Grafting Arborescences for Extra Resilience of Fast Rerouting Schemes

Klaus-Tycho Foerster, Andrzej Kamisinski, Yvonne Anne Pignolet, Stefan Schmid, Gilles Trédan

Grafting Arborescences for Extra Resilience of Fast Rerouting Schemes

Klaus-Tycho Foerster* Andrzej Kamisiński** Yvonne-Anne Pignolet‡ Stefan Schmid* Gilles Tredan◇
*Faculty of Computer Science, University of Vienna, Austria **AGH University of Science and Technology, Poland
‡DFINITY, Switzerland ◇LAAS-CNRS, France

***Abstract*—To provide a high availability and to be able to quickly react to link failures, most communication networks feature fast rerouting (FRR) mechanisms in the data plane. However, configuring these mechanisms to provide a high resilience against multiple failures is algorithmically challenging, as rerouting rules can only depend on local failure information and need to be predefined. This paper is motivated by the observation that the common approach to design fast rerouting algorithms, based on spanning trees and covering arborescences, comes at a cost of reduced resilience as it does not fully exploit the available links in heterogeneous topologies. We present several novel fast rerouting algorithms which are not limited by spanning trees, but rather extend and combine ("graft") multiple spanning arborescences to improve resilience. We compare our algorithms analytically and empirically, and show that they can significantly improve not only the resilience, but also accelerate the preprocessing to generate the local fast failover rules.**

I. Introduction

Communication networks have become a critical backbone of our digital society, as highlighted during the ongoing COVID-19 pandemic. As link failures become more likely with increasing network scale [1] and as even short disruptions of connectivity can cause severe degradation in service quality [2]–[4], it is important that networks detect such events quickly and reroute flows accordingly. This is hard to achieve using control plane mechanisms: reaction times in the control plane are known to be high [1]–[5], not only due to global state advertisements and re-computations performed by widely deployed distributed control plane schemes such as OSPF [6] and IS-IS [7], but also in emerging centralized schemes [8]–[10].

To meet their stringent availability requirements, most modern communication networks hence feature local fast rerouting algorithms in the data plane which typically operates at timescales several orders of magnitude shorter than the control plane [11], [12]: since rerouting decisions are local, failure recovery can in principle occur at the speed of packet forwarding. At the heart of such *fast-reroute (FRR)* mechanisms [13] lies the idea of *pre-computing* alternative paths at any node towards any destination. When a node locally detects a failed link or port, it can autonomously remove the corresponding entries from the forwarding table and continue using the remaining next hops for forwarding packets: a fast local reaction [14]. In FRR, the control plane is hence just responsible for *pre-computing* the failover paths; when a failure occurs, the data plane utilizes this additional state to forward packets. For example, many data centers use ECMP [15] (a data plane algorithm that provides automatic failover to another shortest path), WAN networks leverage *IP Fast Reroute* [16]–[18] or *MPLS Fast Reroute* [19] to deal with failures on the data plane, SDNs provide FRR functionality in terms of *OpenFlow fast-failover groups* [20], and BGP relies on BGP-PIC [21] for quickly rerouting flows, to just name a few.

However, while the local decision used by FRR enables fast reactions, it also introduces an algorithmic challenge: the failover behavior needs to be *pre-defined*, before the actual failures are known. Configuring FRR is hence particularly challenging under multiple and correlated failures [22]–[26] since rerouting decisions need to be taken without knowledge of the failures downstream. Indeed, and while there has been much research on the topic over the last years [13], some of the most basic algorithmic problems are still open [27].

This paper revisits the design of fast rerouting algorithms, aiming to provide a high resilience while using minimal assumptions on the required network functionality. In particular, we are interested in algorithms which do not require packet header rewriting during failover. Our work is specifically motivated by the observation that state-of-the-art solutions relying on spanning trees and arborescences, may result in a suboptimal resilience: in general, tree-based network decompositions can fail to exploit certain links which are vital for connectivity. We hence present several novel fast rerouting algorithms which are not limited by spanning trees, but rather extend and combine trees to improve resilience, a technique called *grafting* in botany.[1] We compare these algorithms analytically and empirically. In particular, we show that our approaches can significantly improve the resilience under different failure scenarios in comparison to the state of the art.

Contributions. We present three new fast-failover routing schemes that retain worst-case guarantees but also leverage network heterogeneity for non-adversarial failure scenarios:

- §III: `DAG-FRR` utilizes the insight that disjoint routing trees can be extended to multiple directed acyclic graphs, greedily selecting the next hop closest to the destination.
- §IV: `Cluster-FRR` is motivated by the observation that routing trees need not be rooted at the destination, but can also be formed in dense clusters, handing the packets over via exchange points (local roots) positioned at sparse cuts.
- §V: `Augment-FRR` benefits from the fact that disjoint routing trees work well on homogeneous topologies (e.g., regular topologies), in turn augmenting the network in this direction with virtual links. When routing, `Augment-FRR` treats these virtual links like link failures.

[1] Grafting is a technique to join two trees (and plants in general) into one. For example, many fruit trees today are grafted onto rootstock.

Our extensive evaluations in §VI showcase the resilience gains over prior work, for various failure models (see §VI-A). In this context, to guarantee reproducibility and to allow other researchers to build upon our work, we have made the source code of our implementation of the algorithms publicly available at https://gitlab.cs.univie.ac.at/ct-papers/fast-failover. Additionally, it is worth noting that our algorithms also perform well while dealing with a small number of failed links.

Moreover, we prove in §II that some state-of-the-art schemes already fail for a single link failure, even though the network remains highly connected, see §VII for further related work.

Model. We will use the following terminology and model.

Network structure. We model the communication network as a graph $G = (V, E)$, connecting $|V| = n$ nodes (routers, switches, hosts) with E links. While bidirected (respectively full-duplex symmetric) links are common in networks, our algorithms also extend to strongly connected directed graphs. In the latter case, we might also use the term *arc* to emphasize the directed nature of a link. In this paper, we often distinguish between *homogeneous* and *heterogeneous* networks: the terms refer to the connectivity, that is, whether the connectivity between every node pair is the same or whether it can differ.

Routing and failover. We consider the common scenario where the static forwarding rules have to be precomputed and deployed at the nodes before the (unknown) link failures occur. In particular, we do not allow any packet modification, dynamic forwarding entry changes, convergence, or randomization: the forwarding rules may only match on the incoming port[2], the destination $t \in V$, and the failure status of incident links. As such, FRR is purely local and comes into effect immediately.

II. Motivation: Limitations of the State of the Art

In order to motivate our perspective and approach, we first revisit the state-of-the-art approaches for implementing FRR in our setting. We first consider arborescence-based FRR in §II-A, which performs well under worst-case failure scenarios and in homogeneous topologies. We then discuss the approach by Yang et al. [28] in §II-B, which utilizes adapted greedy forwarding in heterogeneous network structures. Afterwards, we draw conclusions for new and improved strategies in §II-C.

