Network Design for Tolerating Multiple Link Failures Using Fast Re-Route (FRR)

Rakesh K. Sinha*, Funda Ergun†, Kostas N. Oikonomou*, K. K. Ramakrishnan*
* AT&T Labs - Research, New Jersey, USA
† School of Computing Science, Simon Fraser University, Burnaby, BC, Canada
and School of Informatics, Indiana University, Bloomington, IN

***Abstract*—In this paper we present techniques and protocols for protecting a network against multiple link failures. The existing link-based restoration with MPLS Fast Re-route (FRR) is fast, but can create congestion by overloading edges, which is undesirable since many applications are sensitive to congestion-related packet loss. The problem is exacerbated with multiple link failures, frequent enough to be of concern in production networks. In this paper we investigate enhancements to FRR restoration in the presence of multiple failures through network and protocol design. We describe several network designs that add a small number of edges to an existing topology, intended for use by backup paths. For each design, we describe a protocol (that piggybacks on OSPF) for distributing state information and a distributed algorithm for reconfiguring backup paths, after each failure, based on the state information. We prove that for any k, our design, associated protocol, and distributed backup path reconfiguration scheme can handle k arbitrary link failures without causing disconnection or congestion. Through a series of constructions, our final network design is nearly optimal with respect to the number of additional edges in the topology. We believe that these network designs are the first to have such provable guarantees for failures of arbitrary multiplicity.**

I. Introduction

We rely on IP-based networks, exemplified by the Internet, for much of our communication. Traditionally the focus in carrier networks has been on providing *connectivity* upon failure of *single* elements. However, this focus has its shortcomings:

1) Multiple failures are not uncommon in carrier networks. As noted in [23], analysis of a large commercial IPTV backbone over four months revealed that in 17% of link failure events, at least 2 links failed concurrently, and in 2%, at least 3 failed concurrently. Reasons for these overlapping failures are the long repair time, and the well-known case of a fiber cut affecting multiple links.
2) Avoiding packet loss from congestion is as important as maintaining connectivity. Multimedia occupies a dominant share of the IP traffic in core backbones [7], with a growth rate of 83% over the last 18 months compared to 30% for all traffic. Video is particularly sensitive to packet loss [25]. If the restoration strategy puts multiple flows on a link without sufficient capacity, there can be substantial burst losses, resulting in objectionable visual effects in the reconstructed video signal at the decoder.

In this paper we use MPLS FRR with label stacking to tackle both problems by techniques and protocols for maintaining network connectivity after multiple failures without causing congestion.

Managing *congestion* upon multiple failures is harder than just maintaining connectivity. A graph G is $(k+1)$-*edge connected* if and only if it remains connected after the removal of any k edges. So designing a network that remains connected after k edge failures is exactly the well-studied problem of constructing a $(k+1)$-*edge connected* graph [2]. However, even if the graph remains connected, edges can be highly congested, the extent of congestion depending on the restoration method. There are two main restoration methodologies. A *path-based* method reroutes the entire flow if any link in its route fails. A *local repair* or link-based approach replaces a failed link with its backup path without changing the rest of the route.

Observation 1. *If a path-based restoration scheme does not take into account link capacities, e.g. as in OSPF [17], then after k link failures, in the worst case the flow on each of the failed links can be rerouted on the same link, increasing the utilization of this link by a factor of $k+1$.*

The situation is even worse with link-based restoration, as, for example, in the FRR protocol [18]. FRR is widely deployed in carrier networks because of its speed: several implementations restore within $50\,\mathrm{ms}$[1]. Path reconfiguration in FRR is achieved by defining a backup path for each link ahead of time and then stacking an extra label associated with the backup path for the failed link. Decisions are made *locally* by the head-end nodes of the failed links. Note that when a backup path in use experiences an edge failure, it is "patched" by the backup path of the failed edge, and *not* rerouted completely (local nature of FRR), as shown in Figure I.1: if e_1 fails we restore it by its backup path P_1; then if e_2 in P_1 fails, we replace it by P_2, and use the rest of P_1. If P_1 and P_2 shared a link, this common link will get *two* copies of the flows: once as part of the non-failed portion of P_1, and then as part of P_2. There is no global

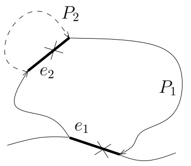

Fig. I.1. Backup paths P_1 and P_2 for edges e_1, e_2.

[1]While lab experiments have demonstrated that the re-convergence time of OSPF can be reduced to sub-seconds by speeding up timers, commercial networks have been reluctant to do so for fear of creating instabilities [8].

controller that sees a flow doubling over an edge and "corrects" it by picking an altogether new path for the failed e_1.

By continuing this example, we can show that in the worst case link utilizations can increase exponentially with the number of failed links (proof available at [21].)

Observation 2. *The worst-case edge utilization after k failures using FRR link restoration can be 2^k times the edge utilization in the absence of failures.*

A standard way to avoid the FRR-induced congestion is to reoptimize the end-to-end primary tunnels after FRR has converged. However, to avoid oscillations, in production networks reoptimization timers are set at several seconds, so congestion can last that long.

In conclusion, methods that avoid link congestion due to failures are important; in this paper we consider the handling of multiple failures by a local repair, FRR-based approach without causing congestion due to backup path overlap.

