Using Multiple Topologies for IP-only Protection Against Network Failures: A Routing Performance Perspective

George Apostolopoulos
ICS-FORTH, Greece
georgeap@ics.forth.gr

***Abstract*— IP networks have to carry increasingly more mission critical traffic where outages even of very short duration are undesirable or in some cases unacceptable. Traffic disruptions occur when the topology of the network changes either due to failures or reconfiguration. In connectionless networks current methods for protecting traffic from network failures rely on tunneling mechanisms for directing traffic around the failed resources and forwarding it to its destination avoiding loops. Recently, IP-only traffic recovery has been explored to avoid the high configuration complexity of the explicit tunneling approaches. In this work we present mechanisms for achieving recovery from failures of a single link or node using only IP forwarding mechanisms and an emerging feature of intra-domain routing, multi-topology forwarding. Using the multi-topology approach we build a small number of *recovery* topologies that are used to route traffic when failures occur. Through detailed and realistic performance evaluation we show that the multi-topology based approach has low computational, storage and forwarding overhead. In particular we focus on how traffic is routed after a failure and we present a heuristic method for optimizing the routing performance of multi-topology recovery. We show that this approach outperforms other IP based recovery methods and its performance can be close to that of an unrealistic optimal failure routing policy as well as to that of a computationally expensive brute force approach which explicitly optimizes the link weights of each recovery topology.**

I. Introduction

IP networks have to carry increasingly more mission critical traffic where outages even of very short duration can be unacceptable for certain applications. Traffic disruptions occur when the topology of the network changes either due to some resource (link, node, interface) changing status (becoming unavailable or becoming available) or due to administrative changes (change of link metrics in an IGP) or change of area configuration, route summarization and so on. When such changes occur, all routers have to recompute their routing tables and update their forwarding tables i.e. *converge* to a new set of least cost paths for the new topology after the failure. We will call this *SPF repair*. During this convergence period traffic can be disrupted. Traffic disruptions include traffic being dropped (for example the router adjacent to a failing link will drop incoming traffic for this link until it converges to new next hops) or traffic being looped temporarily. Because of their short lived nature these forwarding loops are called micro-loops.

Recently, there have been proposals in the Internet Engineering Task Force (IETF) for extending IP forwarding to perform in a "multi-topology" fashion [1], [2]. Routers now can deal with multiple physical or logical topologies, i.e. maintain multiple sets of next hops for each destination. Depending on the network design, two topologies can have disjoint address spaces, or overlapping address spaces with packets carrying information that allow routers to determine which topology to forward the packet on. The main motivation for the multi-topology work in IETF is the co-existence of IPv4 and IPv6 networks and topologies. Other uses are also envisioned most importantly in association with QoS routing, where packets are mapped to different topologies based on the type of QoS they need. In this work we will focus on using multiple topologies for repairing traffic when network resources fail. The network is partitioned into a set of *recovery* topologies and routers compute shortest paths and next hops for each of them. When a failure occurs, routers adjacent to the failed resource can redirect traffic to one of the recovery topologies and mark it with information that will ensure that repaired packets will follow the recovery topology all the way to their destination. Through the use of recovery topologies routers can repair of traffic immediately after they detect a resource failure with relatively little configuration and implementation complexity.

Still, the most important advantage of multi-topology repair is that it offers more flexibility for routing traffic after a failure. Not only more resources can be available to the repaired traffic but we have unprecedented level of control over how traffic will be routed over these topologies. Routes are computed independently for each recovery topology based on the link weight settings for this topology. In the SPF repair case routing of traffic after a failure is performed (at least for some time) using the same link weights that were in effect before the failure. In multi-topology routing we are free to optimize the weight settings on each recovery topology without compromising the no failure routing. It may even be possible to approach the routing performance of an ideal *best-case* failure routing, where one instantaneously optimizes and applies new link weights for any failure that occurs in the network. After pre-computing an appropriate set of recovery topologies and link weights on them it is possible to route traffic into a topology that is optimized for the particular

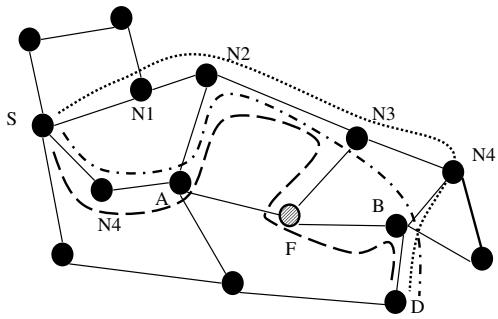


Fig. 1. Examples of various types of recovery

failure.

In this work, we explore how to use this multi-topology approach for providing recovery in case of network resource failures or network reconfigurations. We focus on the routing performance after link failures and we study the best case performance that can be achieved by multi-topology repair and how it compares with an idealized optimal SPF repair method. We then present heuristics for determining recovery topologies and link weight settings on them so that we can achieve routing performance that approaches the best case. These heuristics can result in configurations with a small number of recovery topologies (10 in this work) that outperforms recently proposed IP only repair approaches as well as SPF repair in a variety of realistic topologies and traffic matrices. At the same time we show that the resource requirements for multi-topology routing, most notably the number of recovery topologies and the additional memory in the forwarding component of routers are rather realistic.

This report is structured as follows: In Section II we provide some brief background on the traffic protection problem and in III we present more details on the problem of IP-only recovery of traffic. In Section IV we present the details of the proposed scheme, we discuss its algorithmic complexity, and present a heuristic algorithm for computing recovery topologies. In Section VI we discuss further details of our scheme and its implementation. In Section VII we evaluate the performance of the proposed solution in a variety of network topology and traffic conditions. In Section IX we review related work on the problem and finally in Section X we summarize our findings and discuss potential future work.

II. Overview of the Problem

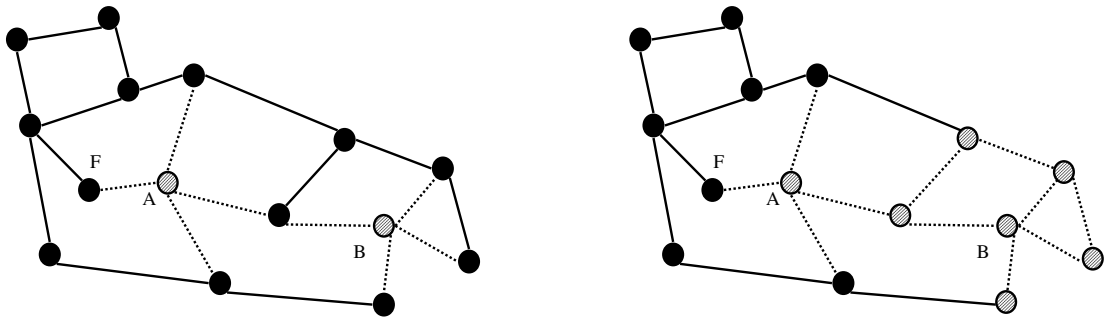
The problem of providing recovery of traffic from network resource failures has been studied extensively and some of the most important contributions will be reviewed in Section IX. Typically solutions involve re-routing of traffic into an alternate set of network resources. The *global* recovery approaches require the source of the traffic to *repair* traffic and can be very slow to react since it will be some time until the source detects the failure. *Local* recovery methods usually pre-compute backup resources that can be activated quickly when the corresponding outgoing link (or next hop-router) has failed. In the example of Figure 1 for traffic between node S and node D, in case node F, or link A-F fails, the global repair would use the path S-N1-N2-N3-N4-B-D, while with local repair the traffic would have to reach router A first and would follow the path S-N4-A-N2-N3-N4-B-D. Since traffic can be repaired immediately when a failure is detected the reaction time of local methods is better. In this work we will focus on local recovery.

In connectionless networks packets always follow a lest cost path to their destination and one needs some external mechanism to force packets to follow an alternate path to their destination in case of failures until new least cost paths that take into account the failure are recomputed. Usually MPLS or IP tunneling are used to setup *tunnels* for this purpose. Since these tunnels are point-to-point it is not practical to setup a backup tunnel for each possible destination of the repaired traffic. Thus the backup path usually terminates at the node behind the failed node as shown in the example of Figure 1 where the repair for either a failure of link A-F or router F is repaired by the tunnel A-N2-N3-N4-B. This is usually referred to as link rerouting and may result in sub-optimal routing of the repaired traffic since it will have to be delivered to the next-next hop even if there is a better path to its destination.

In this work we propose an IP only recovery mechanism that is based on the concept of multi-topology routing. We assume that routers run an IGP with appropriate extensions that allows them to maintain information about multiple coexisting topologies and compute shortest paths in each of them. Routers will compute a number of *recovery topologies*. These are subsets of the real network topology with some resources (nodes and links) missing. We say that these resources are *covered* by the respective recovery topology. When a router detects a failure of one of its adjacent links it repairs traffic by sending it to the recovery topology that covers the failed resources. This mechanism is very general: it can handle a single link, node failure or a failure of a group of network elements. Unlike some other approaches it is applicable to any network independent on its link weights and it does not need to differentiate between node and link failures. The recovery is triggered when a node discovers that the link it should be using for forwarding traffic has failed.

III. Routing on recovery topologies

The brute force approach would be to create one recovery topology for each possible failure by removing the failed resource from the base network topology. This approach clearly requires excessive amounts of resources since each router will have to perform an SPF computation for each possible recovery topology and maintain forwarding state for it. To make the multi-topology approach feasible for traffic recovery, we have to reduce both the computational and the storage requirements by reducing the number of recovery topologies.



(a) recovery topology (b) a node that is disconnected in its recovery topology

Fig. 2. Example of a recovery topology

The only way to achieve this is to create topologies that protect from failures of multiple network resources.

Consider for example the topology shown in Figure 2(a) that is generated from the base network topology by removing nodes A and B and their links. This recovery topology can protect from failures of nodes A, B or any of their adjacent links. When a failure of any of these resources is detected the detecting router will repair affected traffic by routing it over this recovery topology. Thus a recovery topology must satisfy the following properties: (a) All nodes in the network must be reachable over this topology. This means that the topology that is left after we remove the covered resources is connected. Note that this includes even the nodes covered by the topology. In the recovery topology of the example both nodes A and B are covered, but in case e.g. A fails, B should be still reachable over the recovery topology. (b) Traffic in the recovery topology should not be forwarded through covered nodes. This is necessary for correct repair of traffic. When node A fails and traffic is repaired into the recovery topology, we expect that the packets will not have to cross A again in the recovery topology. One needs to be careful though. Consider the example of cases like the one shown in Figure 2(b) where we can not reach node B on the recovery topology since all its communication will have to cross nodes that are also in the same recovery topology. Thus we need to introduce a third rule: (c) Each node that is covered in a recovery topology must have at least one neighbor that is not covered in this topology.