A. Homogeneous Networks: Arborescence-Based FRR

The fundamental idea of arborescence-based FRR is to leverage arc-disjoint spanning trees (arborescences) for forwarding: should a link fail, another arborescence can be selected, where the current arborescence is identified from the incoming port due to arc-disjointness. In more detail, given a k-link-connected graph, k arc-disjoint spanning arborescences rooted at a destination t can be found efficiently [29]. By using, e.g., a global circular permutation of the arborescences [30], $k-1$ arc failures can be tolerated. An exemplary implementation is shown in Algorithm 1, using a given arborescence decomposition.

[2] Without using the incoming port, already very simple link failure scenarios lead to permanent routing loops, as sending a packet back is impossible.

Algorithm 1: Circular Routing for Arborescences

Upon receiving a packet destined for t at node v:

```
current arborescence T_l (determined by in-port, else T_1)
if destination not reached yet, t ≠ v then
    if next hop on T_l available then
        forward packet along T_l
    else
        update T_l to be the next arborescence from the
        circular arborescence list
```

From a worst-case perspective, resilience to $k-1$ arc failures is optimal, as k failures can disconnect a graph of connectivity k. Prior work utilized these worst-case guarantees to show benefits in highly homogeneous networks, such as for $k-1$ failures under k-connectivity [27], but also multiple random failures in k-regular k-connected graphs and well-connected cores of autonomous systems [31], [32]. On the other hand, the arborescence approach gives no guarantees beyond the network's connectivity, and as such scales poorly for heterogeneous networks, e.g., if the network is 1-connected.

B. Heterogeneous Networks: Keep Forwarding FRR

Yang et al. [28] identified an approach to leverage topology heterogeneity, summarized as "*Keep Forwarding*" (KF): essentially, when the packet cannot be brought closer (down) to the destination in a greedy fashion, it tours the nodes of identical distance, until a not-failed down-link appears. In the absence of down-links and neighbors of identical distance, the packet takes a hop away from the destination, in order to find alternate routes. To avoid small loops, and as the only implicit memory of the forwarding function, the incoming port is only picked as the last resort, when all other incident links failed.

Yang et al. [28] show in their evaluations for 1, 2, and 3 link failures that KF retains over 99% reachability for heterogeneous autonomous systems and data center networks. Notwithstanding, no formal guarantee is claimed, not even for a single failure.

In the following, we show that KF already fails after a single failure, even if the remaining network remains highly connected. Consider the graph in Fig. 1, where the packet starts at node v_1 and has to be routed to the destination t. The next hop will be via the only down-link to v, but the link (v, t) has failed, resulting the packet to be forwarded up to some other node from $v_2, \ldots, v_k$. While KF retains the knowledge not to route back via the inport to v, even though that is the only down-link, the next hop will be to another node from $v_1, \ldots, v_k$. However, then the knowledge not to send the packet to v is lost (as the only memory is via the incoming port), and the packet is again forwarded to v, inducing a permanent forwarding loop. Moreover, observe that the network's connectivity remains at $k \in \Omega(|V|)$, i.e., arborescence-based FRR easily reaches the destination.

C. Reaping the Benefits of Both Approaches for New FRR

As we discussed above, arborescences have shortcomings when applied in heterogeneous networks, as they only utilize

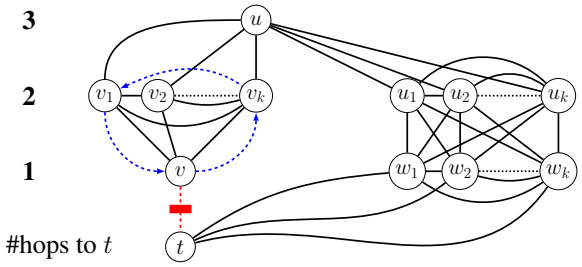

Fig. 1. Example where *Keep Forwarding (KF)* provides no resilience even though only one link failed. When starting on v_1, KF will forward down (closer) to v, where however the link (v,t) has failed. In turn, KF forwards up (away) from v, to, e.g., v_k. As the incoming port is the last choice, KF now forwards to a node of identical distance, e.g., v_1 (or any other node from $v_1,\ldots,v_{k-1}$), again forwarding down. Hence, KF loops as it forwards down to v, up, within $v_1,\ldots,v_k$, down to v, up again etc. Note that the remaining connectivity is $k \in \Omega(|V|)$ and that only one link failed.

the (worst-case) global connectivity. While *Keep Forwarding* leverages heterogeneity, by traversing the current cluster until progress can be made, it does not provide any formal resilience guarantees.

We would like to combine the benefits of both approaches: survivability w.r.t. worst-case failures, but also retain survivability against more failures in dense heterogeneous clusters. To this end, in the remainder of this paper, we extend the known arborescence-based approaches by incorporating non-spanning (partial) arborescences and local greedy routing, using them when classic arborescences fail. In this fashion, we retain the worst-case guarantees, but also utilize the path diversity in heterogeneous settings. We start in Section §III with a first approach that combines partial arborescences with directed acyclic graphs, followed by a more intricate arborescence construction in the following section.

III. DAG−FRR: Leveraging Heterogeneity via DAGs

The idea of using directed acylic graphs (DAGs), instead of a forwarding tree (or arborescences), has already been proposed in [33], [34] (note however that these proposals do not fit into a purely local model, as they use message exchanges [34] or link reversals [33]): when the link to the next hop fails, the current node might have alternate next-hops, globally organized in a way such that no routing loops occur.

However, when only a single DAG is chosen, at least half of the available arcs (directed links) may not be used for routing: choosing both directions of a link for routing is a contradiction to the DAG property. For a simple example, consider routing on a ring topology: routing on a DAG already stops working after a single link failure, whereas arborescences can survive the same link failing. On the other hand, spanning arborescences perform poorly when the underlying connectivity is one. Here a DAG can contain a quadratic number of more links in the extreme case, which the arborescence leaves unused.

In this section we hence focus on combining DAGs and arborescences, to provide the best of both worlds. Instead of generating DAGs from scratch, we propose to extend arborescences to DAGs, meaning that our method can be understood as an extension of any arborescence structure, e.g., [27], [32], [35], [36]. We emphasize that the underlying arborescences do not need to be spanning (as in prior work), partial rooted structures can be enhanced as well.

Our approach, henceforth called DAG−FRR, thus consists of three parts: §III-A describes how to build the partial arborescences, §III-B how to extend the arborescences, and §III-C how to extend the routing itself. We discuss in §III-D implications w.r.t. the worst-case resilience of DAG−FRR.

A. Part 1: Building Rooted Partial Arborescences

Prior work focused on building k arc-disjoint rooted spanning arborescences in k-connected graphs, even if the destination has a degree of $k' \gg k$. As such, in the common scenario that the network has a well-connected core that gets sparser in its outskirts, exactly the low connectivity of the outer regions defines the global survivability.