A. Assumptions and overall scheme

At any given time, each edge e has a backup path. When e fails, this configured path is used for protecting it, and information about the network and the failure is circulated to all nodes (routers). Then the head-end of each non-failed edge reconfigures its backup path ([20], [12]) to avoid path overlaps and failed edges. (We emphasize that the failed edge is patched right away with its *current* backup path so there is no reconfiguration related delay; only paths that are not being used currently get reconfigured.) What makes this problem so tricky is that we need to keep the backup paths of *next failed edge* non-overlapping with back up paths currently in use (Observation 2) without actually knowing which edge will fail next and with the added restriction that all these backup paths have to be formed from a small set of edges we add.

We assume that failures are spaced *sufficiently apart* to allow link state advertisements and the reconfiguration protocol to converge. The mechanism is described in Figure I.2.

```
Configure initial backup path $B(e)$ for each edge $e$
After failure of edge $e$
    Use $B(e)$ to protect $e$
    Propagate relevant information to all routers
        using OSPF TLV pairs
    For each non-failed edge $e'$, reconfigure $B(e')$
```

Fig. I.2. High-level backup path reconfiguration scheme.

II. Formal problem statement and summary of our results

Our goal is to avoid loss of connectivity *and* congestion under multiple failures in a network employing FRR and backup path reconfiguration. In order to give worst-case guarantees under arbitrary traffic variations, we make a conservative decision to *disallow all backup path overlaps*.

Thus, our goal is to add a small number of edges to our network as well as define a non-overlapping backup path reconfiguration scheme on the resulting design that will tolerate multiple failures. A more formal statement follows.

Problem Statement. We are given a graph $G_0(V, E_0)$ with n nodes, and bidirectional edges. (u, v) denotes an undirected edge between nodes u and v, whereas $[u, v]$ and $[v, u]$ denote the two directions of this edge (*arcs* or *directed edges*). These provide independent transport, meaning that their flows do not overlap. However, an "edge failure" in our model means failure in *both* directions. Let k be the number of edge failures we have to tolerate. Our goals are to

1) Pick the smallest possible set of *additional* edges E_a to be used for FRR backup paths, assuming that an edge can be added between any pair of nodes. In §V we show how to extend our results to a setting where certain node pairs cannot be connected, each new edge has a different cost, and we want to minimize the total cost.
2) Configure initial backup paths for each edge in E_0.
3) Define a set of new type-length-value (TLV) tuples for the OSPF protocol to distribute information needed for backup path reconfiguration.
4) Specify a *distributed* backup path reconfiguration mechanism for non-failed edges, with the further restriction that the head-end of any edge computes the new backup path based only on information from previously defined TLV tuples.
5) Give worst-case guarantees that the scheme protects against any k edge failures in E_0 or E_a without losing connectivity or creating backup path overlap.

A. Some intuition on the difficulty of the problem

Observations 1 and 2 in the Introduction can be used to create $(k + 1)$-way path overlap with OSPF, and 2^k-way path overlap with FRR even if the network remains connected. This shows that preventing path overlaps with FRR is much more than maintaining connectivity, and congestion can be significant without the right network design or backup paths.

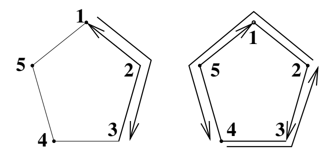

Fig. II.1. Naive vs intelligent choice of backup paths for $(1, 3)$ and $(2, 4)$.

Figure II.1 gives one more example of how the choice of backup paths can make a big difference. Consider a (bi-directional) cycle to be used for backup paths, and let the first failed edge be $(1, 3)$. On the left, we naively use the shortest paths $1, 2, 3$ and $3, 2, 1$ for the two directions of the edge $(1, 3)$. Then if $(2, 4)$ is the second edge to fail, there is no way to find a non-overlapping backup path. On the right of the figure we use paths $1, 2, 3$ and $3, 4, 5, 1$ to protect edge $(1, 3)$. Then we can use the non-overlapping paths $2, 1, 5, 4$ and $4, 3, 2$ to protect $(2, 4)$. There are four messages here.

1) The naive scheme requires one cycle per failed edge, whereas an intelligent scheme is *twice* as efficient.
2) Once we pick the (naive) shortest path for $(1, 3)$, even reconfiguration does not help; a bad choice of backup

path may doom us for future failures.

3) While this example may suggest a simple strategy of protecting two edges with one cycle, additional complexity arises from failures of edges in back-up paths that are already protecting failed edges. So if edge e_1 fails and is protected by backup path p_1, then edge e_2 in p_1 fails and is protected by p_2, then e_3 in p_2 fails, and is protected by p_3, FRR does *not allow* us to go back and pick a path p'_1 for e_1 that does not contain e_2, e_3. We describe the machinery to handle such "recursive" protection in §IV-C.
4) The network design, the protocol for distributing state information, and the distributed algorithm for computing new backup paths have to work in concert. The reconfiguration and path selection must be done in a distributed fashion by the head-end nodes of each link; even the two end-points of a bidirectional link cannot coordinate their decisions. Finally, this information has to be flooded by OSPF LSAs.

B. Organization of the paper and our contributions

In §IV we give three network designs and associated backup path schemes. All three designs are straightforward to describe, but their correctness proofs (no loss of connectivity and no backup path overlap) increase in complexity. The final construction almost achieves the theoretical lower bound on the number of additional links. This section ends with a numerical comparison of the three constructions. In §V, we address how to incorporate several practical concerns imposed by production networks that are not captured by our somewhat idealized problem definition. Finally, in § VI, we specialize our results to the case of a single edge failure in a multicast tree network. In such networks only one direction of any edge carries normal traffic, and the other can be used in backup paths; we simplify the results of [5] and [14].