Nodes can use an SPF computation to determine shortest paths over the recovery topologies. These paths should allow traffic to reach nodes that are covered in each recovery topology but not to transit through them. Such paths are straightforward to compute using a modified Dijkstra shortest path algorithm. The algorithm operates as usual with the exception that we never extend paths that have reached a covered node. This will ensure traffic will not use covered nodes as transit while at the same time allowing covered nodes to be reachable over all their adjacent links. Also, if the router that performs the SPF computation is a covered router then we allow it to compute paths without restrictions, as a result a covered router will be able to use all its adjacent links for sending traffic into the recovery topology. This approach is both simple and effective. Covered nodes can use all their adjacent links to source and sink repaired traffic allowing for more efficient routing in cases of failures.

If routers compute shortest paths for each recovery topology according to the above rules, then we are guaranteed that all possible single link or node failures can be repaired. Indeed, consider two scenarios, one where link F-A fails and one where router A fails. If link F-A fails, all traffic that does not cross the link is delivered using the primary topology. Consider traffic that crosses link F-A, say at router F. Router F will find the link F-A down and will assume that node A has failed. Then it will push the traffic into the recovery topology that covers A. By construction traffic can reach all destinations over this topology. Consider the special case where the destination of traffic is router A. In this case we can perform a simple form of deflection routing. We use the topology where F is protected and we send the packet in one of F's next hops in this topology. Since F is protected in this topology we are guaranteed that the packet will not return to F and consequently it will be delivered to A without routing loops. The neighbor of F that will receive the packet will be able to reach A over F's recovery topology and deliver the packet. Even if both A and F are protected by the same recovery topology F will be able to reach A since there is at least one more path that connects F and A in addition to the F-A link.

Now consider the case where node A fails. Like before, F will detect that link F-A is down and will send traffic over A's recovery topology. If the destination of traffic is some node other than A then again traffic will be delivered to it over the recovery topology. If the destination of traffic is node A then, like above, F will send traffic over its own recovery topology. Traffic will be forwarded towards A over this topology but it

will have to cross another failed link just before it reaches A (since A has failed, all its links have failed too). The router at this point (say router C), will be able to drop the packet since the packet has been received from a recovery topology and there is another failure while forwarding.

We see that the above recovery mechanism does not need to distinguish between node and link failures. Repair is triggered by a router that detects a failure of one of its local links. Traffic will be repaired correctly even when the link failure is a result of the failure of the node at the remote end of the link. Not having to distinguish between node and link failures is very desirable since in general it is rather difficult to determine in a timely manner if a node has failed. In these cases a keep-alive protocol is usually deployed that probes the status of remote nodes. When a node fails some time is lost until the keep-alive protocol detects the failure. On the other hand a link failure in many cases results in a change of the status of the link that is immediately communicated to the nodes adjacent to it.

A. Packet forwarding in multiple-topologies

The above routing mechanism is simple to implement. Each packet will need to contain an identifier that shows the topology it is currently being routed on. Routers will have to maintain a separate forwarding table for each topology (we will discuss in detail how to do this in Section VI-B). The forwarding table will contain a primary and a backup next hop for each destination and a backup topology. The backup next hop will be used if during the forwarding of a packet a router will find that the link to the primary next hop is down. Then the packet will be sent to the backup next hop and potentially the packet will be put in a different topology by modifying its topology identifier. An outline of the forwarding algorithm can be found in Algorithm 1.

As long as the appropriate primary and backup next hops and topologies are computed and downloaded to the lookup table used for forwarding packets (usually called the Forwarding Information Base or FIB) a node can forward and repair traffic without having to perform any additional complex checks in the forwarding path. Each node maintains a FIB for each of the recovery topologies, the base topology (we will call this the $NORMAL$ topology) and a special topology called $DOUBLE\_FAILURE$. For each destination $dest \in N$ and topology $t \in T$ a node maintains primary next hops $pri\_nhop(dest, t)$, the backup next hops $backup\_nhop(dest, t)$ and the backup topology $backup\_top(dest, t)$. The values of $pri\_nhop(dest, t)$ are determined by the the modified Dijkstra least cost path computation that we have described above. The values of the $pri\_nhop(dest, NORMAL)$ are determined by the regular least cost path computation in the base topology, while the values of $pri\_nhop(dest, DOUBLE\_FAILURE)$ are set to $pri\_nhop(dest, NORMAL)$. In general $pri\_nhop(dest, t)$ will be a set of next hops since there may be multiple equal cost multi-paths (ECMP) to the destination. Usually traffic is load balanced along these multiple ECMP nhops. When there are failures it is still possible to forward traffic to its destination through one of the members of $pri\_nhop(dest, t)$. Since we consider only single link or node failures, as long as there are more than two members in $pri\_nhop(dest, t)$ we can reach the destination without the need for repair. Thus repair is needed only when we have only a single nhop to a particular destination. In order to ensure repair of traffic, for these destinations we need to compute backup next hops and topologies. So for all destinations $dest$ where $pri\_nhop(dest, NORMAL)$ contains a single node, say $nhop_p$, the backup topology will be the recovery topology that covers node $nhop_p$ (we denote this as $T(node_p)$) and the backup next hop will be set to $pri\_nhop(dest, T(node_p))$. Similarly the backup topology $backup\_top(dest, NORMAL)$ will be set to $T(node_p)$. This will compute backup next hops for the FIB that corresponds to the NORMAL topology.

Now consider how to compute backup next hops for all the $pri\_nhop(dest, t)$ that contain a single next hop, where t is a recovery topology. The FIB entries are used to forward packets that arrive on a recovery topology. When a node receives a packet on a recovery topology it knows that there has been a failure in the network but it does not know the nature of the failure. It could be either a node failure that will result in all its adjacent links failing or there can indeed be more than one independent failures of nodes or links. In both cases we set the backup next hop to $pri\_nhop(dest, NORMAL)$ and the backup topology is set to $DOUBLE\_FAILURE$. Thus if a node forwards a packet on a recovery topology and discovers a second failure it will push the packet into the $DOUBLE\_FAILURE$ topology. As we discussed above, the $pri\_nhop(dest, DOUBLE\_FAILURE)$ are set to $pri\_nhop(dest, NORMAL)$ and $backup\_nhop(dest, DOUBLE\_FAILURE)$ and $backup\_top(dest, DOUBLE\_FAILURE)$ are set to NULL. This ensures that packets sent over the $DOUBLE\_FAILURE$ topology follow the normal topology but are not repaired. If another failure is encountered while on this topology the packet will be dropped. In essence we fall back to the forwarding we would have done if there was no repair mechanism deployed.

Algorithm 1 Packet forwarding algorithm

```
nhop = find_primary_nhop(dest, top)
if nhop is up then
  forward_packet(nhop, top)
else
  (nhop, new_top) = find_backup_nhop(dest, top)
  if nhop == NULL then
    drop packet
  else if nhop is down then
    drop packet
  else
    forward_packet(nhop, new_top)
  end if
end if
```

To see how this arrangement handles a single node failure

consider the example of Figure 1 and a packet destined to node F and assume that F has failed. When the packet reaches node A it will be put into the recovery topology that covers F. Over this topology it will be forwarded until it reaches say node N3. This node will not be able to forward the packet further and will put it into the $DOUBLE\_FAILURE$ topology. Since the next hop towards F at node N3 will be down the packet will be dropped. The packet is eventually dropped but after is has traveled from A to N3 over the recovery topology. This wastes resources but simplifies forwarding since we do not need to differentiate between link and node failures. We will revisit the topic of multiple failures in Section VIII. Algorithm 2 summarizes how a node is setting up its backup next hop and topology information for all the FIB entries.

Algorithm 2 Algorithm to compute the FIB entries for all topologies

for each $dest \in N$ **do**
  if $pri\_nhop(dest, NORMAL) = 1$ **then**
    $node_p = pri\_nhop(dest, NORMAL)$
    $backup\_nhop(dest, NORMAL) = pri\_nhop(dest, T(node_p))$
    $backup\_top(dest, NORMAL) = T(node_p)$
  end if
end for
for each $dest \in N$ **do**
  $pri\_nhop(dest, DOUBLE\_FAILURE) = pri\_nhop(dest, NORMAL)$
  $backup\_nhop(dest, DOUBLE\_FAILURE) = NULL$
end for
for each $t \in T, t \neq NORMAL, t \neq DOUBLE\_FAILURE$ **do**
  for each $dest \in N$ **do**
    $backup\_nhop(dest, t) = pri\_nhop(dest, NORMAL)$
    $backup\_top(dest, t) = DOUBLE\_FAILURE$
  end for
end for

IV. Determining the recovery topologies

As we showed in the previous section, it is possible to use the same recovery topology to recover from failures of different network elements as long as they do not fail simultaneously. Since each recovery topology consumes forwarding resources and requires a separate SPF computation we would like to have a small number of different recovery topologies. So the question is how we can determine few recovery topologies that will achieve efficient routing of traffic in case of failures.

We assume that we have networks that have node and link connectivity of 2, i.e. they do not become disconnected due to the failure of a single node or link. Not all the topologies we used in our evaluation had this property, especially the PoP level topologies. We will discuss this further in Section VII. One way of formalizing the problem is the following: Given a graph with a set of nodes N and links L and with edges that have integer non-negative weights, compute sets of nodes $N_i, 0 \leq i \leq K$ so that $\forall_i N_i \subseteq N$, $N \backslash N_i$ is connected, and $\bigcup_{i=1}^{K} N_i = N$ so that K is minimized. It is not hard to see the similarities of this problem with the Minimum Set Cover problem (MSC), that is known to be NP-complete [4]. Recall that the MSC is formulated as follows: given a collection C of subsets of a finite set S and an integer K, determine if the collection contains a set cover of S with cardinality less than K, i.e., a subset C' of C so that each node in S belongs to at least one of the members of C'. If the set S is the set of nodes N in the graph, and the collection C consists of all possible protection topologies then we can immediately see how to map the problem at hand to the MSC.

Solving the above problem would allow us to discover the minimum possible number of recovery topologies. It attempts to minimize the amount of overhead that results from the extra recovery topologies but it does not necessarily perform well in terms of optimality of routing in case of failure. Indeed, when the overall number of recovery topologies is minimized, each recovery topology will cover multiple potential failures. Since nodes covered by a recovery topology can not be used when traffic is routed into the recovery topology the remaining resources will be overloaded. Ideally, we would want to determine a collection of recovery topologies that also minimizes the inefficiency of routing in case of failure as this is captured by the maximum link load for each recovery topology. Even if we have access to the traffic matrix and we could associate a "cost" with each of the potential recovery topologies, again computing a collection of topologies that would cover all nodes and had a minimum total cost would be as hard as the MSC problem. Indeed, in a special case of the problem where the cost is the same for all possible recovery topologies, the problem reduces to finding the minimum number of the recovery topologies, which is as we saw above the MSC problem.

