary node in an arbitrary graph, it can be determined whether the node is an articulation point in $\mathcal{O}(|V| + |E|)$ [29]. This articulation point test is done within a nested while- and for-loop. In a theoretical worst case we need one layer for each node in the network, and hence the while-loop will iterate $|V|$ times. The for-loop will for each while-iteration iterate $|V| - c$ times where c is the number of while-iterations currently accomplished. In worst case the total running time for these steps will then be $\mathcal{O}(|V|^2 \cdot (|V| + |E|))$, which in all graphs of practical interest is $\mathcal{O}(|V|^3)$. The balancing method runs with the same complexity, and hence the total running time is bound within $\mathcal{O}(|V|^3)$.

4 Evaluation results

4.1 Scalability - Number of Layers

The scalability of RRL is directly dependent on the number of layers needed to provide a safe layer for each node in the network. Fig. 4 presents the number of layers for a variety of topologies. The calculations are performed with the algorithm from section 3.2. We have used the Brite topology tool to generate synthetic topologies [16]. Two main categories of generation models have been used, Waxman (wa) with default settings [28] and Generalized Linear Preference (glp) with two different settings (glp1 and glp2) [6]. In addition, we have used a set of real world topologies collected from Rocketfuel (Rfuel) [1]. These are intra-provider POP-level topologies. We have generated 100 topologies for each synthetic topology specification, i.e., for each bar in Fig. 4. The bar-name in the figure denotes the model used, number of nodes and number of links (model-nodes-links). The entries with and 'X' denotes that the number of links or nodes has varied within the the category of generated topologies. For glp1 networks the average nodal degree has varied around three, and for glp2 networks the average nodal degree has been about four. The RocketFuel networks (31 networks) represent networks with a wide variation in number of nodes and number of links.

Extracting results from Fig. 4 we have that five layers seem sufficient even for very large networks. We also observe that the number of layers decreases as the average node degree increases.

4.2 RRL Backup Path Lengths

With RRL, traffic affected by a failure will be routed according to the safe layer of the failed component. As illustrated in section 3, a layer has a reduced set of links compared to the full topology. This will of course have consequences for the length of the backup paths, as there are less links to choose from. In this section we present the distribution of RRL backup path lengths for one-fault-tolerance in a collection of topologies. We look at node failures and measure the backup path lengths when applying local protection switching. These are compared with the failure-free primary path and the optimal backup path, which is the shortest path upon removal of the failed component only. In most cases we use the algorithm from section 3.2 to generate layers as input to backup path calculation. The number of layers are then close to a minimum. To

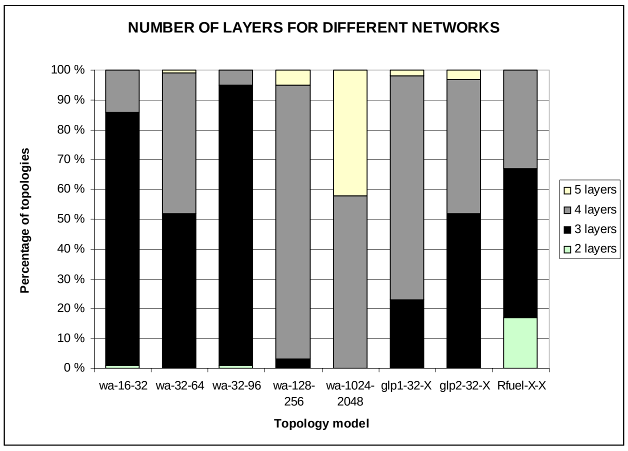

Figure 4. Number of layers for different types of topologies (synthetic (wa, glp1, glp2) and real world (Rfuel)).

demonstrate that there exist a trade-off between the number of layers, and backup path lengths, we show some results using more layers than the minimum. With more layers, each layer will contain less safe nodes, and thus more links will be available for routing.

For each original path in a topology we have calculated one backup path. This backup path has been chosen as the median length backup path from the collection of backup paths for all component failures on the original path.