III. Related work

In [9], [16], [6] and related papers, methods for "the network design problem with connectivity requirements" are studied. These ILP- and multicommodity flow-based methods design a minimum-cost network (from the ground up), able to carry its demands without congestion under a given set of failure scenarios, each consisting of a set of link failures.

Several schemes have been proposed for providing connectivity after double edge failures. For the most part, these do not address preventing congestion (other than by doubling capacities where flows overlap). In addition, they do not easily generalize to failures of higher multiplicity. Several papers extend the idea of constructing disjoint trees, proposed in [11]. In [3], the authors show how to construct optimal pairs of trees for weighted *directed* graphs. Paper [13] divides incident edges to any node into "protection graphs," at least one of which is available after a failure: the first failure is dealt by protection graphs and the second using disjoint trees.

Paper [4] considers the optimization problem of finding the largest set of preconfigured FRR backup paths under the following scenario. Suppose that a second failure, e_2, happens in the backup path of a first failed edge, e_1, and is patched using the backup path of e_2. Then this backup path must not include e_1. A heuristic is given. In [19], it is shown that a solution always exists for all dual edge failures as long as the network graph is 3-connected, and a heuristic is given.

Another way to deal with failures in preconfigured backup paths is to *reconfigure* the paths. Such an approach is discussed in [20], [12] and forms the basis of our schemes in this paper. Reconfiguration provides a powerful framework because the backup path for the second failed edge is selected based on the knowledge of exactly what has failed before (instead of having to deal with all possible failures). Reconfiguration schemes also generalize more easily to higher-multiplicity failures. However, unlike our provable worst-case guarantees, the focus in [20] and [12] is on evaluation and design of reconfiguration schemes given a fixed underlying network. The work in [22] also considers re-provisioning of backup paths, albeit at the end-to-end connection level.

There is a substantial body of work on "p-cycles" [10]. A p-cycle is a preconfigured cycle of unit capacity formed out of the spare capacity in the network. It provides fast restoration for failures on the cycle as well as "straddling" spans with both end-points on the cycle. The original design was proposed for single failures, but has been extended to double failures [1] and SRG (shared risk group) failures [15]. While one of our designs also uses cycles, it is fundamentally different from p-cycles. To our knowledge, there is no p-cycle construction guaranteeing congestion-free restoration for an arbitrary number of failures. Moreover, while many papers on p-cycles propose integer linear programs (ILPs) to find "optimal" cycles in a given network, we propose a simple (and nearly optimal) set of edges that can be added with guaranteed performance and without computation.

In [24] an ILP is provided which provides an initial flow assignment (with flow splitting) and "flow redistribution" scheme that guarantees no congestion with k failures under the following condition: given any set F of edges with total utilization up to k, all the flows can be routed on F without congestion. This work is related to but different from ours; while we propose a network design, they give a guarantee on a network satisfying a pre-condition without directly addressing how to construct such a design.

IV. Network designs to support k edge failures

In this section we present three schemes for protecting against k failures in a network using FRR, without path overlaps. Each scheme consists of a set of additional edges E_a, a protocol for distributing and maintaining relevant state information related to backup paths, and a distributed algorithm for initial backup path assignment and for backup path reconfiguration (recall §II).

Before describing the schemes, we point out a lower bound on the number of additional edges, E_a. We claim that each node must have at least k incident edges in E_a. The intuition is that if a node has at most $k-1$ incident edges in E_a

then we can fail all these $k-1$ edges and one more original edge incident to this node and create a graph where the (original) failed edge (because there are no surviving edges in E_a incident to v) has no FRR protection path. Because a graph has at least (number of nodes) $\times$ (min degree)$/2$ edges, we get

Observation 3. *Supporting k edge failures via FRR without using any of the original edges in E_0 requires at least $\lceil kn/2 \rceil$ additional edges in the worst case.*

This lower bound exploits just the fact that to tolerate a failure each node must have at least one incident edge in E_a after the failure; it uses nothing about FRR paths, or restrictions of a distributed protocol for reconfiguring backup paths.

The three schemes consist of adding (1) a collection of spanning trees (this is simply a baseline for what can be achieved), (2) parallel edges, and (3) disjoint spanning cycles. The schemes are increasingly efficient in the size of E_a. The size of E_a in our final construction matches the lower bound for even k, and is $n/2$ more than the bound for odd k. We conjecture that this construction is optimal even for odd k, and that a matching lower bound can be derived by realizing that a backup path consists of *several* edges, and that backup paths for different edges must be disjoint.

To save space, we give the protocol for distributing state information only for our last, most efficient, construction.

A. A collection of spanning trees

A very straightforward construction is to add a spanning tree for each possible edge failure:

Construction IV.1. *E_a is a union of k edge-disjoint spanning trees.*

This will serve as a baseline to judge the efficiency of the next two constructions.

B. Parallel edges

A trivial way to protect against k failures is to create $k+1$ copies of each edge. We show that if G_0 is single-edge survivable, a condition satisfied by almost any production topology, we can get by with adding *half* as many edges as the trivial solution.

Construction IV.2. *E_a consists of $\lceil (k+1)/2 \rceil$ parallel copies of each edge of E_0.*

Remember that we are giving a theoretical guarantee on failure of any set of k edges so, in our model, failure of (say) two copies of an edge will be counted as *two* (not one) failure. In practice, if we are adding two copies of an edge in the IP layer, we would like to route them differently in the optical layers to make their failures somewhat independent.