V. Optimizing the routing of traffic after a failure

Recently there has been work on how to optimize the setting of link weights so as to achieve "good" traffic routing that minimizes a given objective function [11], [25], [21], [22] given knowledge of the traffic matrix. Also, recent work has made important contributions to the methods for measuring, estimating and using traffic matrices [13], [12], [10], [23]. While the basic problem of optimal routing can be formulated as a integer linear program, it becomes NP-hard when considering the constrains of IP routing, in particular the inability to split traffic non-uniformly across multiple next hops. As a result existing optimization proposals work by searching the state space of all possible link weight settings (usually within a minimum and maximum limit for the weight values) and relying on heuristics to reduce the state space and speed up the search. Some of the above work [11], [22], [25] focuses on optimizing the link weights so that the performance of the network is good even when single links fail using search based solutions similar to what has been used for the no

failures case. Solving the problem when link failures are considered is inherently more difficult. Now weight settings must be evaluated for their performance under *all* possible single link failures leading to multiple evaluations of the objective function, once for each link failure. As we will see, certain objective functions can be quite expensive to compute. Multiple heuristics for reducing the processing cost of these solutions have been proposed and will be reviewed in Section IX.

If we want to optimize the routing of traffic on each recovery topology we face a similar problem since we have to determine link weight settings that optimize some objective function under all possible single link failures. Only in our case the state space is dramatically larger. We have T recovery topologies so we have T times more link weights to optimize. Furthermore, the recovery topologies are not pre-determined. A complete solution should also compute the recovery topologies in addition to optimizing their link weights. It is not hard to see how a state space search approach would be unworkable in our case. Thus we face two problems. First, how to come up with an algorithm for generating recovery topologies and determining their link weights so that we achieve good routing performance and second, how to establish some base-line performance so as to be able to evaluate the performance of the above algorithm.

Lets discuss the second problem first. Clearly, the *optimal* SPF repair algorithm would be the one that for each possible single link failure could re-optimize the link weights so as to handle optimally the traffic after that failure. If we had a way to instantaneously compute such link weights and apply them to the network then we can be certain that we achieve the best possible routing for a given network and traffic matrix. Such a routing is not difficult to compute by simulating link failures and the routing of traffic after that. Still, this is not a fair bound for the multi-topology recovery method that we present here. This is because the multi-topology recovery assumes that when a link fails, it is the node at the remote end of the link that has failed. This assumption greatly simplifies the implementation of the recovery since we do not need to distinguish between link and node failures and allows us to handle node failures and link failures in the same uniform way. Nevertheless, this method of repair leaves less resources available for handling traffic after a link failure. While with SPF repair after convergence of the routing traffic can be routed in the original topology minus the failed link, in our case traffic can be routed in the topology that is left after we remove the remote end node and its adjacent links. Thus, we would expect to be less efficient than the optimal routing described above. In this work we will also characterize how much we loose in routing efficiency due to this implementation decision.

Because of the above, we use a different strategy for creating an empirical *best case* multi-topology repair solution. For this best case, we consider N recovery topologies, each covering only a single node. This allows each recovery topology to have as many resources as possible for handling traffic after a failure. Then for each of these N recovery topologies we optimize their link weights as follows: Consider recovery topology t_i that covers node n_i. This topology will handle traffic after any of the links adjacent to n_i fails. We set the link weights by searching through the state space formed by the link weights of topology t_i. For each weight setting we compute the worse case value of the objective function considering the failures of only the links adjacent to node n_i. This approach will be able to optimize the weight settings for a particular recovery topology given the current utilization of the network links. Unfortunately, this utilization depends on the weight settings on the other recovery topologies. Setting the link weights of a recovery topology to certain values may prevent us from discovering good routings on other recovery topologies. In other words the order that we consider the topologies t_i for optimization will have an effect on the final value of the objective function. Since we lack an algorithm for determining the best evaluation order we evaluate all topologies starting from t_0 until t_N and we repeat this round of optimization for a small number of times. Our experiments showed that after 3 rounds there was no more improvement in the quality of routing.

Although the above method is also a valid algorithm for determining a set of recovery topologies and their link weight settings it is not practical. In addition to its high computational cost (running time for 120 node topologies is more than a day on a Opteron 242 CPU), it is not realistic to assume N recovery topologies. As we saw in VI, the amount of bits available in the packet headers for storing the topology identifier is limited and we should attempt to find solutions that need 4-5 bits of information, i.e 16 to 32 recovery topologies. Thus we investigate a heuristic solution for computing a small number of recovery topologies and their link weight settings. We still perform a limited amount of search in the link weight state space but we use heuristics for guiding this search to areas that we could see the most improvement.

We focus on an objective function that attempts to limit the amount of link overload in the network. Previous works that addressed the link weight optimization problem for the no failure and the single link failure case [22], [21], [25], [11] have used various objective functions, such as maximum link load, overall network load, or link overload. In a real network the most important problem may be the overloading of some links since this will typically result in packet losses. Other metrics such a end-to-end delay through the network are also important measures of performance but in this work we will focus on link overload. [21] has introduced a link cost function that depends on the level of overload of the link. This cost function was derived through the operational experience with the AT&T network and has been used in later work in particular the work for link weight optimization. We will use the same link cost function here. Below we show how this cost function is computed if the capacity of link l is c_l and its current load is w_l.

$$cost(l) = \begin{cases} w_l/c_l & \text{for } 0 \leq w_l/c_l < 1/3 \\ 3 * w_l/c_l & \text{for } 1/3 \leq w_l/c_l < 2/3 \\ 10 * w_l/c_l & \text{for } 2/3 \leq w_l/c_l < 9/10 \\ 70 * w_l/c_l & \text{for } 9/10 \leq w_l/c_l < 1 \\ 500 * w_l/c_l & \text{for } 1 \leq w_l/c_l < 1.1 \\ 5000 * w_l/c_l & \text{for } 1.1 \leq w_l/c_l \end{cases}$$

Consider a network with a set of links L and a set of nodes N. For a given network, traffic matrix and set of link weights there will be a cost $cost(l)$ associated with each link l that depends on its load. The sum $total\_cost = \sum_{\forall l \in L} cost(l)$ gives us the total cost of a given routing. When there are failures we consider all possible failures (link failures in our case) and for each link we compute $cost_{max}(l)$ which is the maximum cost that we observe on this link. The maximum cost of a routing over all possible failures is $max\_cost = \sum_{\forall l \in L} cost_{max}(l)$. Our goal is to determine routings that minimize max_cost. We denote the cost of the optimal SPF repair as max_cost_{opt} and the best-case cost of multi-topology repair as $max\_cost^{mt}_{best}$.

In order to determine a good routing in case of failures we have to address two problems: (a) how to construct the recovery topologies, i.e. which nodes are covered by each topology, (b) how to set the link weights in each of them. In principle we should optimize them jointly. Our results showed that in certain cases the way the recovery topologies are constructed can have an effect on the quality of the routing that can be achieved. Although optimization of link weights on each recovery topology can boost routing performance in certain cases further improvements may not be allowed because of the structure of the recovery topologies. Joint optimization methods are currently under development and will be presented elsewhere.

For now, we first build the recovery topologies and then we optimize their link weights. When determining which nodes are covered by each recovery topology we attempt to ensure that nodes that their failure or the failure of their adjacent links has large impact on the network have enough resources available in the recovery topology they are covered. We measure the impact of the failure of each node or its adjacent links by removing the node and its adjacent links from the network and computing the load and cost of each link in the network. If the cost function of a link after the failure of node n is $cost_n(l)$ then the weight of node n is defined as $weight(n) = \sum_{\forall l \in L} cost_n(l)$. We determine the maximum number of topologies T we want to use and then we assign nodes to each topology in such a way so that we balance the total weight of each topology (which is the sum of the weights of all the nodes it covers). For each topology we maintain a sum of the weights of all nodes that are covered by it. We cover a new node in the topology that has the less overall weight among all topologies. This essentially will put few or only one high weight node in a topology but multiple low weight ones in another. Of course this equalization will be possible to the extent that node connectivity constraints are not violated. There will be cases where we will not be permitted to cover a node in a given recovery topology even if this would appear to be the post possible for it. Determining if a node can be protected in particular recovery topology requires that we ensure the topology remains connected after the node is removed: [5], [6] use this approach. The connectivity test can be done through a simple Breadth-First Search that has complexity $O(E+N)$ [7] and since we need to perform this for each candidate node the cost of determining a single recovery topology is $O(NE + N^2)$ and up to $O(N^2E + N^3)$ for all possible N recovery topologies. We also need to ensure that a node covered by a recovery topology does not have all its neighbors covered by the same topology.

Algorithm 3 Algorithm for optimizing link weights in recovery topologies

```
best_max_cost = INFINITY
reset blacklist
while TRUE do
  noprogress = 0
  find link l_max with the highest cost(l)
  n_1, n_2 are the nodes at the two ends of l_max
  t_1, t_2 are the topologies that cover n_1, n_2
  weight(l_max, t_1) = weight(l_max, t_1) + dw
  weight(l_max, t_2) = weight(l_max, t_2) + dw
  compute new max_cost
  if max_cost < best_max_cost then
    best_max_cost = max_cost
    reset the blacklist
  else if max_cost > best_max_cost then
    reset weight(l_max, t_1), weight(l_max, t_2) to orig values
    add l_max to the blacklist
    noprogress = noprogress + 1
  end if
  if noprogress > 5 then
    start over using a different initial l_max
  end if
end while
```

After the recovery topologies are computed we manipulate their link weights so that we reduce the overall cost across the network, i.e. we reduce the quantity max_cost. Our search space consists of all possible weight settings for each of the recovery topologies, i.e if we have L links and T topologies it contains $L * T$ weights, each link l has a different weight in each topology t and we represent this weight with $weight(l, t)$. These weights can be set independently and the settings of a link weight for a topology t will affect how traffic is routed over topology t when one of the resources covered by topology t fails. We need to explore the link weight state space and discover settings that achieve good cost. We do this using a gradient-descent type of search. In each state we perform a link weight modification that has a good chance of reducing the overall cost of the routing. Luckily, while computing the cost of a given state we can discover enough information to allow us to make a good guess which link weights should

be changed. In particular, we know the load on links and which failure causes the maximum overload on a link. We can attempt to reduce the cost of routing as follows: consider an overloaded link l that its highest load has been caused by the failure of e.g. a link l' between nodes n_1 and n_2. This overload has been caused by the additional traffic link l had to carry when link l' fails. Since the failure of this link l' was repaired as a failure of both n_1 and n_2 link l had to carry additional traffic in the topologies that cover n_1 and n_2 say t_1 and t_2 respectively. Thus we have a good chance of reducing its load if we increase its weight in both t_1 and t_2. This increase in weight should shed some traffic away from link l.