We overcome this restriction by building as many rooted arborescences as the destination can support by its number of neighbors. While not each of these arborescences can reach all nodes (i.e., be spanning), already a greedy approach will guarantee that every node is contained in at least one arborescence: else, there must be at least one node v, not in any arborescence, neighboring a node w in an arborescence T, a contradiction, as T can be extended from w to v.

We hence propose the following simple decomposition algorithm. We assign each of the k' incoming arcs of the destination to a different arborescence $T_1,\ldots,T_{k'}$, and grow them greedily one after another, switching to the next arborescence T_{i+1} if no more nodes can be added to T_i. Notwithstanding, this could leave many arcs unassigned to any arborescence, which we cover in the next subsection via arborescence extension.

B. Part 2: Extending Arborescences to DAGs

The underlying idea of our extension process is to utilize the so-far unused links and add them to the arborescence routing structures. As an input, we require a set of arborescences, or generalized, DAGs, and proceed as follows: For each DAG D_i, we add unused arcs e to it, as long as: 1) D_i with e is cycle-free, and 2) e connects to $V(D_i)$.

We hence check all arcs if they can extend the given DAGs with the above two properties, repeating the process if some DAGs increased in size while doing so, see Algorithm 2.

C. Part 3: Extending Arborescence Routing

A fundamental strength of arborescence routing is that by considering the inport on which a packet arrives global information about the packet's journey so far can be inferred: the current node immediately knows which arborescence the packet is currently routed on, even though the packet remains unchanged. We thus strive to retain a packet in its current arborescence, which is easy in the case of DAGs: due to loop-freedom, at each node, each outgoing link of the current arborescence-DAG brings the packet closer to the destination.

A natural routing extension herein is to choose a surviving link that reduces the distance to the destination the most[3], and

[3] Though any choice from the set of outgoing links is viable.

Algorithm 2: DAG Extension Algorithm

Input: Directed graph $G = (V, E)$, arc-disjoint arborescences or DAGs $\mathcal{D} = \{D_1, \ldots, D_k\}$
Output: A maximal set $\mathcal{D}$ of arc-disjoint DAGs

```
1: extended←true
2: while extended=true do
3:    extended←false
4:    for all D_i ∈ D do
5:       for all e = (u, v) ∉ D do
6:          if v ∈ V(D_i) then
7:             if E(D_i) ∪ {e} is cycle-free then
8:                E(D_i) ← E(D_i) ∪ {e}
9:                extended←true
```

only switch to the next arborescence-DAG when all outgoing links of the current arborescence-DAG have failed. Note that the remaining routing logic stays intact and hence the routing extension can be implemented conceptionally as a simple subrouting, respectively an extension of the priority list of outgoing ports. We formalize this in Algorithm 3.

Algorithm 3: Circular Routing for DAGs

Upon receiving a packet destined for t at node v:

```
1: current DAG D_l (determined by in-port, else D_1)
2: if destination not reached yet, t ≠ v then
3:    if a next hop on D_l available then
4:       forward packet along outgoing arc in D_l that reduces
         distance to t the most
5:    else
6:       update D_l to next DAG from circular DAG list
```

D. Part 4: Worst-Case Resilience Discussion

Prior work already established that circular routing on k arc-disjoint rooted spanning arborescences is resilient to $k-1$ arc failures [30]. The underlying reasoning is that after $k-1$ failures, at least one arborescence will be intact, i.e., the packet will eventually reach the destination. The invariant still holds for k arc-disjoint rooted spanning DAGs: by fixing a global circular order on the DAGs, one will eventually reach a failure-free DAG. We cast our observation into the following theorem:

Theorem 1. *Starting on k arc-disjoint rooted spanning arborescences/DAGs, `DAG-FRR` is resilient to $k-1$ arc failures.*

We note that in §III-A we advocated for building partial rooted decompositions, instead of enforcing (fewer) spanning ones, to better leverage the network's heterogeneity, and might hence lose out on worst-case resilience. On the other hand, our `DAG-FRR` is flexible and can also start by employing a network decomposition that uses k arc-disjoint rooted spanning arborescences on k-connected topologies [29].

IV. CLUSTER-FRR: LOCAL CLUSTER ALGORITHM

We now describe a more sophisticated arborescence decomposition algorithm, called `Cluster-FRR`. It applies a generic rooted spanning arborescence decomposition, e.g., the greedy algorithm also used in [30]. The algorithm is based on local clusters: nodes having a positive clustering coefficient[4] may form local regions of higher link connectivity relative to the entire graph. Here, algorithms computing the set of arc-disjoint spanning arborescences covering the graph would only return so many trees as is the value of the link connectivity of the graph, while the potential of the highly-connected regions would remain under-utilized. Indeed, by constructing additional arc-disjoint trees covering the highly-connected regions and using them to extend the primary set of arborescences, our algorithm (see Algorithm 4) is able to improve the fast recovery capabilities of a network in the case of multiple failures.

A. `Cluster-FRR` Arborescence Generation

The algorithm starts by computing a spanning arborescence decomposition of G rooted at the destination. Next, the clustering coefficient is determined for all nodes. Nodes for which the corresponding value is greater than zero are marked, together with their direct neighbors. Based on the previously marked nodes and the existing links between them, the subgraph G_{m} of G is also created. Note that it is not required that subgraph G_{m} be connected — in fact, as it comprises nodes and links belonging to each of the highly-connected regions of graph G, the next action is to decompose G_{m} into strongly-connected components and store them in set S_{m}. The output set T_{out} is also initialized to contain the arc-disjoint spanning arborescences stored in set T. When the graph is k-connected, T_{out} at this point contains k arc-disjoint destination-rooted spanning arborescences $T_1, T_2, \ldots, T_k$.

Then, the algorithm considers each of the connected components G_{cc} individually, ignoring those that contain less than three nodes. First, to improve the local link connectivity of G_{cc}, all nodes of degree 1 are removed from G_{cc} in a series of subsequent iterations of the *while* loop, until no more such nodes can be found in the graph. If the resulting graph contains less than three nodes or its link connectivity is less than two, G_{cc} is rejected and the next connected component is considered instead. Second, the local root node r_{cc} of the expected additional arborescences is selected in G_{cc} based on the shortest distance to r in the original graph G. The arc-disjoint spanning arborescences rooted at r_{cc} and covering G_{cc} are then determined — the result is stored in set T_{cc}. Next, the algorithm adds to set A all arcs of G_{cc} that are included in one of the additional arborescences stored in T_{cc}, except for the arcs being part of the primary arborescences already included in the output set T_{out}. The main reason behind this selection process is to guarantee that no arc is assigned to multiple arborescences rooted either at r or r_{cc}; thus, that a failure of a single network link can disable at most two different arborescences associated with the related pair of opposite arcs. Each arc in set A is also

[4] For the formal definition of the clustering coefficient in the context of graphs, the reader is referred to [37].