BACKUP PATH SELECTION AND RECONFIGURATION SCHEME

Each edge (in E_0 or E_a) picks one of its copies as its backup path. When an edge discovers that it is the last surviving copy, it reconfigures its backup path to be the first copy of an edge along a path in G_0 from it head to its tail.

It is possible that this first copy may have failed, but, by our construction, the failed edges along backup paths are guaranteed to be backed up by their copies. The edge picking its backup path does not need to know which edges along this path have failed, which greatly simplifies the protocol. For example, if e picks a backup path consisting of e_1, e_2, e_3, e_4, the head-end of e does not need to know that e_3 has failed and was replaced by e'_3. It will simply push a label for e_1, e_2, e_3, e_4, and when the packet arrives at e_3, the head-end node will push a label for e'_3. This new label will be popped at the tail-end of e'_3 and the packet will continue on e_4. In other words, none of this information needs to be distributed in the network.

Theorem IV.1. *If G_0 is 2-connected (single edge-survivable), Construction IV.2 suffices to handle k failures with no backup path overlap.*

Proof: The only bad case we need to handle is when all copies of an edge e fail. We will assume that k is even; the proof is similar for k odd.

If all $k/2+1$ copies (one original and $k/2$ additional) of e fail, then there are $k-(k/2+1) = k/2-1$ edge failures outside of e and its copies. So for any edge except e, at least $(k/2+1)-(k/2-1) = 2$ copies survive[2]. That leaves at least one more copy for each edge in E_0, except for e. Then e finds a path in G_0 between its endpoints (such a path exists because G_0 is single-edge survivable) with the modification that a spare copy is used in place of the original edges.

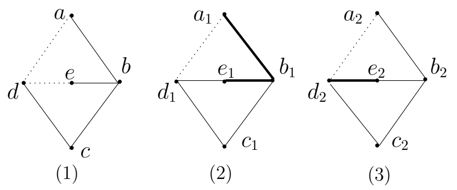

Fig. IV.1. A graph G_0 with parallel edges shown as having an extra two copies, shown in (2) and (3). The thick lines show the final path from a to d after failures $(a,d),(a_1,d_1),(e,d),(a_2,d_2)$.

An example of this scheme is in Figure IV.1; here a, a_1 and a_2 (similarly with b,c,d,e) refer to the same node; they are numbered in order to distinguish between parallel edges. After the first two copies of (a,d) fail, its traffic is routed on (a_2,d_2). Afterwards, (e,d) fails and (e_1,d_1) is used to back it up. When (a_2,d_2) fails, the backup path is $(a_1,b_1),(b_1,e_1),(e_2,d_2)$. Notice that (e_2,d_2) has replaced (e,d) since (e,d) has failed and (e_1,d_1) is already in use backing up (e,d), leaving (e_2,d_2) as the spare copy. ■

Note: the construction can be improved for odd k by using one less copy of each edge and adding a spanning tree.

C. Disjoint spanning cycles

This is our most efficient construction:

[2]Weaker consequence: at most one edge can lose all its copies.

Construction IV.3. *E_a is the union of $p = \lceil k/2 \rceil$ mutually edge-disjoint spanning*[3] *cycles.*

BACKUP PATH SELECTION AND RECONFIGURATION SCHEME

Remember that an edge (u, v) consists of two (directed) arcs: node u is responsible for arc $[u, v]$ and node v for arc $[v, u]$. In our first two constructions, to keep the exposition simple, we described all decisions in terms of edges but in reality, u and v act *independently* so here we will be more precise and talk in terms of arcs, not edges.

The main idea of our design is that we add several cycles to our network. The goal is for each cycle to support two backup paths, one in each direction. The difficult case is when the cycle itself experiences a failure, taking out both directions of an edge. We analyze each case below, and, show that, even if the cycle experiencing the failure is already carrying a backup path, we can still reconfigure.

In what follows, we call an arc `active` if it is in use. The formal (recursive) definition is: a non-failed arc is active if it is either (a) part of the original graph G_0, or (b) in the backup path of a failed active arc.

The spanning cycles are used only for backup paths. Let C_i denote the ith *undirected* cycle. C_i has two characteristics: *tolerance*, and *state*. The tolerance $tol(C_i)$ is the number of failures that C_i can be used to back up. Initially all tolerances are 2 because each orientation of any cycle can restore one failed edge, as described in "intelligent scheme" in Figure II.1. If a cycle has one failed edge and no active arcs then it is in state `LinkFail`. Such a cycle has tolerance 1 since it can be used to restore one edge. If the cycle has no failed edges but its clockwise (but not the opposite) orientation is used for restoring an edge, its state is `alloc` and it has tolerance 1.

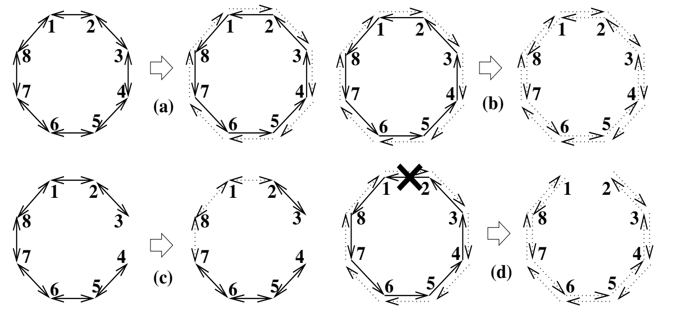

Fig. IV.2. (a) shows a Type I allocation from a cycle with tolerance 2, (b) and (c) from a cycle with tolerance 1 with one active backup path and one failed edge, Types II and III respectively; (d) shows a Type IV allocation after $(1, 2)$ fails. After the allocation the dashed edges are in use (active) and the solid edges remain free.