This cost reducing change can have three outcomes. The load on the link will be reduced so we will have a lower routing cost, the load of the link will be reduced but the load on some other link will increase resulting in a higher routing cost, or the load on the link will not change and neither will the cost. The latter case is possible if the overloaded link is an articulation link after the failure. In this case, independent of the setting of its weight it will have to carry all the network traffic between the two partitions of the network. Another reason that we may not be able to shed traffic away from the link is if the increase in its weight is too small to cause some traffic to move away.

Algorithm 3 summarizes the search procedure. The search through the link weight space for a good cost routing is based on the above cost reduction moves. In a given state s we find the most overloaded link and try to reduce its cost by increasing its weight in topologies t_1 and t_2 by a fixed amount dw. For each change we recompute the max_cost and our link load statistics. If we managed to reduce the cost we repeat with the new most loaded link and so on. Even if the cost remains the same we repeat the above step. If the link that we applied the weight change is still the most loaded, it will be picked again and its link weights will be incremented by dw again. Since this can not continue indefinitely we limit the maximum value of link weight we can reach. If after a change the routing cost becomes higher then we do not consider this link again and we put it in a small blacklist. This blacklist will be cleared when we manage to find a link that increasing its weight will result in an overall reduction of max_cost. Since we never make a non cost improving move we are not in danger of getting into a cycle. Furthermore, in order to avoid getting stuck in a local minimum if we have not been able to improve the cost after a small number of link weight changes (we use 5 in this work) then we repeat the search from the top level by picking a different overloaded link and attempting to reduce its cost. Determining of max_cost is rather expensive computationally since we will have to compute all routing tables and how traffic is routed after each possible link or node failure. Optimizations similar to the ones used in [11] could be used here, nevertheless the amount of steps we had to perform was rather small (less than 50 in all cases) so we could afford a less optimized implementation.

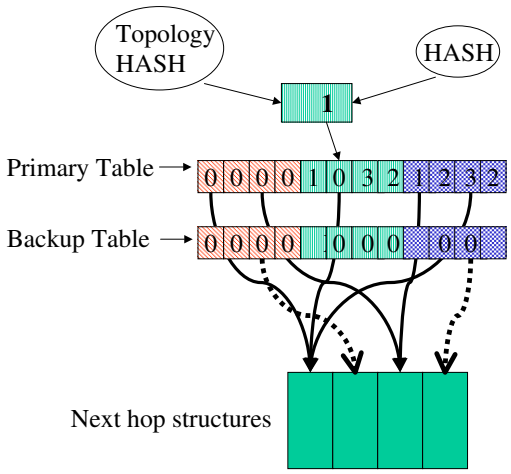


Fig. 3. Example organization of the FIB with multiple topologies

VI. Implementation Issues

A. Achieving forwarding of packets in different recovery topologies

For each incoming packet, routers have to determine to which topology it belongs. Also, a router should be able to send a packet into a particular topology. The topology information necessarily will have to be encoded by some fields in the packet. Alternatives include using some fields of the Layer 3 header, or information in the Layer 2 header (Ethertype of PPP information for example [16], [17], [2]) or some information that is added to the packet such as an MPLS shim header. One attractive possibility is to use the TOS/DSCP bits in the IP header [18] as a simple way to convey topology information. Indeed, the 6 DSCP bits would allow us to use 64 different topologies which are more than enough to provide efficient traffic recovery in all the networks we experimented with. Even 5 bits would be sufficient and if the other 3 bits available in the IP TOS field were used, we could still have 8 levels of priority for each topology. Using existing bits in the packet header has the advantage of not requiring the additional packet real estate and forwarding overhead for the inclusion of an MPLS label. In any case, forwarding of packets will have to be modified to take into consideration this information. This is not an unreasonable assumption. At least one major vendor [24] provides extensions in its forwarding plane for multi-topology forwarding where different topologies are used for IPv4 and IPv6 packets. Similar extensions can be conceived in the case that an MPLS label is used to convey topology information.

B. FIB organization and forwarding complexity

Routers today already maintain a number of next hops for each destination in their FIB for load balancing purposes.

Usually, a lookup structure is used to locate a longest prefix match to the destination address and the result points to an array of 8 or 16 next hop entries. A hash function based on information on the packet header determines which of the next hops will be used for forwarding the packet. If we want to handle multiple topologies, since all the topologies have the exact same set of destinations, there is no need to replicate the lookup structure. Actually this would be rather uneconomical since sometimes these lookup structures are optimized hardware systems based on Content Addressable Memories or other fast hardware with strict size limitations. Thus, the most efficient way to incorporate the additional topologies is by adding next hops to each FIB entry. The longest prefix match will be the same, but now the decision which next hop to use will also depend on the incoming topology of the packet and not only the load balancing hash. Given that multiple topologies will share the same next hops between them, it is not cost effective to allocate a separate fixed number of next hops per topology since the same information will be stored multiple times. On the other hand, we do not want each topology to have a variable number of next hops since this will make locating the correct next hop hard when forwarding a packet. A simple and efficient way to organize the next hop information is to use a table based lookup structure. Such a structure is shown in Figure 3 and its benefits for load balancing of traffic have been presented in [19]. Given that the table entries are much smaller than the size of the next hop structures (that need to maintain media specific information such as MAC addresses) we can allocate multiple table entries with each topology having its own dedicated set of table entries. All topologies can have the same number of table entries so that hashing is simplified. Multiple table entries can map to the same next hop and next hops are stored only once. An example is shown in Figure 3 where we assume 3 recovery topologies each allocated 4 table entries allowing up to 4 different next hops for each topology. Hashing to find the right next hop is very simple. The topology information of the packet is used to determine which topology will be used and will compute the topology index ($T, 0 \leq T \leq 2$). Then, the regular traffic hashing function will compute the next hop index ($H, 0 \leq H \leq 3$). Then the next hop to be used is pointed to by position $T * 4 + H$ of the table.

One important point here is that in general when forwarding a packet out of a next hop the router may need to update the topology information in the packet, for example when the packet has to be redirected to a different topology. If information about the new topology is stored in the next hop structure then the same physical next hop will have to be stored twice if it can lead to two different recovery topologies. Clearly, it makes more sense to store the outgoing topology information outside the next next hop structure and in particular in the table entries. Then, a next hop will have to be stored only once and the maximum number of next hops to be stored for each destination will be bounded by the number of physical active interfaces of the router. In the example of Figure 3 the numbers inside the table entries represent the outgoing topology.

The additional cost is the extra FIB storage needed for the additional next hops and the table entries. As we will see in the evaluation section the storage for the extra next hops is not excessive and the multi-topology requires the storing of roughly twice the number of next-hops when compared to the conventional case. The storage needed for the additional table entries is fixed and depends on maximum number of recovery topologies we need to support. For 16 topologies and 2 next hops for each on them we will need 32 table entries. Each table entry will have to encode the index of the next hop to be used for each topology and the outgoing topology information. 8 bits should be enough to encode both leading to a table size of 256 bits. Using the same amount for the backup table brings the total to 512 bits per FIB entry. This size is not unreasonably large since we expect it to be equal to the storage needed for 1-2 next hop structures.

C. Forwarding Operations

A router has to perform the following forwarding actions for an incoming packet: (a) forward to the same topology and (b) forward to a different topology. In the second case the router will have to modify the topology information contained in each packet. If the topology information is stored in the packet header the operation is simple since the packet size does not change, but in certain cases the header checksum may have to be recomputed (for example the IP header checksum has to be recomputed when the DSCP bits are modified). This may have an adverse effect on the forwarding performance but we should consider that this operation is not the common case, i.e. occurs only when a router reacts to a failure and re-routes traffic over the recovery topologies. Furthermore, even in this case, this operation will be performed only once for each packet, the packet is then forwarded to its destination over the recovery topology.

In cases where the topology information is carried into some extra field that is usually not a part of the IP packet (an MPLS label for example) routers will have to add and remove this information at appropriate places in the network. According to our discussion so far we would add the topology information only at the point where a router detects a failure of one of its adjacent links. This router will have to pay the penalty for pushing traffic into the recovery topology and its performance may suffer since it may have to deal with high data rates. Still, this is not the only available option. We can envision this recovery mechanism as a service to be offered in a the backbone of a provider (perhaps to only certain types of traffic). In this case, topology information can be added to each packet at the “perimeter” of the network, i.e. when the packet enters the provider’s network. Similarly, the information can be removed on exit of the provider’s network. Telling the routers when they should add/remove this information can be done relatively simply with some extensions to the IGP protocol that will allow each interface to be mapped as an entry/exit interface. On entry, the packet can be put in the default topology. Edge routers are optimized for this operation

Topology	Nodes	Links	Avg. degree	PoPs
ISP6461	113	357	6.3	22
ISP1221	54	106	3.9	40
ISP1755	60	135	4.5	23
ISP3257	93	270	5.8	49
ISP3967	63	133	4.2	22
ISP1239	199	941	7.98	29

Fig. 4. Basic characteristics of the topologies used in the experiments

and they have to perform it at the relatively low rates of incoming/outgoing traffic at the edge of the network.

D. IGP Extensions

Work on standardization of the extensions of the interior gateway protocols for supporting multi-topology forwarding has already started in IETF with work in IS-IS more mature [2] and work on OSPF having started recently [1]. The extensions described in these documents will be sufficient for implementing our scheme. In particular there is a mechanism for routers to indicate their capabilities in terms of supporting multi-topology forwarding that can be used to support cases where not all routers in the domain implement multi-topology forwarding. Furthermore, it is proposed that some topologies are reserved for special uses and use specific topology identifiers, for example topology 0 is reserved for the primary topology. Our scheme can easily respect the assignment of reserved topologies identifiers. One further concern for a realistic implementation is how to distribute the information that is needed for computing the recovery topologies. Unavoidably, these will have to be computed through an offline tool and communicated to the routers through some mechanism external to the routing protocol not unlike the cases where off-line methods are used e.g. for optimizing link weights.

VII. Performance Evaluation

We evaluate the performance of multi-topology traffic repair and compare it with that of SPF repair according to the following metrics:

- *Maximum cost* This is the quantity max_cost defined in Section V and represents the overall maximum load observed on all links when considering all possible link failures.
- *Provisioning overhead* For each link we measure its load $load(l)$ and we compute $load_{max}(l)$ which is the maximum load observed on a link under all possible failures. The maximum network load over all possible failures is $max\_load = \sum_{\forall l \in L} load_{max}(l)$. Considering the max_load of SPF repair as the base case we report the percent increase of the other methods over this value, e.g. $(max\_load_{mt} - max\_load_{spf})/max\_load_{spf}$.
- Increase in FIB storage due to the additional next hops. We report the average increase in the number of next hops for the whole network, that is the sum of the extra next hops over all nodes in the network divided by the sum of the primary next hops (i.e. the next hops required for the primary topology) over all the nodes in the network. Next hops are stored in the FIB as we described in Figure 3.
- We also show how path lengths are affected when there is a failure. The increase in the path length is a good indication of the increase in the delay of traffic through the network and routing inefficiency. For all possible failures we keep track of the average and the worse path length observed between each pair of nodes. Then we report the averages of these two values over all the pairs of nodes.