Algorithm 4: Cluster-Based Forest Construction

Input: Directed graph $G = (V, E)$, destination $r \in V$
Output: A set T_{out} of rooted arc-disjoint spanning arborescences and additional directed acyclic subgraphs of G

1: $T \leftarrow \text{ArborescenceDecomposition}\,(G, r)$
2: **for all** $v \in V$ **do**
3: $\quad c_v \leftarrow \text{ClusteringCoefficient}\,(G, v)$
4: $\quad$ **if** $c_v > 0$ **then**
5: $\quad\quad$ Mark node v (not marked by default): $m_v \leftarrow 1$
6: $\quad\quad$ Mark all neighbors of v: $\forall_{u \in \text{Neighbors}(G,v)} m_u \leftarrow 1$
7: $G_{\text{m}} \leftarrow$ Subgraph induced by the marked nodes
8: $S_{\text{m}} \leftarrow \text{StronglyConnectedComponents}\,(G_{\text{m}})$
9: $T_{\text{out}} = T, \;\; A_{\text{count}} = 0$
10: **for all** $G_{\text{cc}} = (V_{\text{cc}}, E_{\text{cc}}) \in S_{\text{m}}$ **do**
11: $\quad$ **while** $|V_{\text{cc}}| > 3$ **and** $\text{MinDegree}\,(G_{\text{cc}}) < 2$ **do**
12: $\quad\quad$ Remove all nodes of degree 1 from G_{cc}
13: $\quad$ **if** $|V_{\text{cc}}| < 3$ **or** $\text{LinkConnectivity}\,(G_{\text{cc}}) < 2$ **then**
14: $\quad\quad$ Proceed to the next iteration
15: $\quad r_{\text{cc}} \leftarrow \text{GetClosestNodeToDestination}\,(G_{\text{cc}}, r)$
16: $\quad T_{\text{cc}} \leftarrow \text{ArborescenceDecomposition}\,(G_{\text{cc}}, r_{\text{cc}})$
17: $\quad A \leftarrow \emptyset$
18: $\quad$ **for all** $(a, b) \in E_{\text{cc}}$ **do**
19: $\quad\quad$ **if** (a, b) not included in any graph in T_{cc} **or** (a, b) already included in a graph in T_{out} **then**
20: $\quad\quad\quad$ Proceed to the next iteration
21: $\quad\quad id \leftarrow \text{GetArborescenceId}\,(T_{\text{cc}}, (a, b))$
22: $\quad\quad A \leftarrow A \cup (id, (a, b))$
23: $\quad s \leftarrow$ The number of unique arborescence identifiers among the elements of A
24: $\quad$ Index of an extra graph: $j = 1$
25: $\quad$ **for all** identifiers id among the elements of A **do**
26: $\quad\quad$ **for all** (a, b) associated with id in A **do**
27: $\quad\quad\quad$ Current graph identifier: $q \leftarrow \text{LinkConnectivity}\,(G) + A_{\text{count}} + j$
28: $\quad\quad\quad$ Add (a, b) to the q-th graph in T_{out}
29: $\quad\quad j \leftarrow j + 1$
30: $\quad A_{\text{count}} \leftarrow A_{\text{count}} + s$
31: **return** T_{out}

associated with the unique numeric identifier of the related arborescence. Finally, in lines $23 - 28$, the algorithm groups arcs associated with the same tree and transfers the resulting groups from A into the output set T_{out}. Both the primary arborescences and the newly added groups of arcs are assigned unique numeric identifiers from a contiguous range (the primary arborescences have the lowest indices starting from 1).

B. `Cluster-FRR` Routing and Resilience

For k-connected graphs, `Cluster-FRR` generates a destination-rooted spanning arborescence decomposition $T_1, T_2, \ldots, T_k$, along with further non-spanning partial arborescences, with local root nodes and identifiers greater than k. `Cluster-FRR` then employs the circular arborescence-based routing scheme from Algorithm 1, starting in the arborescence with the lowest identifier. Hence, `Cluster-FRR` retains good worst-case guarantees: after up to $k - 1$ arc failures, at least one of the primary spanning arborescences $T_1, T_2, \ldots, T_k$ remains intact and can be used to route to the destination. In other words:

Theorem 2. *On k-connected graphs, `Cluster-FRR` is resilient to $k - 1$ arc failures.*

After more than $k - 1$ arc failures, `Cluster-FRR` might need to resort to the local partial arborescences, and we can hence no longer provide the worst-case guarantees. On the other hand, the increased local survivability leads to greater resilience to non-adversarial failures, as we will see later in §VI.

V. Augment-FRR: Connectivity Augmentation

So far, we proposed two methods that start with *few* arborescences, and then incorporate the remaining links, either by means of DAGs in §III or by building small local arborescences in §IV. In this section, we take a fundamentally different approach and start with *many* arborescences.

To this end, we augment the graph with virtual links to extend large local connectivity to the whole graph, build arborescences on top, and then remove these virtual links from the arborescences (hence they are no longer spanning) and let standard failover routing take over. In other words, we leverage that arborescences work well on homogeneous topologies.

In the following, we first describe how to augment the network efficiently to the desired level in §V-A and then cover in §V-B how to build the arborescences in a way that takes the distinction between virtual and real links into account.

A. Turning a Network Homogeneous

In order to maximize the efficiency of arborescence routing, a first step would be to obtain a network that is minimally k-connected, i.e., the removal of any bidirected[5] link reduces the connectivity below k. In such networks, a rooted spanning arborescence decomposition can include every arc, except for the ones outgoing from the root.

A simple way of achieving this goal would be to turn the network into a clique K_n, which however raises the question of how many virtual links we should include. On the one hand, we want to cover as many real arcs as possible with our augmentation, on the other hand, we want to include only few virtual links, as they disrupt the routing behavior.

To this end, we pick the node with the highest degree Δ in the network (as each arborescence, for each node $v \neq t$, can only cover one outgoing arc of v), and augment the network to be Δ-connected. For bidirected graphs, this is a well-studied problem, and selecting the minimum number of links can be solved efficiently [38].[6]

Hence, to summarize, `Augment-FRR` selects the largest degree Δ in the network and then turns the network to be Δ-connected with a minimum number of virtual links.

[5] The problem can be defined analogously for directed networks, but we focus on bidirected full-duplex links given their prevalence in most networks.
[6] We refer to Frank [38] for a discussion on further (directed) model variants.

B. Building Arborescences on Virtual Links

Building Δ arc-disjoint rooted spanning arborescences on Δ-connected graphs is well understood and can be computed efficiently [29]. However, in our context, we have to consider how to distribute real and virtual links over the arborescences.

`Augment-FRR` follows an approach that preserves worst-case resilience. To this end, it takes a Δ-connected network as the input, and then runs an greedy rooted spanning arborescence decomposition [30] adapted to the problem at hand as follows: when growing arborescence T_i, (1) a candidate arc a is only added if the remaining network, without $T_1, \ldots, T_i$ and without a is still be $\Delta - i$-connected, and (2), when multiple arcs can be selected for growing the arborescence T_i, arcs belonging to real links are strictly preferred over virtual ones.