There are four types of path allocations on a cycle, based on its tolerance and state. We define Types I, II, and III for backing up undirected edges in both directions and Type IV for backing up a single direction. In Types I, II, III, the failed edge is restored in a cycle C_i that it does *not* belong to; in Type IV, the failure and the restoration happen within the same cycle C_i. We demonstrate these allocations in Figure IV.2. For (a), (b), and (c), the edge to be backed up is $(2, 7) \in G_0$.

Type I: C_i has tolerance 2, it is intact. We allocate clockwise around C_i, with paths $2, 3, 4, 5, 6, 7$ and $7, 8, 1, 2$ as in Figure IV.2(a). The resulting cycle has state `alloc` and tolerance 1.

Type II: C_i has tolerance 1 and state `alloc`. In this case the clockwise orientation has been used previously, so we perform a counterclockwise allocation $2, 1, 8, 7$ and $7, 6, 5, 4, 3, 2$ as in Figure IV.2(b). The resulting cycle has tolerance 0.

Type III: C_i has tolerance 1 and state `LinkFail`. We perform a bidirectional allocation. Figure IV.2(c) shows an example, where the backup paths are $7, 8, 1, 2$ and $2, 1, 8, 7$. The resulting cycle has tolerance 0.

Type IV: C_i has tolerance 1 and state `alloc`; additionally, the failed edge, say $(1, 2)$, is on C_i. By definition, only *one* direction, $[1, 2]$ in this case, was used in a backup path. We use the counterclockwise orientation of C_i to backup the failed edge. Figure IV.2 shows how this can be done with $1, 8, 7, 6, 5, 4, 3, 2$. The resulting cycle has tolerance 0.

D. Distributed protocol details

We have described which restoration path to use after a failure for illustration, but in reality our scheme must ensure that the backup paths must get reconfigured *before* the actual failure. Figure IV.3 shows how, in our scheme, backup paths are reconfigured and the parameters of the cycles.

```
INITIALIZATION PHASE:
for all $i$, set $tol(C_i) = 2$
configure Type I backup paths in $C_1$ for all arcs in $E_0$
REROUTING UPON FAILURE OF $e = (u, v)$:
if both $[u, v], [v, u]$ are inactive, i.e. $(u, v) \notin E_0$
    let $C_i$ be the cycle containing $(u, v)$:
    decrement $tol(C_i)$, set its state to LinkFail
else
    if $e$ is active in $[u, v]$ direction
        let $C_j$ be the cycle with backup path for $[u, v]$:
        restore $[u, v]$ along its backup path in $C_j$
    if $e$ is active in $[v, u]$ direction
        let $C_j$ be the cycle with backup path for $[v, u]$:
        restore $[v, u]$ along its backup path in $C_j$
    decrement $tol(C_j)$; if it is 1, set state of $C_j$ to alloc
RECONFIGURATION AFTER RESTORATION OF $(u, v)$:
for all active arcs $[u', v']$
    if $[u', v']$ is on a $C_j$ with tolerance 1, state alloc,
        $u'$ configures a Type IV path on $C_j$
    else
        $u'$ finds the smallest $i$ such that $tol(C_i) = 2$ and
        configures a Type I path on $C_i$
    if no such cycle exists then
        $u'$ finds smallest $i$ such that $tol(C_i) = 1$ and
        configures a Type II or III path on $C_i$
```

Fig. IV.3. Backup path reconfiguration and updating of tolerance values

[3]i.e., visiting all the nodes, but not necessarily Hamiltonian, since self-intersection is allowed.

1) Coordinating decisions between two endpoints of an edge: As pointed out earlier, u and v work independently on arcs $[u, v]$ and $[v, u]$. When both arcs need to be restored (Types I, II, III), we force them to pick the *same* cycle by picking the available cycle with the lowest index.

2) Distributing and maintaining states related to cycles: We assume that each node u knows, for each C_i, (a) the edges in C_i, (b) $tol(C_i)$ and state of C_i, (c) if $tol(C_i)$ is 1 and state is `LinkFail`, which link in C_i has failed. Additionally, for each incident edge (u, v), it knows (d) whether or not $[u, v]$ and $[v, u]$ are active.

We briefly describe extensions to OSPF to achieve this knowledge. Each cycle is assigned an `id` in the router configuration file. Similar to the link state data base exchange, where each u floods the entire network with information on its incident links, we require u to also send $([u, v], i)$ for each $[u, v] \in C_i$. Then each node, in addition to maintaining a copy of all links, also maintains a copy of all links in each cycle. The pseudocode describes how head-end node u of a failed edge updates the state and tolerance of one cycle. Node u floods the `id` of the cycle used, its new tolerance, and state value to all other nodes using new type-length-value pairs as described in [20]. This triggers each edge whose configured backup path overlaps with the new active backup path to configure a new path using one of the allocation schemes described above.

We must now infer whether or not $[u, v]$ and $[v, u]$ are `active`: if $(u, v) \in E_0$ then both directions are `active`. Otherwise let C_i be the cycle containing (u, v). If $tol(C_i)$ is 2 or $tol(C_i)$ is 1 and its state is `LinkFail`, both $[u, v]$ and $[v, u]$ are `inactive`. If $tol(C_i)$ is 1 and its state is `alloc`, we consider the clockwise arc to be `active` and the counterclockwise arc to be `inactive`; finally if $tol(C_i)$ is 0, we consider both $[u, v]$ and $[v, u]$ `active`. (The last condition may declare some `inactive` edges as `active`, but it keeps the bookkeeping simple without affecting the correctness of the scheme.) In other words, the determination of `active` or `inactive` can be made based on tolerance and state of cycles and does not require any extra distribution of information.