We compare the SPF repair where a failure is repaired when all nodes in the network re-converge in a new set of routes to the multi-topology and not-via approaches. For multi-topology routing we consider cases where we compute 5 and 10 recovery topologies in order to be able to observe the effects of the number of protection topologies on the quality of the routing. We also compare the performance of SPF repair and multi-topology repaid to that of the not-via recovery method based on IP tunneling. In this method each interface is assigned a second address that is to be used to send traffic to it if the node in the other side of the link has failed. Thus all routers compute shortest paths to the not-via addresses of the next next hop nodes and tunnel traffic to these addresses. In the example of Figure 1 in order to handle failures of node F, B would have a special address that would mean "deliver traffic not via F" and other routers in the network, including router A would compute a shortest path to this address that does not go through F, say A-N2-N3-N4-B-D. When node F or link A-F fails, A will put all traffic that would have to go through B into a tunnel with endpoint the not-via address of B. Note that both link and nodes failures are handled assuming that node F has failed.

A basic problem with performing realistic evaluation of routing performance is the difficulty in obtaining both realistic network topologies as well as traffic data. We partly resolve the problem of obtaining realistic network topologies by using the results of the Rocketfuel [9] project that was able to measure the topology and link weights of a number of real ISP topologies. The Rocketfuel topologies include link weights and latency information but no information about link capacities. Rocketfuel topologies are router level with routers organized into Points of Presence (PoPs). In Table VII we show some basic information for each topology. The Rocketfuel topologies do not offer any information about the traffic demands on the network. We generate traffic matrices using the gravity model which has been used and validated in numerous studies [11], [12], [13]. This model captures the traffic between Points-of-Presence (PoPs) and computes the traffic intensity from PoP A to PoP B to be proportional to the amount of traffic exiting PoP A and the amount of traffic entering PoP B divided by the square of the distance between them. In order to use the gravity model we need to know at least the traffic volumes that enter and leave each PoP. Since we do not have this information we approximate

this with the total capacity of the links into and out of each PoP. Since the Rocketfuel topologies do not provide link capacities we approximate them with the inverse of their weights. This is based on a common configuration guideline [14] that sets each link weight to be inversely proportional to its capacity. Previous work that used both the gravity model and Rocketfuel topologies has used this technique [13]. While previous works on traffic optimization have considered the PoP level topologies, here we consider the router level topologies. One reason for that is that we also study (but we do not report in this work) the effects of router failures, something that can not be captured at the PoP level topologies since nodes there correspond to whole PoPs. Since the gravity model captures only the inter-PoP traffic we will need to extend this model in order to generate traffic demands between routers. The traffic that a PoP originates is considered to originate equally from each node in the PoP and the same for the traffic that a PoP sinks. Thus the traffic $traf(A, B)$ between node A and node B will be $traf(P_A, P_b)/(size(P_A) * size(P_B))$ where P_A is the PoP where A belongs, P_B is the PoP of B and $size(P)$ is the number of nodes contained in a PoP. To maintain the validity of our traffic model we do not consider any intra-PoP traffic. In order to ensure the best possible performance for the SPF routing we optimize the link weights for the no failure case using techniques similar to these presented in [21] using the traffic matrix we have computed from the above process. Before the optimization of the link weights the traffic demands are scaled so that no link has to carry more than 60% or 80% of its capacity. After we have an optimal set of link weights we use this set of weights and the traffic matrix computed above for the rest of the experiments.

In table 5 we compare the best case routing performance that we can achieve with the optimal SPF repair (max_cost_{opt}) and the best case multi-topology recovery ($max\_cost^{mt}_{best}$). As expected, in the cases where the maximum link load is low, the cost of routing after failures is much lower for all topologies. We can see that the multi-topology repair can be as effective as the re-optimization of link weights after each link failure. In some cases, it may even do better. After a link failure, optimal SPF repair will use all the remaining links for routing traffic, while the multi-topology repair will not use the links that are adjacent to the nodes at the two ends of the failed link. If it were these links that were overloaded, SPF repair may not have been able to reduce their load (since we limit the maximum value of link weights that SPF repair can use). The multi-topology repair ignores them altogether and depending on the traffic matrix may be able to achieve a better overall cost. The fact that multi-topology repair can do as well as that of optimal SPF repair is an important result since multi-topology is a practical and implementable method. Optimal SPF repair is not really a practical repair method: even if the link weights to be used after a failure are pre-computed, one can not hope to be able to change the link weights in a network after a link failure; such a change has a high administrative complexity, it is slow, and itself may cause significant disruption in the network as traffic shifts around.

Topology	$max\_cost^{mt}_{best}$	SPF repair	multi-top	not-via
	80% load			
ISP1221	28900	40992	34434	39682
ISP1755	360	15903	20872	40958
ISP3257	12707	70129	20457	105960
ISP3967	32	5922	120	6864
ISP6461	124	1252	595	7728
	60% load			
ISP1221	7672	7698	7695	7706
ISP1755	91	426	156	707
ISP3257	309	22393	11385	41938
ISP3967	15	6006	30	35
ISP6461	30	582	52	12193

Fig. 6. Routing cost, 10 recovery topologies

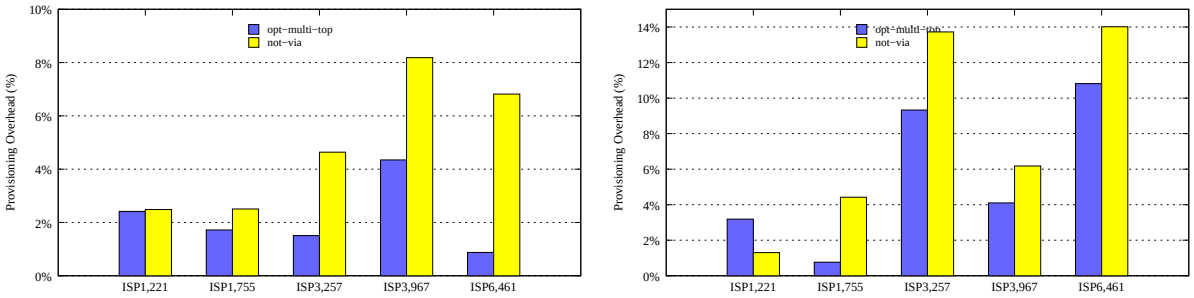
Table 6 shows the routing cost for SPF repair, multi-topology routing with 10 recovery topologies and not-via and how it compares with the best-case values we showed in Figure 5. We see that the multi-topology approach outperforms both SPF repair and the not-via repair, at times quite dramatically. Also, in most cases the performance of our optimization heuristic is fairly close to $max\_cost^{mt}_{best}$. For 80% load only for ISP1755 our approach fails to get reasonably close to $max\_cost^{mt}_{best}$ and this is also the only case where it performs worse than SPF repair. We will discuss why this happens later on. For 60% load, our method has problems with ISP3257 although in this case it outperforms SPF repair.

In Figure 7 we show the provisioning overhead of multi-topology repair and not-via compared to that of SPF repair. Multi-topology repair has the advantage when compared to not-via although the increases are rather small in general, except for some topologies in the 60% load case. In general, we can see that the increased routing flexibility of the multi-topology approaches where we can control the topology that traffic would be routed after a failure makes a difference. The not-via tunnels are constrained to terminate at the node that is "behind" the failed one (node B in the example of Figure 1 when node F fails) and this can result in excessive load on some network resources. Note though that in the case of ISP1755 the performance of the multi-topology approach is worse than that of SPF repair at least when we have 10 recovery topologies. More on this when we compare with the performance with 5 recovery topologies.

We show the storage overhead in Figure 8. The numbers shown is the amount of next hops we will need to store when compared with the SPF repair where essentially next hops are recomputed after each failure. Recall that we only have to store unique next hops and not a different set of next hops for each recovery topology as a result the increase in the number of next hops although not negligible does not grow as the number of recovery topologies increases. This points to the fact that storage will not be an issue with multi-topology approaches. The overall increase is around 100%-150% that compares well with the 100% increase in the amount of addresses we will

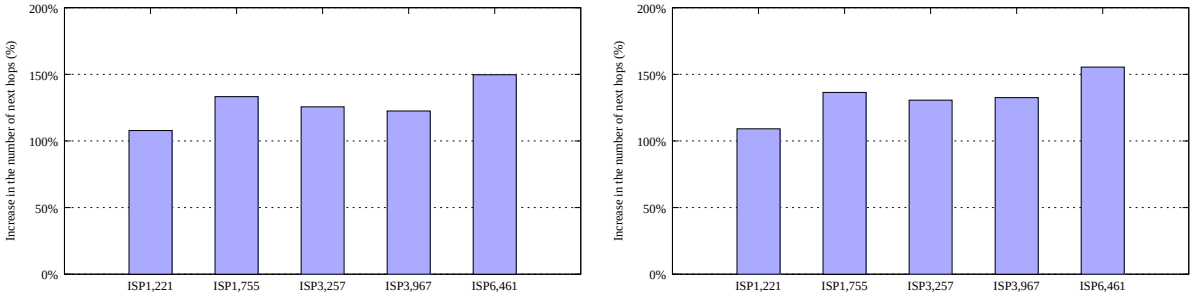
Topology	max_cost_{opt}	$max\_cost^{mt}_{best}$	max_cost_{opt}	$max\_cost^{mt}_{best}$
	80% load		60% load	
ISP1221	22826	28900	7672	7690
ISP1755	435	360	227	91
ISP3257	12170	12707	430	309
ISP3967	86	32	57	15
ISP6461	80	124	10	30

Fig. 5. Optimal max costs



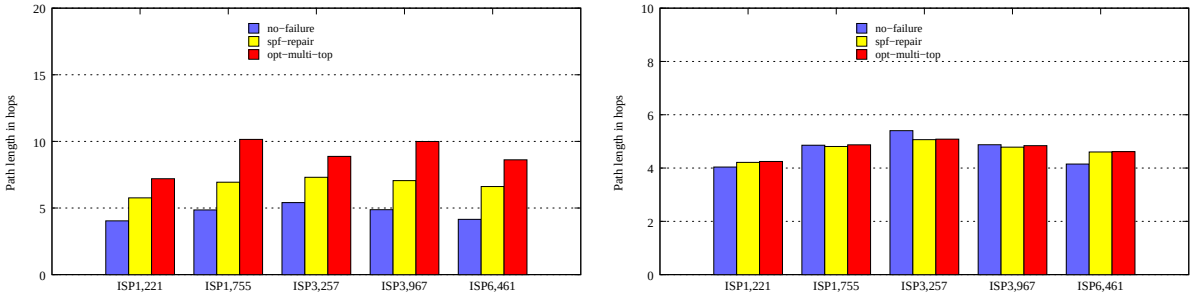
(a) 80% load (b) 60% load

Fig. 7. Network load, 10 recovery topologies



(a) 10 recovery topologies (b) 5 recovery topologies

Fig. 8. Storage for next-hops



(a) maximum path length (b) average path length

Fig. 9. Path lengths, 10 topologies

Topology	5 topologies	10 topologies
ISP1221	45881	34434
ISP1755	13611	20872
ISP3257	19473	20457
ISP3967	605	120
ISP6461	595	83

Fig. 10. Effects of the number of recovery topologies, 80% load

Topology	naive	optimized
ISP1221	39687	34434
ISP1755	15903	20872
ISP3257	80324	20457
ISP3967	6601	120
ISP6461	6434	83

Fig. 11. Naive versus optimized multi-topology recovery, 10 topologies, 80% load

have to use in the not-via method.