C. `Augment-FRR` Routing and Resilience

`Augment-FRR` utilizes circular arborescence routing from Algorithm 1 on the obtained decomposition, treating virtual links as link failures, in turn obtaining high resilience:

Theorem 3. *`Augment-FRR` achieves resilience to $k-1$ arc failures on k-connected graphs.*

Proof: As proven in prior work [30], using circular arborescence routing on k-arc disjoint t-rooted spanning arborescences achieves resilience to $k-1$ arc failures. It is left to show that `Augment-FRR` retains these worst-case guarantees. Observe that when building the arborescences, `Augment-FRR` leverages the greedy algorithm from Chiesa et al. [30] in such a way, that real links always have preference over virtual links. Hence, as the greedy algorithm builds k arc-disjoint t-rooted spanning arborescences on the non-augmented network, the greedy algorithm's output behavior will be identical for the first k arborescences on the augmented network. As such, virtual links will only be included from the $k+1$ arborescence on.

As $k-1$ arc failures leave at least one of the first k arborescences unharmed, circular routing starting on arborescence T_1 will hence allow `Augment-FRR` to reach the destination. ■

VI. Evaluation

To evaluate our approach and algorithms, we conducted extensive simulations. In general, whether the network will be able to restore the connectivity between any pair of nodes in the event of one or more link failures, depends on the structure of the network graph, and its link connectivity in particular.

In the following, we hence first present different link failure models for which we will evaluate our algorithms, and also describe the experimental setup. We then report on our main insights from our evaluation.

A. Link Failure Models

To investigate how the proposed solutions perform in different failure scenarios, we considered two different link failure models, denoted as RANDOM and CLUSTER.

Random Link Failures (RANDOM). The first considered model introduces link failures uniformly at random across the entire network. It assumes that each link in the network has an equal probability of failure, and that none of the observed link failures result from targeted actions taken by malicious actors. Thus, in this model, we do not focus on how to take the maximum advantage of local redundancy — instead, we investigate how the network deals with multiple link failures in the general case. We also assume that each failure affects the corresponding network link in both directions.[7]

Targeted Attacks (CLUSTER). In this failure model, we try to capture more adversarial failures, e.g., due to an attack. We imagine that clusters in a network graph may represent well-connected regions, e.g., associated with groups of cities of strategic importance. In this context, the potential adversary might be interested in carrying out targeted and coordinated attacks against links belonging to one or more clusters, to degrade the fast-recovery capabilities of the network in the affected areas. Indeed, despite the relatively high connectivity in such areas, it is still possible that targeted link attacks will disrupt some of the precomputed or predefined primary and backup paths used by many data flows [40]. Thus, the related challenge for the investigated fast-recovery algorithms is how to use the local redundancy in those regions to respond to targeted attacks effectively.

According to the considered model, failed links are selected as follows. First, the clustering coefficient is computed for all nodes in the network, and then, each arc incident to a node with a non-zero value of the clustering coefficient is added to the set F_{cand} of candidate arcs that might be disabled, unless the set already contains the opposite arc. Second, as the number of failed bidirectional links is limited by one of the simulation parameters, f_{num}, the following two cases are considered:

- $|F_{\text{cand}}| > f_{\text{num}}$: if the set of candidate arcs contains more elements than the value of f_{num}, select f_{num} arcs uniformly at random from set F_{cand} and disable the corresponding bidirectional network links;
- $|F_{\text{cand}}| \leq f_{\text{num}}$: if the set of candidate arcs contains no more elements than f_{num}, disable all bidirectional network links corresponding to arcs included in F_{cand}.

B. Experimental Setup and Metrics

Each of our experiments was performed based on the following general scheme, proceeding in five steps:

1) Given the topology of a network, a destination node t was selected uniformly at random from the set of all nodes.
2) Then, the precomputations to construct the routing tables were performed for each of the investigated algorithms and the time for these computation was measured separately for each algorithm.
3) In the next step, f_{num} undirected links were selected as the failed network links based on one of the link failure models discussed in §VI-A. In addition, `sample_size` source nodes were chosen uniformly at random from the set of all nodes in the graph excluding the destination.

[7] We note that it would also be interesting to consider unidirectional link failures. For example, a recent data center study found that "*only 8.2% of the links [...] with packet corruption had bidirectional corruption*" [39].

4) Then, the routing function was executed on the graph with f_{num} failed links. In some cases, the source nodes may no longer have been connected to the destination.
5) Finally, the success rate (defined as the number of data flows reaching the destination, divided by the number of all data flows) was computed separately for the investigated algorithms. The *expected success rate for a perfectly resilient algorithm* ρ is thus $(n_{cc}-1)/(n-1)$ where n_{cc} denotes the number of nodes in the connected component of the destination after failures.

C. Routing Success in Highly Heterogeneous Networks

We first investigate the performance of our algorithms in highly heterogeneous networks in which connectivity varies significantly. To this end, we consider the following synthetic *ring-of-cliques* network topology model which contains several well-connected regions. Specifically, we assume that the network topology consists of L interconnected cliques (see Fig. 2), each having an internal link connectivity of k_{clique}.

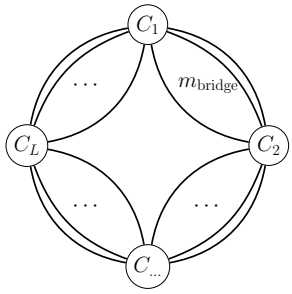

Fig. 2. Illustration of the ring-of-cliques network topology model. Each of the cliques $C_1, C_2, \ldots, C_L$ has an internal link connectivity k_{clique} and there are m_{bridge} links between adjacent cliques.

Two adjacent cliques are connected by m_{bridge} links incident to randomly chosen nodes on the two involved cliques, forming a graph for which the overall link connectivity is $k_{\text{graph}} \geq \min(k_{\text{clique}}, 2m_{\text{bridge}})$. To investigate the effectiveness of the considered algorithms in terms of their ability to deal with simultaneous failures of multiple network links, we further assume that $k_{\text{clique}} > k_{\text{graph}}$ (the internal link connectivity of cliques is higher than the link connectivity of the entire graph).

In our first experiment we vary the number of failed links the algorithms have to cope with and study their success rate. For randomly created rings of 10 cliques with $k_{\text{clique}} = 9$ and $m_{\text{bridge}} = 2$, i.e., graphs with 100 nodes and 470 links we observe the following behavior in Fig. 3:

Under randomly chosen failures the three proposed algorithms are able to sustain around 100 link failures without any routing failures, while the pure Greedy Arborescence Decomposition and Keep Forwarding approaches exhibit routing problems already for as few as 10 failed links. When the number of failed links exceeds 100, the number of nodes in the connected component of the destination is below n, yet on average `DAG-FRR`, `Cluster-FRR` and `Augment-FRR` achieve a success rate of ρ, the expected success rate of a perfectly resilient algorithm for randomly chosen sources.