Figure 5 gives the distribution for 100 topologies based on the waxman model with 32 nodes and 64 links. In addition, we have plotted the RRL backup path lengths for 100 GLP topologies having similar settings. We observe that there are no major differences in the results obtained using different models. The main observation is that RRL backup paths are longer than the optimal backup paths, however, we find that the differences are within acceptable bounds. The average length for for optimal backup paths is 3.5, as RRL gives an average of 4.3.

Figure 6 shows the same pattern for networks with higher node degrees, hence showing shorter average lengths. It also shows how the relative distribution is repeated for larger networks, although with longer average lengths.

RRL provides a high degree of freedom in how to build layers and how many layers to build. Figure 7 compare the distributions of backup path lengths for optimal backup paths and RRL backup paths with the number of layers varying from the original number generated with the algorithm from Sec. 3.2 to a fixed number of eight. The figure shows how increasing state, i.e., number of layers, will give more efficient backup path routing.

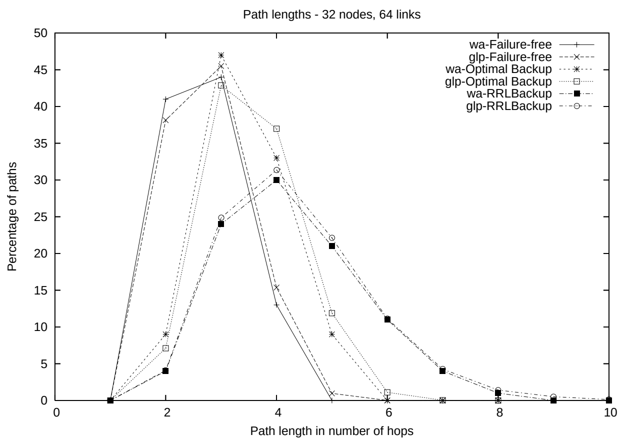

Figure 5. Distributions of path lengths when introducing one node failure in the original path.

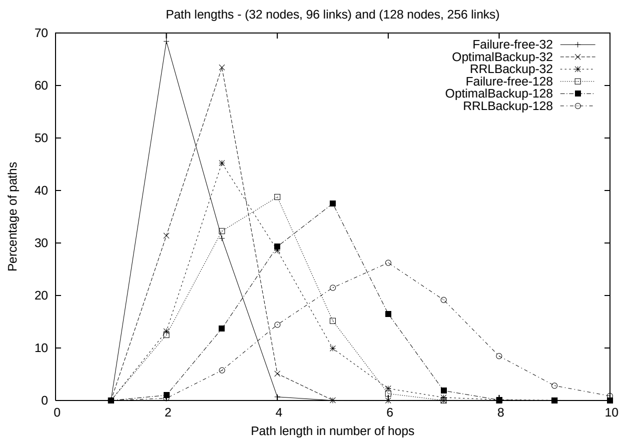

Figure 6. Distributions of path lengths when introducing one node failure in the original path.

4.3 Comparisons

In this section we make a comparison between the three main approaches offering various recovery abstractions. These are RRL, Protection Cycles and Redundant Trees. Table 1 gives an overview for some relevant criteria. For RRL, the numeric values have been evaluated using 100 Waxman networks with 16 nodes and 32 links. Statistics on backup path lengths for RRL have been extracted from figure 8 showing the backup path distributions for two RRL alternatives when introducing link failures. We have used link failures due to pcycles poor node failure handling.The RRL alternative denoted min uses the original algorithm presented in Sec. 3.2, while the RRL alternative denoted 5 generates about two more layers than the original. Vaules for pcycle performance have been extracted from results in [7]. The pcycle method denoted large attempts to build few large cycles, and the method denoted small attempts to make many small cycles to optimize the backup path length. Statistics on RT backup path lengths have been collected from [4]. Note that those numbers have been obtained based on a single chordal ring topology.

The most important observations from table 1 is that the number of recovery abstractions, i.e., layers, is very modest for RRL compared to pcycles with many small cycles and RT. Pcycles with a few large cycles, also provides few abstraction, however that comes with the expense of very long backup paths. The number of abstractions for RT stems from the fact that redundant trees must be built for every candidate egress node. If not all 16 nodes will be egress nodes, the number of abstractions will decrease.