Theorem IV.2. *Construction IV.3 handles k failures with no backup path overlap. Further, this construction uses the optimal number of additional edges for even k and at most $n/2$ edges more than optimal for odd k.*

Proof: Recall the notion of tolerance: $tol(C_i) = 2$ iff C_i is untouched by failures or backup paths. $tol(C_i) = 1$ iff the arcs that have not failed or been used in any backup paths form either (a) an undirected spanning path after a Type I allocation or (b) a directed spanning cycle since an edge in C_i has failed. If $tol(C_i) \leq 0$ then C_i cannot be used for any backup path.

Let $NT = \sum_i tol(C_i)$ denote the network tolerance, which is how many more failures the network can handle at any given time. We show that after $0 \leq r \leq k$ failures, $NT = 2p - r$ and the network can survive $2p - r$ more failures. Initially, $r = 0$, $tol(C_i) = 2$ for each C_i, and $NT = 2p \geq k$. We will show that NT decreases by one after a failure, say, of edge $e = (u, v)$, by a case analysis. In the rest of this proof, we abuse notation by using C_i to denote the set of arcs in C_i that have not failed or been used in any backup paths. E.g, if C_1 experiences a failure, we refer to the resulting undirected spanning path again as C_1. We decompose each undirected cycle C_i into two directed cycles, one in each orientation, named C_i^1 and C_i^2.

Case 1: $e = (u, v)$ is an edge in G_0. Then we use a C_i with $tol(C_i) \geq 1$ to back up $[u, v]$ and $[v, u]$ and decrease $tol(C_i)$ by 1. This will result in an allocation of Type I, II, or III as shown in Figure IV.2. If initially $tol(C_i) \geq 1$ then the existence of the required backup path(s) is guaranteed and $tol(C_i)$ and NT decrease by 1.

Case 2: e is an edge in C_i not used in any backup paths. We do not need to back up e. If $tol(C_i) = 2$, the failure reduces C_i to an undirected spanning path. If $tol(C_i) \leq 1$, C_i will not be used in any subsequent backup paths. In all cases, $tol(C_i)$ and NT decrease by 1. [4]

Case 3: e is an edge in C_i and e is being used in a backup path. Because C_i is being used in a backup path, we know that $tol(C_j) \leq 1$. There are two sub-cases.

Case 3.1 If both directions of e are used in backup paths, then, by the algorithm, C_i must have been used in a Type I and Type II allocation, dropping its tolerance to 0. Because $NT > 0$ by our inductive hypothesis, there must be a cycle C_j with $tol(C_j) \geq 1$, so we can back up e in C_j using a Type I, II, or III allocation, and decrement $tol(C_j)$ and NT by 1.

Case 3.2 If only one direction of e is used in a backup path, then we have used one orientation of C_i, say C_i^1. C_i^2 has now become a directed path from u to v and can be used to back up $[u, v]$, with a Type IV allocation; see Figure IV.2 (d). The tolerance of C_i decreases by 1 to 0 and every other cycle remains unchanged so NT decreases by 1. This is the only case where we back up only one direction of an edge.

We now show the optimality of our construction. From Theorem 3, we need at least $\lceil kn/2 \rceil$ additional edges. For even k, this is $kn/2$ which matches the number of edges in $k/2$ cycles. For odd k, the construction requires $(k + 1)/2$ cycles and $(k + 1)n/2$ edges, which is $n/2$ more than the lower bound, as claimed. ■

E. Numerical Comparison of Constructions

Table I shows the additional edges as a percentage of the number of original edges for constructions IV.1 to IV.3. This percentage only depends on the average degree of the graph. We consider three cases, corresponding to original graph G_0 being sparse (average degree 3) to dense (average degree 10). Optimal construction IV.3 is twice as good as the naive construction IV.1. Comparing constructions IV.1 and IV.2, the former is better for dense graphs because we are adding fixed edge trees instead of making multiple copies of each edge. While the number of additional edges increases linearly in all constructions, the performance is impressive. E.g, a network with average degree 7 and 57% overbuild can support

[4]Even if the tolerance of a node becomes negative, NT is still a lower bound on how many failures we can handle.

	k			
	2	4	6	8
CIV.1	133	266	400	533
CIV.2	100	200	300	400
CIV.3	66	133	200	266

(a)

	k			
	2	4	6	8
CIV.1	57	114	171	228
CIV.2	100	200	300	400
CIV.3	28	57	85	114

(b)

	k			
	2	4	6	8
CIV.1	40	80	120	160
CIV.2	100	200	300	400
CIV.3	20	40	60	80

(c)

TABLE I
(A) % ADDITIONAL EDGES FOR SPARSE G_0, AVG. DEGREE 3, (B) % ADDITIONAL EDGES FOR MEDIUM G_0, AVG. DEGREE 7, (C) % ADDITIONAL EDGES FOR DENSE G_0, AVG. DEGREE 10

FRR-based restoration with guarantees of connectivity and no congestion for failures of up to 4 edges.

V. Some extensions for production networks

We now discuss several extensions to deal with practical constraints not addressed by our problem and solution so far.

A. Disallowed edges and different edge costs

Our goal has been to minimize the number of additional edges. In reality, each new edge may have a different cost and we want to minimize the sum of the costs. Also, direct links between certain node pairs may be prohibited.