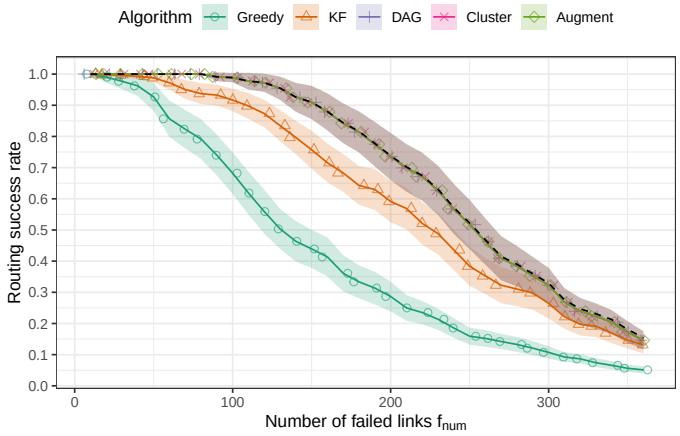

Fig. 3. Routing success rates on the ring of cliques for varying number of RANDOM failures and for different algorithms. The black dashed line represents the average value of ρ, the percentage of the number of nodes in the connected component containing the destination (after failures), and is therefore a statistical upper bound on the success rate. Ribbons represent 1/4th of the standard deviation over 200 independent repetitions.

When choosing the failed links with the cluster failure approach, see Fig. 4, Greedy and KF perform even worse. While KF gets stuck in local sinks as in Fig. 1, the Greedy method cannot leverage the local link diversity. On the other hand, our algorithms `DAG-FRR` and `Cluster-FRR` still reach ρ, while `Augment-FRR` is a bit below them on average.

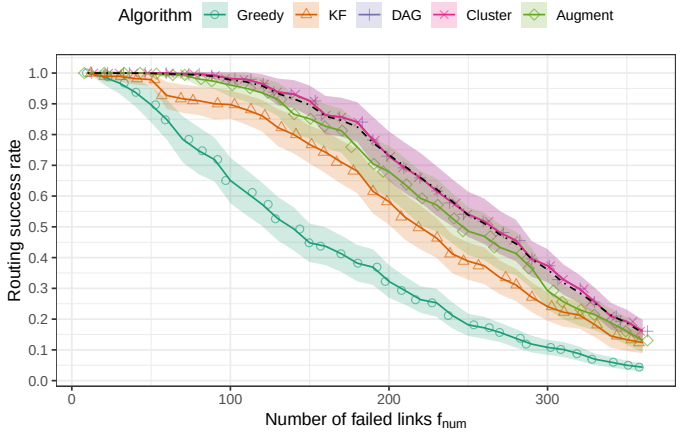

Fig. 4. Routing success rates on the ring of cliques for varying number of CLUSTER failures and for different algorithms. The black dashed line represents the average value of ρ, the percentage of the number of nodes in the connected component containing the destination (after failures), and is therefore a statistical upper bound on the success rate. Ribbons represent 1/4th of the standard deviation over 200 independent repetitions.

The reason is due to how `Augment-FRR` chooses the roots of partial non-spanning arborescences. Whereas `Cluster-FRR` picks them according to the distance to the destination, the corresponding choice for `Augment-FRR` is driven by the underlying connectivity augmentation algorithm, which takes this distance orientation only by a lesser degree into account. Similar in result to `Cluster-FRR`, `DAG-FRR` points

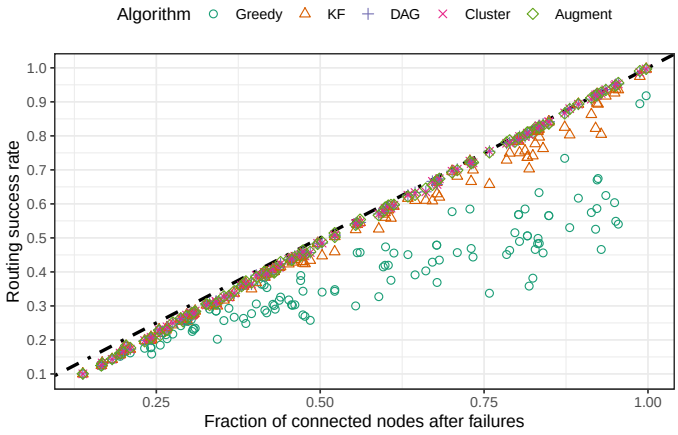

Fig. 5. Routing success rates on the Topology Zoo dataset, as a function of the fraction of nodes remaining connected after 10 RANDOM failures. The black dashed line represents the average value of ρ, the percentage of the number of nodes in the connected component containing the destination (after failures), and is therefore a statistical upper bound of the success rate. A point (x, y) indicates that for a network topology on which 10 random link failures disconnect an average fraction of $1 - x$ nodes, the corresponding algorithm achieved a routing success rate of y.

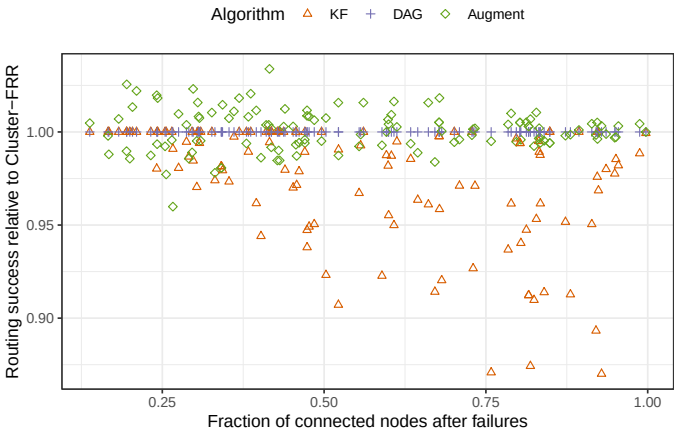

Fig. 6. Relative routing success rates of algorithms on the Topology Zoo dataset, for all graphs containing between 20 and 50 nodes, as a function of the fraction of nodes remaining connected to the destination after 10 RANDOM failures, compared to `Cluster-FRR`.

the additional arcs towards nodes that are closer to the destination, w.r.t. distance in routing along the current directed acyclic graph, and hence also performs well here.

D. Routing Success on Real-World Networks

We next evaluate the performance of our algorithms for the real-world network topologies collected as part of the Internet Topology Zoo project[8] Since the number of nodes and links in these graphs as well as their local and global connectivity varies considerably, we restrict ourselves to the subset of 122 topologies with 20 to 50 nodes and study the routing performance under random link failures.

To avoid cases where the destination is not connected to a significant number of nodes, we select them from the largest connected component after link failures. In Fig. 5 we present the success rate of the routing algorithms with respect to the percentage of nodes that are in the connected component of the destination after 10 random failures. We observe that `DAG-FRR`, `Cluster-FRR`, and `Augment-FRR` achieve a success rate of almost ρ, the expected success rate of a perfectly resilient algorithm for randomly chosen sources, while Greedy and Keep Forwarding perform less well.