When it comes to the additional state information imposed by the schemes, RRL requires at most fixed additional state proportional to the number of layers, while pcycles requires at most additional state proportional to the number of pcycles. Redundant Trees, on the other hand, builds subtopologies that implicitly gives the routing, and the state increase is constantly two.

For backup path lengths we observe that RRL performs well, and also that the lengths can be decreased if the number of layers is increased.

None of the schemes affect the routing in the failure free situation.

When it comes to coverage of the schemes, both RT and RRL easily cover both link failures and node failures. P-cycles is not designed for resisting node failures, however there exist some very inefficient ways of obtaining node fault-tolerance [24].

From [4] and [24] we find no obvious way RT and pcycles can be applied to global recovery as has been described for RRL.

For completeness, we mention that practical implementations may be compared using other criteria, e.g., failure detection time and notification time. These criteria depend however on implementation rather than the recovery scheme.

To sum up we would like to point out that RRL seems to give few simple and intuitive recovery abstractions of the network. Coupled with acceptable backup path lengths, flexibility on global and local recovery, modest state increase, and coverage of both node and link failures, RRL provides a good alternative for practical handling of recovery in packet networks.

Table 1. Properties of the different recovery schemes

Property	**Pcycle small**	**Pcycle large**	**RT**	**RRL min**	**RRL 5**
#Abstractions	12.8	2.2	16	3.13	5
Global recovery	no	no	no	yes	yes
Local recovery	yes	yes	yes	yes	yes
Alg. Complexity	$\mathcal{O}(\|V\|^2 log(\|V\|))$	$\mathcal{O}(\|V\|^4 log^2(\|V\|))$	$\mathcal{O}(\|V\|^3)$	$\mathcal{O}(\|V\|^3)$	$\mathcal{O}(\|V\|^3)$
Path lengths	4.1	8.7	3	3.4	3.2
Additional state	12.8	2.2	2	3.13	5
Affect normal	no	no	no	no	no
Node recovery	poor	poor	yes	yes	yes
Link recovery	yes	yes	yes	yes	yes

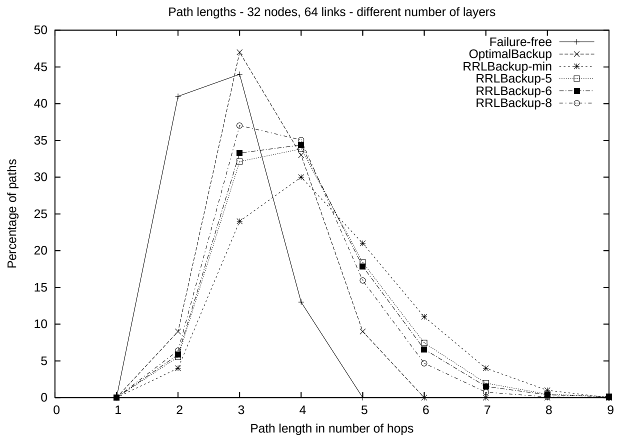

Figure 7. Distributions of path lengths when introducing one node failure in the original path.

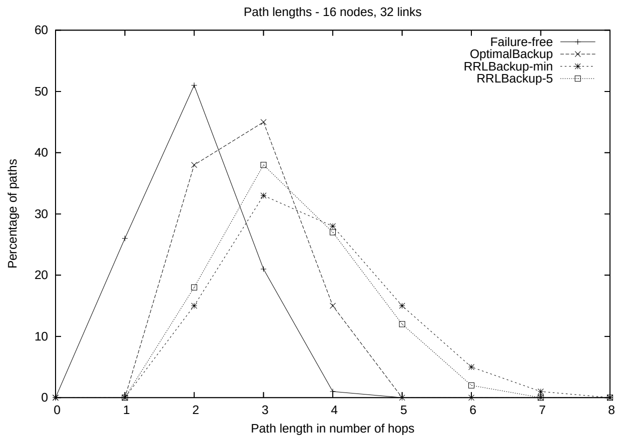

Figure 8. Distributions of path lengths when introducing one link failure in the original path.