To address these issues, we generalize the construction of §IV-C to the case where the C_i's are not simply cycles, but arbitrary 2-connected spanning graphs. A spanning cycle is a special case, so this generalization gives us greater freedom in selecting additional edges, thereby reducing the overall cost. Our construction, definitions, and proofs are similar to those in §IV-C, so here we just highlight the differences.

When C_i is a cycle with tolerance 2, we can back up both directions of any edge $e = (i, j)$ using one of C_i's orientations. This leaves C_i with tolerance 1 and a directed cycle, which can back up any other edge in both directions. Now assume instead that C_i is a 2-edge connected graph (but not a cycle) H with tolerance 2. H contains by definition a cycle containing both i and j, so we can back up $[i, j]$ and $[j, i]$ using one orientation of this cycle, reducing the tolerance of H to 1. It is not clear that after this H can be used to back up another edge: even though originally there were two disjoint paths between any pair of nodes in H, both may have lost edges to the backup path of e. We show that this is not a problem:

Claim V.1. *If $tol(H) = 1$, then H contains edge-disjoint backup paths for $[i, j]$ and $[j, i]$ for any node pair i, j.*

Proof: There are two cases. (a) H is 2-edge connected minus one edge (which has failed). Then it still has an undirected path from i to j, which we use to back up $[i, j]$ and $[j, i]$. (b) H consists of a 2-edge connected graph minus a directed cycle (used earlier for backing up a failed edge). This is described in Figure V.1 where the directed cycle a, b, y, c, d, x remains after its opposite orientation has been used. Initially H was 2-edge connected, thus there were two edge-disjoint paths between i and j. The undirected $i \to a$ path is the portion of first $i \to j$ path before it intersects the directed cycle. Likewise, the $i \to b$ path is the part of the 2nd $i \to j$ path before it intersects the cycle. The rest of the construction is similar to that in §IV-C with the $k/2$ spanning cycles.

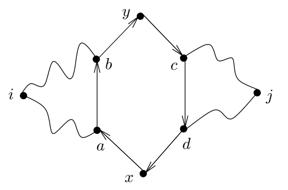

Fig. V.1. A 2-connected graph with a directed cycle removed.

To back up $[i, j]$, we use the $i \to b$ path, then follow the directed cycle to c and use the $c \to j$ path. Similarly, to back up $[j, i]$, we go from j to d, follow the cycle to a, then go to i. ■

Thus, using an argument similar to that of adding cycles to the network, we see that adding $\lceil k/2 \rceil$ arbitrary 2-connected subgraphs will provide protection against k failures.

B. Dealing with simultaneous failures

Multiple failures occur for two reasons. First, a second failure may occur while the first hasn't been repaired yet, a third while the first two are still being repaired, and so on. We refer to these failures as *overlapping*. Our worst-case guarantees in §IV apply to these kinds of failures because we need time to reconfigure backup paths between link failures. A set of failures can also occur *simultaneously* when there is a common cause, such as a fiber cut, failure of an optical amplifier, etc. *No scheme can give a worst-case guarantee for such failures* because, in the worst case, a link and its configured backup path can fail together. We make some suggestions below.

Construction IV.2 is resilient against simultaneous failures, as long as not *all* $k/2$ copies of an edge fail together. We can guarantee this by routing IP layer edges disjointly on the underlying optical layer. Second, a concern with simultaneous failures is that our protocols will create many congested edges. E.g., in Construction IV.3, the backup paths can be selected from one of $k/2$ cycles but our deterministic protocols force each head-end node to pick a path from the *same* cycle. A simple change to the protocol may improve its performance in practice: have each head-end pick its path from a *randomly-chosen* cycle. This does not affect any of the worst-case guarantees for overlapping failures but reduces the probability of path "collisions" in case of simultaneous failures.

C. Allow original edges in backup paths

In order to give worst-case guarantees under arbitrary traffic variations, we conservatively assumed that all original edges are carrying traffic and none of them have the capacity needed to carry additional flow from a failed edge. Thus all backup paths had to use (new) additional edges.

However, if the edge utilizations are guaranteed to be small, one might use available capacities for backup paths instead of adding edges. One heuristic is to view each edge as a set of parallel edges where the first one carries traffic and the rest are to be used as additional edges in our scheme. E.g, if the utilization is under 33%, we split each edge into 3, each with $1/3$ the capacity of the original. Since this is a heuristic, our worst case guarantees do not apply.

VI. Special case: a single edge failure in a multicast tree

We now consider a special case where the original graph G_0 is a single-source multicast network. Such networks, because of their efficient use of capacities, are ideally suited for multimedia distribution. All of our results so far apply to multicast networks, but here we exploit their special structure. A key observation is that only one direction of any edge in a multicast network carries normal traffic and the other direction can be used in FRR backup paths. Recall that our techniques in the previous section assumed that both directions of edges in G_0 carry traffic and therefore the backup paths require the addition of new edges E_a. Although a few of our results in this section can be generalized to multiple failures, we focus on single failures only.

We first show that any 2-(edge) connected graph can be used to construct a multicast tree and a set of arc-disjoint (from the tree) backup paths for each arc of the tree. As first shown in [5], this suffices to guarantee that single edge failures will not cause congestion. The primary advantage of our alternate proof is that we map it to a well-studied graph theory problem of finding strong orientations. Using known results in graph orientation, our constructions are much simpler, and hopefully yield better insight into the problem. We also show that essentially the same construction works for the problem in [14], of constructing disjoint unicast routes and a multicast tree.