Figure 6 provides a detailed look at the relative performance of our algorithms. It shows, for each Topology Zoo graph, the routing success rate of each algorithm normalized by the routing success of `Cluster-FRR` in the same graph. This allows for a more precise comparison of our algorithms: a point above $y = 1$ translates that the algorithm under scrutiny performed better than `Cluster-FRR` on the corresponding graph, whereas a point below translates a routing success rate under the one of `Cluster-FRR`. We omit Greedy for readability, and `Cluster-FRR` as all its corresponding datapoints lie on the $y = 1$ line by construction.

[8] http://www.topology-zoo.org

Interestingly, `DAG-FRR` performs precisely identically to `Cluster-FRR` (all data points on $y = 1$). KF always performs worse than `Cluster-FRR`, especially on lightly impacted topologies ($x > 0.5$). `Augment-FRR` performs differently than `Cluster-FRR`, but with overall the same performance.

The gap between `Augment-FRR` and `Cluster-FRR` performances increases on heavily impacted topologies. The small amplitude of these differences can be explained by the performance of these algorithms: even on highly affected topologies, `Augment-FRR`, `Cluster-FRR` and `DAG-FRR` all rarely fail to route in the remaining connected component. As before, the underlying augmentation process of `Augment-FRR` explains its variance, but as we see here, there is a good set of topologies where `Augment-FRR` also performs better than all other algorithms.

E. Runtime to Compute Routing Tables

In order to evaluate how the precomputation time complexity of the algorithms scales in terms of the number of nodes in the network, we consider the ring-of-cliques network which is defined for any number of nodes by changing the number of cliques from 3 to 20 with 5 nodes per clique.

Figure 7 plots the runtime of the different algorithms, specifically showing the average wall clock time. Our results show that for all algorithms, the routing tables in networks with up to one hundred nodes can be precomputed in less than twenty seconds, rendering these algorithms practical in many scenarios. In fact, except for `Augment-FRR` which is significantly slower, all algorithms even complete in less than two seconds — an overhead we deem bearable, especially for infrequent updates. KF is significantly faster. However, as we have seen above, this comes at a price of reduced resilience.

VII. Related Work

Failures are common in ISP networks [41], [42], cloud provider WANs [43], and datacenters [1], [44]. The design of fast rerouting algorithms has already been studied intensively

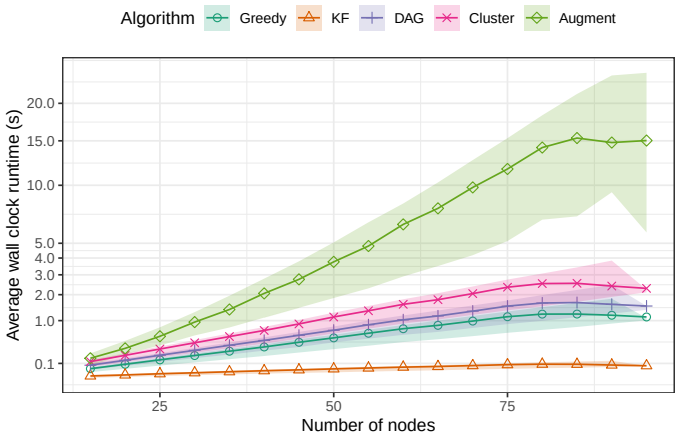

Fig. 7. Average wall clock times for the precomputation of the routing tables of each algorithm on networks of increasing number of nodes. The shaded area symbolizes the 10 and 90 percentiles of the distribution sampled over 80 independent runs. The y-axis is square-rooted.

in the literature. While there exists much interesting work on how to efficiently react to link failures in the control plane [45], [46], e.g., using fast reconvergence techniques [47], link reversal schemes [48], [49] or leveraging centralized approaches arising in software-defined networks [48]–[51], the reaction times provided by these solutions are significantly higher compared to fast failover mechanisms in the data plane [52]: the focus of our paper. Data plane-based failover mechanisms are typically used as a first line of defense and also well-explored in the literature; implementations exist for most relevant protocols, IP [53], [54] (including Segment Routing [55]–[57]), MPLS [19], [58], BGP [59], SDN and programmable data planes [20], [60], among many more. We refer the reader to the recent survey by Chiesa et al. [13] for a literature overview on fast recovery schemes in the data plane.

Fast failover algorithms for the data plane can be categorized according to whether they require dynamic packet headers modifications (or even dynamic state at the routers), and according to which header fields a router needs to be able to match. By rewriting packet headers, it becomes possible to, e.g., carry failure information in the packets [26], [61] or to employ classic graph exploration algorithms [62], [63] as well as rotor router approaches [64], which can be exploited to render networks more resilient. However, this is often impractical, and is generally not supported. It is also known that in scenarios in which not only the destination field but e.g., also the source or the inport can be matched, the design of resilient fast failover algorithms is greatly simplified [27], [30], [65]. We also note that the focus in this paper is on *deterministic* algorithms, which, in contrast to related work such as [66], [67], do not require random number generators at routers.

There already exist several interesting results on fast failover algorithms which do not require packet header rewriting. Feigenbaum et al. showed [12] that it is not always possible to achieve "perfect resilience" in this scenario, that is, it is not always possible to locally reroute packets to their destination even if the underlying network is remains connected after the failures; similar observations were obtained in parallel by Borokhovich and Schmid [68] who also derive bounds on what can be achieved in terms of load-balancing in such scenarios. Foerster et al. [69] recently generalized the impossibility results related to connectivity by establishing a connection between graph minors and resilience; in particular, the authors showed that perfect resilience cannot be achieved on any non-planar graph, but is at least possible on outerplanar graphs.

The state-of-the-art approach to design highly resilient failover algorithms is based on arc-disjoint arborescence covers and is due to Chiesa et al. [27], [30], [66]. This approach generalizes the widely-used approaches based on spanning trees [70]. However, while this approach may work well on graphs which are homogeneously k-connected (this is still an open question), it is not well-suited if directly applied to the general setting considered in our paper. In contrast, we in this paper have shown how to extend these concepts to more general and more heterogeneous networks.

The work closest to ours is by Yang et al. [28], who aim to go beyond the limitations of spanning trees and acyclic graphs by introducing what they call the partial structural network model. The authors show that this model indeed has several interesting features in practice. However, while their simulations look promising, the approach fails under worst-case failure scenarios. In particular, we have shown in this paper that even if the remaining connectivity is very high (linear in the number of nodes), a single arc failure suffices for their algorithm to fail. In contrast, our algorithms retain worst-case guarantees from arborescence approaches and can maintain $k-1$ arc failures, tolerating adversarial link failures.