5 Implementation issues in MPLS

Multiprotocol label switching (MPLS) is much used as a framework for network recovery, and traffic engineering in general. In this section we demonstrate the applicability of RRL for network protection within the MPLS framework. We stress that our method is by no means bound to MPLS, and may be applied to connectionless IP, Ethernet and other technologies as well.

MPLS is a connection-oriented technology and adhere therfore to the connection-oriented implemention requirements discussed in Sec. 3.1.2. As shown in Sec. 4, in most cases as few as four layers suffice to cover relatively large networks. A network with four safe layers will calculate the LSPs for five topologies, the fifth topology being the original, error-free one.

Another requirement imposed by RRL is that the node carrying out the recovery action must know what LSPs are affected by the failed node or link. Then, only the traffic from affected LSPs will be moved to backup LSPs. For local recovery, the node detecting the failure is obviously aware of what LSPs are originally passing through the failed link or node. However, for global recovery, the ingress node is moving traffic to backup LSPs. The ingress node can be informed about all nodes of an LSP using, e.g., Record Route Object in RSVP ([2]).

5.1 MPLS Algorithms

LSPs for the layers can be created in several ways. A simple approach is, for each layer, to set up an LSP from each node to each candidate egress node, i.e., to each node that could possibly be used as egress for a network path. This approach is simple and provides protection also for later dynamically created LSPs. In addition, it allows any node in the network to switch layer. However, it would likely produce unnecessarily many protection LSPs.

The second alternative is to operate similarly to the standard MPLS operation mode, and provide protection for the existing MPLS paths. In the following we present an algorithm for calculating both global and local protection LSPs. In a network topology $G = (V, E)$, a set of primary LSPs $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$, and a set of safe layers $\mathcal{L}$ are given. Let each p_i be represented by the ordered set $\{n_i^1, n_i^2, \ldots, n_i^{m_i}\}$, where m_i is the length of LSP p_i. The algorithm below creates global backup LSPs g_i^l in each layer $l \in \mathcal{L}$ to backup a particular primary LSP p_i. In addition, it creates local backup LSPs q_i^j from a particular point of repair n_i^j in the primary LSP p_i.

```
foreach $p_i \in \mathcal{P}$
    foreach $L \in \mathcal{L}$
(1)    create LSP $g_i^l = \mathrm{sp}(n_i^1, n_i^{m_i}, L)$
    endfor
    for $j = 2$ to $m_i - 2$
(2)    create LSP $q_i^j = \mathrm{sp}(n_i^j, n_i^{m_i}, L(n_i^{j+1}))$
    endfor
(3) if $(\mathrm{link}(n_i^{m_i-1}, n_i^{m_i}) \in L(n_i^{m_i}))$
(4)     if $(\mathrm{link}(n_i^{m_i-1}, n_i^{m_i}) \in L(n_i^{m_i-1}))$
            choose neighbor $n'$ of $n_i^{m_i-1}$ so that $n' \neq n_i^{m_i}$
            create LSP $q_i^{m_i-1} = n_i^{m_i-1} \vdash \mathrm{sp}(n', n_i^{m_i}, L(n_i^{m_i-1}))$
        else
(5)         create LSP $q_i^{m_i-1} = \mathrm{sp}(n_i^{m_i-1}, n_i^{m_i}, L(n_i^{m_i-1}))$
        endif
    else
(6)     create LSP $q_i^{m_i-1} = \mathrm{sp}(n_i^{m_i-1}, n_i^{m_i}, L(n_i^{m_i})$
    endif
endfor
```