Figure 9 shows the maximum and average path lengths for the no-failure case, SPF repair and the multi-topology repair for 10 recovery topologies. The length of paths is an important indication of the increased delay that traffic will have to endure after a failure in the network. The worst case path lengths for the multi-topology repair methods increase significantly and in some cases are almost 100% larger when compared with the no failure case while for SPF repair they are halfway in between. Thus in some cases we would expect an increase in the latency through the network. On the other hand, the average path lengths are almost the same for all three methods showing that the average latency will not be affected too much.

Table 10 shows how cost results depend on the number of recovery topologies used for 80% load. The cost results presented in Table 6 were computed with 10 recovery topologies. Still we have a strong incentive to use as few recovery topologies as possible since then we will less packet header real estate for the topology identifier and each router will have less topologies to perform shortest path computation on. In most cases using 5 topologies results in worse max_cost that when we use 10 recovery topologies. The most notable exception is that of ISP1755 where having 5 recovery topologies perform better than 10. Remember also that topology ISP1755 was the only one where our method was not able to achieve better routing performance than SPF repair. In both the 5 and 10 topology cases, the optimization stops when it is not possible to improve the cost of routing by adjusting link weights. The structure of the underlying recovery topologies will have an important impact on the best value of cost that can be achieved. For example, in the ISP1755 network, when having 5 recovery topologies two links are overloaded. It is not possible to further reduce their load since in the recovery topologies that we would increase their weights they are articulation links so it is not possible to move traffic away from them. When using 10 recovery topologies, there are 3 links that are overloaded, the additional cost of the third link is the cause of the performance difference. Again, it is not possible to reduce the load on the third link because it is an articulation link in the recovery topologies that we would like to reduce its load. When using 5 recovery topologies it happens that we avoid the problem with the third link although we are using fewer recovery topologies overall. This exposes the main deficiency of our scheme. While we make an effort to generate good recovery topologies, they are still relatively arbitrary and in cases like above they may prevent the optimization step from achieving good link weight settings. A comprehensive solution should include mechanisms to jointly optimize recovery topology composition and link weight settings on them. This is currently work in progress and will be reported elsewhere.

Finally, in Table 11 we compare the performance of the optimized multi-topology method with that of an unoptimized multi-topology approach that creates a set of recovery topologies just attempting to equalize the total cost of nodes assigned to each topology as discussed in Section V. Results are for 10 recovery topologies and 80% load. With the exception of the ISP1755 topology that we have already discussed, the naive approach does significantly worse. This shows that a brute force generation of recovery topologies based on simplistic criteria (or even modestly sophisticated like the one we used here) will not be able to ensure good routing of traffic after failures.

In order to get yet another perspective on the performance of our methods, we also considered the pop level topologies. From the 6 Rocketfuel topologies only ISP1755, ISP3967 and ISP1239 are doubly connected at the pop level thus we did not consider the other topologies for these experiments. The traffic matrix is again determined by a gravity model which now naturally captures the inter-pop traffic demands. The results are shown in Figure 12. Performance of the various methods are similar to the results for the router level topologies above. Again our optimization method is effective in achieving routing performance better than both SPF repair and not-via and in most cases can get reasonably close to $max\_cost^{mt}_{best}$.

VIII. Discussion

The multi-topology forwarding methods we propose here have some similarities with MPLS forwarding. Indeed, MPLS forwards packets based on a label contained in the packet. In our approach packets are forwarded based on the topology information they contain. The main difference with MPLS is that currently MPLS requires various other mechanisms for setting up label switched paths which are essentially point-to-point (or multipoint-to-point and recently point-to-multipoint) connections. Typically such connections are setup through the RSVP-TE [26] or LDP [27] protocols. Running these protocols requires network resources and increases the complexity of operating the network. The protocols use relatively complex signaling for setting up and maintaining these paths. On the other hand, our method requires only modest extensions to the path computation component of the IGP (and none in the IGP signaling if it adopts the multi-topology extensions [2],

Topology	max_cost_{opt}	$max\_cost^{mt}_{best}$	SPF repair	multi-top	not-via
	80% load				
ISP1755	7270	32340	68164	38033	77052
ISP3967	157	13970	33858	15010	30741
ISP1239	1006	8159	26286	13300	25700
	60% load				
ISP1755	150	280	23560	6700	13150
ISP3967	78	725	13400	12900	13660
ISP1239	319	223	246	167	268

Fig. 12. Pop level topologies, routing cost, 10 recovery topologies

[1]) so that it can compute the alternate next hops for possible failures. If different nodes have a method to choose a common topology identifier for each recovery topology there is no need to even communicate with each other beyond the standard IGP exchanges. Also, conceptually, what we create here is not a connection but an overlay mesh. MPLS connections usually follow the point-to-point or point-to-multipoint semantics and are one way and in order to build such a mesh a number of point-to-point or point-to-multipoint label switched paths each with different labels and signaling will have to be merged together.

Also, one could argue that although we claim to solve the problem without the need for IP tunneling (unlike the IETF approaches that all require IP tunneling) we will still have to perform the expensive operation of adding a topology identifier to a packet effectively pushing it into a recovery topology. Indeed, the cost of adding the topology information in the header of the packet could be high and could be comparable to the cost of pushing an MPLS packet into an LSP (i.e. adding the shim header in the packet and the appropriate label). Still, MPLS routers are fairly optimized for this operation and many implement this entirely in hardware or firmware and can perform it at high speeds. Also, this operation is still simpler than an IP tunneling operation that requires one to push an additional IP header into the packet and is not well optimized in most routers. Thus although we can not claim to be faster than MPLS tunneling we believe we are faster than IP tunneling.

One concern with all the methods for protecting traffic is their behavior in case the recovery mechanisms are implemented in only a part of the network. Allowing for partial deployment is important since it enables the operator of the network to incrementally deploy the modified software without a flag day. On the other hand, all methods, even the ones that allow incremental deployment have degraded performance when only few of the routers in the network support the traffic protection functionality. Especially in the traffic recovery and micro-loop prevention applications we discuss in this work, having even a single router not being able to react to a failure compromises the whole scheme, since it is not possible to provide any meaningful performance guarantees for the network as a whole. In the same spirit, our proposal relies on all the routers implementing multi-topology routing. This is not a fundamental limitation of our scheme though. It would be easy to know which nodes in the network do not support multi-topology forwarding (the multi-topology IGP extensions provide mechanisms for detecting this) and avoid them when computing the recovery topologies. Everything would work as described so far, but traffic could not be protected from certain resource failures.

We also need to consider what happens in cases when there are multiple simultaneous failures. Although our scheme would not be able to provide recovery of traffic it should allow the network to operate at least at the level it would if our scheme was not employed. Indeed, when there are multiple failures, routers will first detect one of the failures and send traffic in the corresponding recovery topology. If while the packet is forwarded in this particular recovery topology, a router encounters another failure that it is not handled by the current recovery topology then the router can deduce that there is a double failure. Then the router has the option to send the packet back to the primary topology to be routed to the destination through regular routing. Of course the danger is that after the packet is pushed back to the primary topology and while it is being forwarded there another router adjacent to some of the failed resources will try to put it into another recovery topology. It is possible that a routing loop will result from the multiple attempts to protect the packet. Essentially we need a way to signal to the routers that there is a multiple failure in the network and they should not attempt to protect traffic. This is simple to do if the packet is put into a special double failure topology which prevents the routers from protecting the packet. This could be by using some reserved topology number. Note that routers will not need to perform any extra computation for this reserved topology. The next hops for this topology are simply the next hops of the primary topology. The topology related information in the packet is used to signal the double failure. The router that detects the double failure (detects an unexpected failure while forwarding a packet in a recovery topology) will push the packet into the special topology. When the multiple failures have been fixed, no packets will be marked with this special topology since the conditions for their marking will not exist anymore.

A. Eliminating Micro-Loops

The scheme we have presented so far handles the recovery of traffic in cases where network resources fail. Here we will show that we can easily handle micro-loops in a similar fashion. Unlike the cases of traffic recovery, micro-loops can be created due to reasons other that resource failures. In particular, micro-loops can be created by any change in routing that requires the IGP to re-converge e.g. changes in the costs of the links or network resources (both links and routers) being added to the network potentially after a previous failure.

The main idea for dealing with micro-loops is to divert traffic into a alternate topology while the primary topology is converging after the change in the network. In this way, traffic can reach its destination over a stable topology that does not change and as a result there are no micro-loops. In the case of failures, the basic recovery scheme will not provide recovery from micro-loops since traffic that does not cross a failed link will not be pushed into a recovery topology and can still micro-loop. One method to avoid micro-loops is to maintain a "frozen" version of the primary topology, i.e. do not allow updates to it even when network resources fail. Then, when a router is notified for a failure, it immediately (and before staring the re-computation of its routing table) instructs its forwarding plane to start forwarding traffic over the frozen topology. As the failure notification propagates throughout the network, routers switch traffic over the frozen topology. Since the frozen topology does not change and it is repaired by the routers adjacent to the failed resources, all traffic can be delivered to its destination without micro-loops. In case there is no resource failure but there is an addition of resources of changes in link costs, then routers behave the same way, diverting traffic into the frozen version of the primary topology. Until routers are instructed to stop using the frozen version of the primary topology, traffic will flow without micro-loops, but unable to take advantage of the newly added resources and link costs.

Of course the question becomes when routers should stop using the frozen version and switch to the new topology. Clearly, this must be done after all routers have finished computing the new topology, i.e., the IGP has converged. It is hard to know when this has happened without extensions to the IGP. Most approaches in the IETF setup a time delay that allows all routers to converge. Switching to the new topology had to be done carefully. Since there is no easy way to ensure that all routers switch in a synchronized manner, it is possible to have some routers that have not switched yet remark the switched packets back into the frozen topology and create routing loops. A simple method to avoid this problem is to have two timeouts, TIME_COMPUTE, and TIME_SWITCH. Then each router does the following: (a) when a router is informed of a condition that can result in micro-loops, starts a timer, and starts pushing packets that arrive over the primary topology into the frozen topology. If packets arrive over the frozen topology they are forwarded in the same topology. (b) at expiration of TIME_COMPUTE, it continues to forward to the frozen topology packets that arrive over this topology, but it stops pushing to the frozen topology packets that arrive over the primary topology, these are forwarded normally in the primary topology. (c) at expiration of TIME_SWITCH, the router starts pushing packets arriving to it over the frozen topology into the (new) primary topology. If the time intervals TIME_COMPUTE and TIME_SWITCH are sufficiently large for all the routers to compute their SPFs and perform the switch, this sequence ensures that there will not be any routing loops at both the switch to the frozen topology and the one back to the primary one. This is true because in both cases after a packet is pushed into either the frozen or the primary topology it will reach the destination following this topology, and will never have to be pushed to another topology precluding in this way routing loops.