A. Improved proof of dealing with single edge failures in a multicast network

A strong orientation of an undirected graph is an assignment of direction to edges so that the resulting graph is strongly connected (has a directed path between each pair of nodes.)

Theorem VI.1. *(Robbins' theorem, Ch. 9 of [2]): Given a simple (undirected) graph G, its edges can be directed to form a strongly-connected graph iff G is 2-connected.*

Given any 2-connected graph G, apply Robbins' theorem to get an orientation and let S_1 be the resulting strongly-connected graph. Consider the reverse orientation, resulting in an arc-disjoint strongly-connected graph S_2. Then we can pick a multicast tree from S_1 and backup paths from S_2.

There is a subtlety: the actual multicast tree is computed in a distributed fashion based on link weight assignments. Thus we need to find a set of link weight assignments that leads to the selection of a multicast tree in S_1. Because of space limitations, we omit the proof of how this is done.

In [14] the authors construct a multicast tree T and unicast routes disjoint from T. This can be done with the above construction by picking the multicast tree from S_1 and all unicast paths from S_2.

VII. Conclusion

We presented various network designs and backup path reconfiguration schemes for restoring multiple failures using FRR. These are the first designs with formal (worst case) guarantees of no congestion and no loss of connectivity. Although our proofs of correctness are complex, the descriptions of the designs and protocols are simple. Using a lower bound argument, we showed that one of our constructions is nearly optimal in the number of edges. A numerical evaluation showed that, e.g., a network graph with average degree 7 with 57% overbuild can support failures of up to 4 edges.

References

[1] J. Akpuh, J. Doucette. *Enhanced failure-specific p-cycle network dual-failure restorability design and optimization*. J. of Optical Networking, vol. 8 (1), 2009.

[2] C. Berge. *Graphs and Hypergraphs*, North-Holland, 1979.

[3] Y. Bejerano, P. Koppol. *Optimal construction of redundant multicast trees in directed graphs*. INFOCOM, 2009.

[4] J. Choi, S. Subramaniam, H. Choi. *On double-link failure recovery in WDM optical networks*. INFOCOM, 2002.

[5] R. Doverspike, G. Li, K.N. Oikonomou, K. Ramakrishnan, D. Wang. *IP Backbone Design for Multimedia Distribution: Architecture and Performance*. INFOCOM, 2007.

[6] M. Garg, J. Cole Smith. *Models and algorithms for the design of survivable multicommodity flow networks with general failure scenarios*. Omega, vol. 36 (6), 2008.

[7] A. Gerber, R. Doverspike. *Traffic types and growth in backbone networks*. OFC, Anaheim, CA, 2011.

[8] M. Goyal, K. Ramakrishnan, W. Feng, *Achieving faster failure detection in OSPF networks*. IEEE ICC, 2003.

[9] M. Grötschel, C.L. Monma, M. Stoer. *Polyhedral and Computational Investigations for Designing Communication Networks with High Survivability Requirements*. Operations Research, Vol. 43 (6), 1995.

[10] W. Grover, D. Stamatelakis. *Cycle-oriented distributed preconfiguration: ring-like speed with mesh-like capacity for self-planning network restoration*. ICC, 1998, pp. 537–543.

[11] A. Itai, M. Rodeh. *The multi-tree approach to reliability in distributed networks*. FOCS, 1984.

[12] S. Kim, S. Lumetta. *Evaluation of protection reconfiguration for multiple failures in optical networks*. OFC, 2003.

[13] S. Kini, D. Ramasubramanian, A. Kvalbein, S. Hansen. *Fast recovery from dual link failures in IP networks*. INFOCOM, 2009.

[14] G. Li, D. Wang, R. Doverspike. *Smart IGP weight setting in multimedia IP networks*. Mini Symposium of IEEE INFOCOM, 2007.

[15] C. Liu, L. Ruan. *p-cycle design in survivable WDM networks with shared risk link groups (SRLGs)*. J. Photonic Network Communications (3).

[16] T.L. Magnanti, S. Raghavan. *Strong formulations for network design problems with connectivity requirements*. Networks, vol. 45, 1999.

[17] J. Moy. *OSPF: Anatomy of an Internet routing protocol*. Addison Wesley, 1998.

[18] P. Pan, G. Swallow, A. Atlas. *Fast reroute extensions to RSVP-TE for LSP tunnels*. IETF RFC 4090.

[19] S. Ramasubramanian, A. Chandak. *Dual-link failure resiliency through backup link mutual exclusion*. IEEE/ACM Transactions on Networking 16 (1), 2008.

[20] A. Todimala, K. Ramakrishnan, R. Sinha. *Reconfiguration for surviving multiple-link failures in backbone networks*. OFC, 2008.

[21] www.cs.sfu.ca/~funda/DRCN/appendix.pdf

[22] J. Zhang, J. Zhu, B. Mukherjee. *Backup reprovisioning to remedy the effect of multiple link failures in WDM mesh networks*. IEEE JSAC 24 (8), 2006.

[23] M. Yuksel, K. Ramakrishnan, R. Doverspike, R. Sinha, G. Li, K.N. Oikonomou, D. Wang. *Cross-layer techniques for failure restoration of IP multicast with applications to IPTV*. COMSNET, 2010.

[24] Y. Wang, H. Wang, A. Mahimkar, R. Alimi, Y. Zhang, L. Qiu, R. Yang, *R3: Resilient routing reconfiguration*. SIGCOMM, 2010.

[25] Q. Zhu, Y. Wang. *Compressed Video over Networks*. Ming-Ting Sun and Amy R. Reibman, eds. Marcel Dekker, 2001.