Here, $L(n)$ is the safe layer of node n in the topology G, and $\mathrm{link}(n_i^j, n_i^{j+1})$ is the link connecting nodes n_i^j and n_i^{j+1} in path p_i. $\mathrm{sp}(a, b, G)$ calculates the shortest path from from a to b in the topology G. Line (1) creates an end-to-end (global) backup LSP for g_i in each layer l by finding the shortest path between the end points. Line (2) creates backup LSPs for local recovery. We create a backup LSP from the detecting node (n^j) in the safe layer of the upstream node (n^{j+1}). The ingress node has been handled in line (2), and the special case with link failure for a link connected to an egress node is handled in lines (3) to (6). Line (3) tests whether there exist a link to the egress node that is included in the safe layer of the egress node. If so, we have the two alternatives from Sec. 3. Line (4) covers the case where the link is also included in the safe layer of the detecting node (the upstream node). In that case we create an LSP from the detecting node $n_i^{m_i-1}$, but make sure that we use a deflection node n' as forced next hop. Line (5) covers the case where we are allowed to use the safe layer of the detecting node without deflection. In line (6) we cover the case where the link is not in the safe layer of node $n_i^{m_i}$.

Either the node detecting the failure or the ingress node of the primary LSP has to move traffic from original LSPs to backup LSPs in the case of a failure.

If two node and link disjoint paths exist, MPLS global recovery, as specified in [19], requires one backup LSP per primary LSP. For local MPLS recovery, each primary LSP requires n-1 backup LSPs where n is the number of nodes in the LSP [19].

When comparing this with the RRL MPLS specification, we get that both methods require $\sum_{i=1}^{|\mathcal{P}|}(m_i - 2)$ backup LSPs for local recovery. For global recovery, standard MPLS requires $|\mathcal{P}|$, while RRL MPLS requires $|\mathcal{P}| \cdot |\mathcal{L}|$ backup LSPs.

Some MPLS applications advise the local backup path to merge with the primary path as soon as possible after bypassing the failure, even if its not the shortest path. That is because the primary path is configured to fulfill all the service requirements of the traffic, which need not be the case for the backup path. RRL can also support such an approach. The local node detecting the failure establishes a backup LSP according to the safe layer of the failed node or link. This backup LSP will end in the first point in which it merges with the primary LSP. In case of failure, the detecting node then moves the traffic to the correct layer and stacks the corresponding header to the packets. This header will be popped at the merge point and the packet routed according to the full topology from there.

6 Conclusion and future work

In this paper we have presented a novel method for handling recovery in packet networks that we call Resilient Routing Layers (RRL). RRL is based on building simple and intuitive recovery abstractions of the network that offer many choices of different optimization criteria.

Even though RRL is supported by algorithms, we have demonstrated that it is sufficiently simple even for a network engineer to implement it by hand in relatively short time for reasonably sized networks.

Compared to other recovery schemes, RRL seems to be a good candidate for handling recovery in packet networks. Even for the two parameters for which RRL arguably should have the most problems in competing, namely path lengths and additional state information, we have shown that it performs comparably to other methods. We have also demonstrated that RRL is scalable, even very large networks can be covered by 5 layers. In addition, it has been shown how RRL is a good foundation for easy creation of MPLS backup LSPs.

Future work regarding RRL will cover optimizations for handling link failures and optimizations for fast local protection in connectionless networks like IP and Ethernet. As part of our future work, we will also explore network traffic after failures, packet reordering, packet loss, load balancing, and recovery differentiation.

References

[1] http://www.cs.washington.edu/research/networking/rocketfuel/.

[2] D. Awduche et al. RSVP-TE: Extensions to RSVP for LSP tunnels. In *IETF*, RFC 3209, Dec. 2001.

[3] R. Bartos and M. Raman. A scheme for fast restoration in MPLS networks. In *Proc. of the Twelfth IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS)*, pages 488–493, 2000.

[4] R. Bartos and M. Raman. A heuristic approach to service restoration in MPLS networks. In *Proc. ICC*, pages 117–121, June 2001.

[5] A. Bremler-Barr, Y. Afek, H. Kaplan, E. Cohen, and M. Merritt. Restoration by path concatenation: Fast recovery of MPLS paths. In *Proc. ACM Symposium on Principles of Distributed Computing*, pages 43–52, 2001.

[6] T. Bu and D. Towsley. On distinguishing between internet power law topology generators. In *IEEE INFOCOM*, pages 638–647, New York, June 2002.