IX. Related work

In this work we focus on achieving traffic repair within the least possible amount of time after the failure. Thus we only consider local repair approaches that trigger the repair at the nodes that are next to the failed network resource. We also necessarily consider only methods where the backup routes for the packets affected by the failures have been pre-computed and installed in the forwarding components of the routers. In this context we can consider the following criteria for comparing different solutions:

- *Type of failure covered*: different methods can handle different types of failures such as link failures, node failures or failures of Shared Resource Link Groups (SRLGs) that describe an arbitrary combination of network resources that can fail together. Still, in most works (and ours is no different) there is the assumption that there is only a single failure (of a link, router or SRLG) at any time in the network.
- *Failure indication*: some methods need to be able to distinguish between link or node failures. While link failures (depending on the link media type) are usually simple to detect, node failures may take more time or require the use of some fast keep-alive protocols such as BFD [3].
- *Coverage*: can a method handle all possible failures or there are some failures that can not be repaired? Some methods can handle only link failures for example. Furthermore, some methods are applicable only in network that satisfy some requirements. For example networks where all link weights are symmetric (i.e. the same for both directions of a link).
- *Cost of computation*: The amount of computation a router has to perform in order to compute the additional information needed for recovery of traffic.
- *Cost of additional storage for forwarding state*: the cost for storing the additional forwarding information needed for the recovery of traffic.
- *Cost of additional processing during forwarding*: some methods require some additional processing while forwarding protected traffic, most commonly some form of

MPLS or IP tunneling.

- *Non-optimal routing of traffic while recovering from failure*: necessarily when a network resources fail, traffic is routed along a path that was not intended by the routing protocol and it may be suboptimal. This will not only affect the traffic being forwarded (for example it may increase its latency) but may also result in the need for additional bandwidth to be provisioned, since some links may become overloaded.
- *Overall complexity* which usually results from having to manage a large number of backup routes. Some methods can achieve their goals with a smaller number of such backup route related information.

Protection of traffic from network resource failures has attracted for long the attention of researchers. One class of solutions uses node or edge disjoint paths between pairs of sources and destinations (see [5] for an overview). The repaired traffic can be switched to a disjoint path locally at the node that is adjacent to the failure. The main drawback of this approach is the potentially large number of paths that has to be created and the unavoidable increase in the size of the connection tables in nodes. To cope with this there have been proposals to use other structures for protecting traffic. One approach is to use multiple edge or node disjoint trees [28], [29] while another possibility is to use a combination of cycles [30]. These structures are fewer and easier to manage but can also prove quite ineffective when routing traffic after a failure. Also some of these methods are more amenable to global repair while other have trouble handling node failures. While all the above methods have been proposed for connection oriented networks such as optical networks they have been extended to operate in IP networks with the help of path setup capabilities of MPLS [31], [32], [33]. These methods have the disadvantage that they need all the MPLS support protocols (usually RSVP-TE or LDP) for setting up the detours [44], [45]. In some recovery models there is the need to setup a rather large number of detours resulting in unmanageable complexity.

Given this complexity and that certain networks may not want to deploy MPLS, there have been proposals to address the problem with an IP-only solution. A simple approach is using Equal Cost Multi-path (ECMP) forwarding, where a router has multiple potential next hops for reaching a destination. When some of the next hops fail the other next hops can still be used to forward traffic. This approach is simple but does not offer good coverage. To improve on the coverage, some have proposed to create "virtual links" using MPLS or approaches like Loop Free Alternates [34] that attempt to explicitly compute alternate next hops where a node can forward traffic when one of its adjacent link (and/or router) fails. These next hops are selected in such a way so that there is no danger of creating a routing loop. In order to do so, the router that computes the alternate next hops essentially needs to perform the SPF computation from the view point of each of its neighbors, since it requires knowledge of their shortest paths. While this approach improves on the coverage of ECMP still does not guarantee that a backup next hop can be found for each possible failure. In order to further improve coverage, extensions have been proposed to allow packets to U-turn [16] (effectively loop detection instead of simply loop prevention) so the loop is broken. The increased coverage is achieved with some extra computation at each router (that now needs to perform an SPF for each neighbor of its neighbors) but still 100% coverage can not be guaranteed.

To liberate from the constrains of the particular topology a series of tunneling based approaches have been proposed for IP-only recovery of traffic. One [17], sets up tunnels around failed resources using an intermediate "release" point. This method can compute tunnels for all possible failures as long as the weights on the links are not asymmetric, so in certain topologies 100% coverage may not be possible. Furthermore, this method needs to be able to distinguish between node and link failures. A newer proposal from the authors of [17] is the use of the "notvia" addresses [15] where alternative addresses are used to direct traffic around a failure. Each interface in the network has an alternative address and when a failure is detected traffic is tunneled to the appropriate alternative address so it avoids crossing the failed resource. This method is able to repair all possible failures and does not need to distinguish between link and node failures. One of the disadvantages of this method compared to the multi-topology repair approaches is the increase in the size of the routing table. The notvia method requires additional addresses that will increase the size and consequently the cost of the expensive forwarding lookup structure. The multi-topology approaches may require more storage for the next hops but do not affect the forwarding lookup structure. Furthermore, the notvia approach still needs to perform the expensive operation of tunneling. and it suffers from instability during convergence after topology and cost changes. Finally, as we showed in our evaluation the notvia approach tunnels traffic around the failed resource to the node at the other end of the failed link or node. This forces traffic to follow suboptimal paths and result in increased loads on certain links during repair. In the topologies we evaluated, the notvia repair approach resulted in higher provisioning overhead in all node failure cases.

An altogether different approach to the problem is to use different routing algorithms instead of the SPF to ensure that there will be backup next hops for each possible network failure. In [35] a Failure Independent Routing (FIR) approach is proposed where a special algorithm computes per interface routing tables that can handle alternate routing of packets in case of failure. Still, this algorithm seems to be able to repair only link failures and only in networks that have symmetric link costs. In addition, it is based on the ability of a router to detect if packets arrive from "unexpected" interfaces; this check may slow down the forwarding plane. An improved version that can handle node failures has also been proposed [36]. Then, since this algorithm is susceptible to micro-loops, network traffic may still get lost while the network convergence after a failure. In [37] a routing algorithm called O2 is presented that can compute exactly two next hops

for each destination in the network, but only if the network satisfies certain conditions. The algorithm covers both link and nodes failures. Another approach [38] has been to have only a subset of routers in the network update their routing tables so that a failure is repaired without any changes in the forwarding operation of the routers. This requires additional complexity in the routing protocol to determine which routers it will notify in cases of failures and will react slowly to a failure since surrounding nodes will have to be notified for the failure and update their paths before traffic is repaired.

Finally, the idea of using multi-topology forwarding for recovery of traffic had been discussed before. [39] briefly reports on this possibility without actually providing any details on the algorithms required or the implementation details. The work presented in [5], [6] follows an approach quite similar to ours. The algorithm proposed in [6] also needs to be given a target number of topologies which can be rather limiting. In their evaluation they conclude that even as few as 3-4 recovery topologies are enough to offer recovery from failures. Although this may be topologically true in the sense that in most graphs it is possible to find as few sets of nodes that their simultaneous removal will not partition the graph this does not say much about the effects of traffic. In this work through the introduction of the provisioning overhead metric and the related evaluation we showed that one will need to be careful how to generate the recovery topologies since naive approaches will have worse routing performance.

A. Routing traffic optimization

Recently there has been work on how to optimize the setting of link weights so as to achieve "good" traffic routing that minimizes a given objective function [11], [25], [21], [22]. While the basic problem of optimal routing can be formulated as a integer linear program, it becomes NP-hard when considering the constrains of IP routing, in particular the inability to split traffic non-uniformly across multiple next hops. As a result existing optimization proposals work by searching the state space of all possible link weight settings (usually within a minimum and maximum limit for the weight values) and relying on heuristics to reduce the state space and speed up the search. Some of the above work [11], [22], [25] focuses on optimizing the link weights so that the performance of the network is good even when single links fail. Search based solutions similar to what has been used for the no failures case have been proposed. A difficulty with these approaches is that now the computation of the objective function is more expensive since we have to consider its value under all possible failures. As a result the exhaustive search of [11] can become too time consuming. To deal with this, [21] considers for optimization only the links that their failures have the largest impact, while [25] proposes a method that relies on building statistics for the importance of each link during the no-failure link weight optimization phase that are then used to compute the best link weight setting in cases of single link failures. These methods can be quite effective. As shown in [25], the method of [11] can find routings that are within 30% of the cost of optimal SPF repair and the method proposed by [25] performs similarly. The performance of our method as shown in Figure 6 is similar in general, although there are cases where we are not able to determine a good solution due to the structure of the underlying recovery topologies.

Overall, we believe that the multi-topology approach is superior to attempting to find a set of link weights that optimize the routing in the network in case of failures. This is because these optimization methods suffer from the fact that they have to optimize the behavior of the network in case of failures without compromising the no-failure performance. To achieve this their objective function is a linear combination of the cost of the no-failure and the failure cases. Giving more weight to the no-failure case may result in bad behavior under failures and vice versa. This performance inter-dependency does not appear in the multi-topology approach; since traffic after a failure is routed on recovery topologies, the weight settings in them will not have any impact on the no-failure operation of the network. In this way we have more flexibility for aggressive weight optimization on these topologies. Note that all this previous work has focused on link failures and did not consider node failures. The multi-topology recovery method we have described in the previous sections can handle both link and node failures, although in this work we showed only results for link failures. Arguably link failures are a much more common type of failures in networks and handling them is probably a priority for every recovery scheme.

B. Micro-loops

Although there have been earlier attempts to measure routing loops, they have focused on the Internet as a whole and larger timescales or persistent loops. Recently there has been some work on characterizing routing loops based on measurements on real networks of a single ISP [40], [41]. They observed that most of the routing loops last for short periods (less than 10 seconds) and they are mostly related to BGP dynamics rather than the operation of the IGP. Still when these loops happen can have quite dramatic effects on the traffic causing a lot of losses and a drastic increase in latency. Furthermore, these studies did not consider the impact of configuration changes (i.e. changes of link weights or bringing new links on-line) to traffic. In certain applications that require absolute network performance even the occasional short lived loop may not desirable.