[7] T. Cicic, A. Kvalbein, A. F. Hansen, and S. Gjessing. Resilient Routing Layers and p-Cycles: Tradeoffs in Network Fault Tolerance. In *to appear at High Performance Switching and Routing (HPSR2005)*, Hong Kong, May 2005.

[8] W. D. Grover and D. Stamatelakis. Cycle-oriented distributed preconfiguration: Ring-like speed with mesh-like capacity for self-planning network restoration. In *Proc. ICC*, volume 1, pages 537–543, June 1998.

[9] W. D. Grover and D. Stamatelakis. Self-organizing closed path configuration of restoration capacity in broadband mesh transport networks. In *Proc. CCBR'98*, 1998.

[10] H. Huang and J. Copeland. A series of Hamiltonian cycle-based solutions to provide simple and scalable mesh optical network resilience. *IEEE Communications Magazine*, 40:46 –51, Nov. 2002.

[11] A. Itai and M. Rodeh. The multi-tree approach to reliability in distributed networks. *Inform. Computation*, 79:43–59, 1988.

[12] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian. Delayed Internet routing convergence. *IEEE/ACM Transactions on Networking*, 9(3):293–306, June 2001.

[13] C. Labovitz et al. Origins of Internet routing instability. In *Proceedings of IEEE/INFOCOM*, Mar. 1999.

[14] M. H. Macgregor and W. Groover. Optimized k-shortest-paths algorithm for facility restoration. *Software-practice and experience*, 24(9):823–834, 1994.

[15] M. Medard, S. G. Finn, and R. A. Barry. Redundant trees for preplanned recovery in arbitrary vertex-redundant or edge-redundant graphs. *IEEE/ACM Transactions on Networking*, 7(5):641–652, Oct. 1999.

[16] A. Medina, A. Lakhina, I. Matta, and J. Byers. BRITE: An approach to universal topology generation. In *IEEE MASCOTS*, pages 346–353, Aug. 2001.

[17] K. Menger. Zur allgemeinen kurventheorie. *Fund. Math.*, 10:95–115, 1927.

[18] F. Otel. On fast computing bypass tunnel routes in MPLS-based local restoration. In *Proceedings of 5th IEEE International Conference on High Speed Networks and Multimedia Communications*, pages 234–238, Jeju, Korea, 2002.

[19] P. Pan et al. Fast reroute extensions to RSVP-TE for LSP tunnels. In *IETF*, Internet Draft, Aug. 2004.

[20] E. Rosen et al. Multiprotocol label switching architecture. In *IETF*, RFC 3031, Jan. 2001.

[21] A. Sack and W. D. Grover. Hamiltonian p-cycles for fiber-level protection in homogeneous and semi-homogeneous optical networks. *IEEE Network*, 18:49–56, Mar. 2004.

[22] V. Sharma and F. Hellstrand. Framework for multi-protocol label switching (MPLS)-based recovery. In *IETF*, RFC 3469, Feb. 2003.

[23] D. Stamatelakis and W. D. Grover. Rapid span or node restoration in IP networks using virtual protection cycles. In *Proc. CCBR'99*, Ottawa, Nov. 1999.

[24] D. Stamatelakis and W. D. Grover. IP layer restoration and network planning based on virtual protection cycles. *IEEE Journal on selected areas in communications*, 18(10), Oct. 2000.

[25] J. W. Suurballe. Disjoint paths in a network. *Networks*, pages 125–145, 1974.

[26] J. W. Suurballe and R. E. Tarjan. A quick method for finding shortest pairs of disjoint paths. *Networks*, 14:325–336, 1984.

[27] 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)*, 2003.

[28] B. M. Waxman. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*, 6(9):1617–1622, Dec. 1988.

[29] M. A. Weiss. *Data Structures and Algorithm Analysis*. Benjamin/Cummings, 1992.

[30] J. S. Whalen and J. Kenney. Finding maximal link disjoint paths in a multigraph. In *Proc. IEEE GLOBECOM '90*, 1990.

[31] G. Xue et al. Delay reduction in redundant trees for preplanned protection against single link/node failure in 2-connected graphs. In *IEEE GLOBECOM2002: (Optical Networking Symposium)*, pages 2691–2695, 2002.