A variety of approaches has been proposed for handling micro-loops. Some are similar to the ones that compute a backup next hop [42]. Similarly to the loop-free backup next hops, there exist micro-loop-free next hops, where traffic can be safely forwarded without the danger of a micro-loops being formed when the network re-converges. Similar to the traffic recovery case, these approaches can not guarantee full coverage, so in some cases micro-loop-free next hops can not be found. Another approach is to minimize the convergence time by optimizing the propagation of the IGP updates and their processing. Still this can result in network instabilities in certain cases. Another alternative for avoiding micro-loops is

to order the SPFs in the network or control the propagation of IGP updates so that the convergence of the network happens in an orderly manner and routers converge to the new topology in a particular order that prevents the formation of micro-loops. The main problem with these techniques is that they may delay significantly the convergence of the network so they rely on the existence of some recovery mechanism that will prevent traffic losses during the longer convergence period. A more extensive overview of the approaches to the micro-loop problem can be found in [43].

X. Conclusion and Future Work

In this work we presented a comprehensive evaluation of multi-topology recovery methods with respect to the quality of routing they can achieve in cases of failures. While multi-topology recovery methods have some important advantages in terms of simplicity and ability to repair immediately all types of single resource failures, their routing behavior is not very well understood. Still, multi-topology recovery offers unique opportunities for efficient routing since it allows us to individually optimize the link weight settings in each of the recovery topologies and achieve almost optimal routing of traffic in cases of failures without compromising the quality of the no failure routing. We discussed how to experimentally compute a best-case multi-topology routing that is a reasonable approximation of the best routing performance that multi-topology routing can achieve. We showed that this best-case performance of the multi-topology approaches is very similar to that of an ideal routing scheme that after each link failure it could compute and apply instantaneously link weights that optimize routing for this failure. Thus multi-topology routing is a practical and implementable mechanism that can achieve these levels of optimal performance. Then we presented a method for approximating the best-case multi-topology routing by generating a good set of recovery topologies and optimizing the link weights on them and we showed that in multiple different topologies and realistic traffic loads it can achieve performance close to the best-case. Although we mostly focused on achieving routings that reduce the amount of overload on links, we also showed results for the total network load, and increase in path lengths. The performance of the proposed optimization heuristic was good, but in certain case is limited by the inability to jointly optimize the composition of the recovery topologies and their link weight settings.

Some of the most important lessons we learned from this work are: (1) routing performance of multi-topology repair can be as good as that of an optimal SPF repair that instantaneously re-optimizes link weights after each failure, (2) our method can outperform both SPF repair and not-via repair, in some cases quite dramatically, (3) naive breakdown of the network in recovery topologies will not be sufficient for achieving good routing performance, and finally (4) our approach depends on a good selection of recovery topologies and our current two phase approach where we first compute the recovery topologies and then we optimize the link weights on them may not perform well in certain cases. Clearly we need a mechanism for jointly optimizing the composition of the recovery topologies and their link weights.

In terms of future work, we are currently engaged in the implementation of this method in open source forwarding planes and we hope to be able to have some real forwarding performance measurements soon. Also, in future work we will consider the details of applying this approach to recovery in multicast networks, to flesh out the details of applying this method to a real OSPF/IS-IS network where we have to handle complexities such as multiple areas.

Acknowledgments

This work has been carried out under the support of a Marie Curie Excellence Chair (EXC) grant from the European Commission, contract number MEXC-CT-2003-509729

References

[1] P. Psenak et al. Multi-Topology (MT) Routing in OSPF, Network Working Group, Internet-Draft, draft-ietf-ospf-mt, work in progress

[2] Tony Przygienda et al., M-ISIS: Multi Topology (MT) Routing in IS-IS, Network Working Group, Internet Draft, draft-ietf-isis-wg-multi-topology, work in progress

[3] D. Katz and D. Ward, Bidirectional Forwarding Detection, Network Working Group, Internet Draft, draft-ietf-bfd-base, work in progress

[4] M. Garey and D. Johnson, Computers and Intractability, W. H. Freeman and Company, 1979

[5] A. Hansen et al., Resilient Routing Layers for Recovery in Packet Networks, in the International Conference on Dependable Systems and Networks (DSN), June 2005

[6] A. Kvalbein et al., Fast IP Network Recovery using Multiple Routing Configurations, to appear in Proceedings of INFOCOM 2006, Spain, April 2006

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

[8] T. H. Cormen, C. Leiserson, R. Rivest, Introduction to Algorithms, McGraw-Hill, 1990

[9] Rocketfuel: An ISP Topology Mapping Engine, http://www.cs.washington.edu/research/networking/rocketfuel/, accessed on April 2006

[10] Roughan, M., Thorup, M., and Zhang, Y. Traffic engineering with estimated traffic matrices. In Proceedings of the USENIX/ACM Internet Measurement Conference (Miami, Florida, Oct. 2003), pp. 248–258

[11] A. Nucci, B. Schroeder, S. Bhattacharyya, N. Taft, C. Diot, "IGP Link Weight Assignment for Transient Link Failures," 18th International Teletrafic Congress, 2003.

[12] A. Medina, et al., Traffic Matrix Estimation: Comparisons and New Directions, Proceedings of ACM SIGCOMM 02, Pittsburgh, PA, August 2002

[13] M. Roughan et. al, Experience in Measuring Backbone Traffic Variability: models, metrics, measurements and meaning, in Proceedings of the 2nd Internet Measurement Workshop, ACM, 2002

[14] Cisco, Configuring OSPF, 1997, http://www.cisco.com/univercd/cc/td/doc/product/software/ios122/122cgcr/fipr_c/ipcprt2/1cfospf.htm, accessed on April 2006

[15] S. Bryant and M. Shand, IP Fast Reroute using Notvia Addresses, Internet Draft draft-bryant-shand-ipfrr-notvia-addresses, work in progress

[16] A. Atlas, U-Turn Alternates for IP/LDP Fast-Reroute, Internet Draft draft-atlas-ip-local-protect-uturn, work in progress

[17] S. Bryant et al., IP Fast Reroute using Tunnels, Internet Draft, draft-bryant-ipfrr-tunnels, work in progress

[18] K. Nichols et al., Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers, Network Working Group, Request for Comments: 2474, December 1998

[19] Z. Cao, Z. Wang and E. Zegura, Performance of Hashing Based Schemes for Internet Load Balancing, in proceedings of IEEE INFOCOM 2000, March 2000

[20] B. Fortz and M. Thorup, Internet Traffic Engineering by Optimizing OSPF Weights, in proceedings of INFOCOM 2000, Tel-Aviv, Israel, March 2000

[21] B. Fortz and M. Thorup, Robust optimization of OSPF/IS-IS weights, in Proceedings INOC 2003
[22] B. Fortz, M. Thorup, Optimizing OSPF/IS-IS Weights in a Changing World, in IEEE Journal on Selected Areas in Communications, 20(4), 2002
[23] D. Applegate, E. Cohen, Making Intra-Domain Routing Robust to Changing and Uncertain Traffic Demands: Understanding Fundamental Trade-offs, in Proceedings of ACM SIGCOMM (2003) 313-324.
[24] Multi-topology IS-IS, http://www.cisco.com/en/US/tech/tk872/ technologies_white_paper09186a00801e199f.shtml, accessed on March 2006
[25] A. Sridharan, R. Guerin, Making IGP Routing Robust of Link Failures, in Proceedings of the 4th International IFIP-TC6 Networking Conference 2005, pages 634-646
[26] Awduche, D. et al., RSVP-TE: Extensions to RSVP for LSP Tunnels, RFC 3209, December 2001.
[27] L. Andersson et al., LDP Specification, Request for Comments 3036, January 2001
[28] M. Medard et al., Redundant Trees for Pre-planned Recovery in Arbitrary Vertex-redundant or Edge-redundant Graphs, in IEEE/ACM Transactions on Networking, 7(5):641-652, October 1999
[29] A. Itai and R. Rodeh, The multi-tree approach to reliability in distributed networks, Proc. 25th IEEE Symp. Foundations of Comp. Sci. (1984), 137–147
[30] D. Stamatelakis, W. D. Grover, IP Layer Restoration and Network Planning Based on Virtual Protection Cycles, IEEE JSAC Special Issue on Protocols and Architectures for Next Generation Optical WDM Networks, vol.18, no.10, October, 2000, pp. 1938 - 1949
[31] 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), October 2000
[32] R. Bartos and M. Raman, A Heuristic Approach to Service Restoration in MPLS Networks, in Proceedings of ICC 2001, June 2001
[33] A Bremler-Barr, et. al Restoration by Path Concatenation: Fast Recovery for MPLS Paths, In Proceedings of ACM Symposium on Principles of Distributed Computing, 2001
[34] A. Atlas, Basic Specification for IP Fast-Reroute: Loop-free Alternates, Internet draft draft-ietf-rtgwg-ipfrr-spec-base, work in progress
[35] S. Lee et al., Proactive vs. Reactive Approaches to Failure Resilient Routing, in proceedings of IEEE INFOCOM 2004, Hong Kong, March 2004
[36] Zifei Zhong, Srihari Nelakuditi, Yinzhe Yu, Sanghwan Lee, Junling Wang, and Chen-Nee Chuah, "Failure Inferencing based Fast Rerouting for Handling Transient Link and Node Failures," In the Proceedings of IEEE Global Internet, Miami, FL, March 2005
[37] G. Schollmeier et al., Improving the Resilience of IP Networks, in proceedings of IEEE HPSR, Torino Italy, June 2003
[38] P. Narvaez et al., Local Restoration Algorithms for link-state routing protocols, in Proceedings of International Conference on Computer Communications and Networks (ICCCN), October 1999
[39] M. Menth, R. Martin, Network Resilience Through Multi-Topology Routing, Technical Report no. 353, University of Wuerzburg, Institute of Computer Science, May 2004
[40] A. Sridharan, S. Moon, and C. Diot, On the Correlation between Roue Dynamics and Routing Loops, in proceedings of Internet Measurements Conference (IMC) 2003, Florida, USA, October 2003
[41] U. Hengartner, S. Moon, R. Mortier, and C. Diot, Detection and analysis of routing loops in packet traces, in Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment. ACM Press, 2002, pp. 107–112
[42] A. Zinin, Analysis and Minimization of Microloops in Link-state Routing Protocols, Internet draft draft-zinin-microloop-analysis, work in progress
[43] G. Choudhury, IP Fast-Reroute: An Analysis and Applicability in a Core Network, NANOG Meeting, January 2005, http://www.nanog.org/mtg-0501/atlas.html, accessed on July 2005
[44] V. Sharma, Ed., Framework for Multi-Protocol Label Switching (MPLS)-based Recovery, Network Working Group, Request for Comments: 3469, February 2003
[45] P. Pan, Editor, Fast Reroute Extensions to RSVP-TE for LSP Tunnels, Network Working Group , Request for Comments: 4090, May 2005