VIII. Conclusion

This paper has studied how to extend existing fast failover mechanisms based on spanning trees and arborescences to improve the resilience of more heterogeneous networks whose connectivity varies across the topology. In particular, we presented three novel data plane algorithms which outperform state-of-the-art solutions in that they allow to maintain connectivity under significantly more failures.

From the three algorithms, our cluster and DAG algorithms perform relatively identical throughout our simulations, with our connectivity augmentation algorithm showing some slight variance in comparison. Due to our DAG algorithm having the fastest routing table precomputation time, we recommend it as a general choice amongst the three, but note that the augmentation algorithm sometimes performs slightly better, and hence a brief simulation for the intended topology might be worthwhile. To this end, and in order to guarantee reproducibility and allow other researchers to build upon our algorithms, we have made our code publicly available together with this paper.

We believe that our work opens several interesting avenues for future research. In particular, it remains to explore the resilience of our algorithms in more specific failure scenarios, both analytically and empirically. It would also be interesting to explore whether there are opportunities to improve our approach with randomization.

Acknowledgements: Research supported by the Vienna Science and Technology Fund (WWTF), grant ICT19-045, 2020-2024.

REFERENCES

[1] P. Gill *et al.*, "Understanding network failures in data centers: measurement, analysis, and implications," in *SIGCOMM*. ACM, 2011.

[2] M. Alizadeh *et al.*, "Data center TCP (DCTCP)," in *SIGCOMM*. ACM, 2010.

[3] B. Vamanan *et al.*, "Deadline-aware datacenter tcp (D2TCP)," in *SIGCOMM*. ACM, 2012.

[4] D. Zats *et al.*, "Detail: reducing the flow completion time tail in datacenter networks," in *SIGCOMM*. ACM, 2012.

[5] P. François *et al.*, "Achieving sub-second IGP convergence in large IP networks," *ACM CCR*, vol. 35, no. 3, pp. 35–44, 2005.

[6] J. Moy, "OSPF version 2," *RFC*, vol. 2328, pp. 1–244, 1998.

[7] ISO, "Intermediate Ststem-to-Intermediate System (IS-IS) Routing Protocol," ISO/IEC 10589, 2002.

[8] A. G. Greenberg *et al.*, "A clean slate 4d approach to network control and management," *ACM CCR*, vol. 35, no. 5, pp. 41–54, 2005.

[9] N. McKeown *et al.*, "Openflow: enabling innovation in campus networks," *ACM CCR*, vol. 38, no. 2, pp. 69–74, 2008.

[10] H. Yan *et al.*, "Tesseract: A 4d network control plane," in *NSDI*, 2007.

[11] J. Liu *et al.*, "Ensuring connectivity via data plane mechanisms," in *NSDI*, 2013.

[12] J. Feigenbaum *et al.*, "Brief announcement: on the resilience of routing tables," in *PODC*. ACM, 2012.

[13] M. Chiesa *et al.*, "A survey of fast recovery mechanisms in the data plane," *TechRxiv*, May 2020.

[14] D. Stamatelakis and W. D. Grover, "IP layer restoration and network planning based on virtual protection cycles," *IEEE J. Sel. Areas Commun.*, vol. 18, no. 10, pp. 1938–1949, 2000.

[15] A. Kabbani *et al.*, "Flowbender: Flow-level adaptive routing for improved latency and throughput in datacenter networks," in *CoNEXT*. ACM, 2014.

[16] J. Papán *et al.*, "Overview of ip fast reroute solutions," in *ICETA*, 2018.

[17] A. Jarry, "Fast reroute paths algorithms," *Telecommunication Systems*, vol. 52, no. 2, pp. 881–888, 2013.

[18] A. Kamisiński, "Evolution of IP fast-reroute strategies," in *RNDM*. IEEE, 2018.

[19] P. Pan *et al.*, "Fast reroute extensions to RSVP-TE for LSP tunnels," *RFC*, vol. 4090, pp. 1–38, 2005.

[20] Switch Specification 1.3.1, "OpenFlow," in *https://bit.ly/2VjOO77*, 2013.

[21] Cisco, "Configuring BGP PIC Edge and Core for IP and MPLS," Oct. 2017.

[22] D. Xu *et al.*, "Failure protection in layered networks with shared risk link groups," *IEEE Network*, vol. 18, no. 3, pp. 36–41, 2004.

[23] P. Sebos *et al.*, "Auto-discovery of shared risk link groups," in *OFC*, vol. 3, 2001, pp. WDD3–WDD3.

[24] M. Menth *et al.*, "Resilience analysis of packet-switched communication networks," *IEEE/ACM Trans. Netw.*, vol. 17, no. 6, pp. 1950–1963, 2009.

[25] A. Atlas and A. Zinin, "Basic specification for IP fast reroute: Loop-free alternates," *RFC*, vol. 5286, pp. 1–31, 2008.

[26] T. Elhourani *et al.*, "IP fast rerouting for multi-link failures," in *INFOCOM*. IEEE, 2014.

[27] M. Chiesa *et al.*, "On the resiliency of static forwarding tables," *IEEE/ACM Trans. Netw.*, vol. 25, no. 2, pp. 1133–1146, 2017.

[28] B. Yang *et al.*, "Keep forwarding: Towards k-link failure resilient routing," in *INFOCOM*. IEEE, 2014.

[29] A. Bhalgat *et al.*, "Fast edge splitting and edmonds' arborescence construction for unweighted graphs," in *SODA*, 2008.

[30] M. Chiesa *et al.*, "The quest for resilient (static) forwarding tables," in *INFOCOM*. IEEE, 2016.

[31] K.-T. Foerster *et al.*, "CASA: congestion and stretch aware static fast rerouting," in *INFOCOM*. IEEE, 2019.

[32] ——, "Bonsai: Efficient fast failover routing using small arborescences," in *DSN*. IEEE, 2019.

[33] J. Liu *et al.*, "Data-driven network connectivity," in *HotNets*. ACM, 2011.

[34] S. Ray *et al.*, "Always acyclic distributed path computation," *IEEE/ACM Trans. Netw.*, vol. 18, no. 1, pp. 307–319, 2010.

[35] K.-T. Foerster *et al.*, "Improved fast rerouting using postprocessing," in *SRDS*. IEEE, 2019.

[36] ——, "Local fast failover routing with low stretch," *ACM CCR*, vol. 48, no. 1, pp. 35–41, 2018.

[37] D. Watts and S. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, pp. 440–442, June 1998.

[38] A. Frank, "Augmenting graphs to meet edge-connectivity requirements," *SIAM J. Discret. Math.*, vol. 5, no. 1, pp. 25–53, 1992.

[39] D. Zhuo *et al.*, "Understanding and mitigating packet corruption in data center networks," in *SIGCOMM*. ACM, 2017.

[40] H. Cetinay *et al.*, *Comparing Destructive Strategies for Attacking Networks*. Cham: Springer International Publishing, 2020, pp. 117–140.

[41] A. Markopoulou *et al.*, "Characterization of failures in an operational IP backbone network," *IEEE/ACM Trans. Netw.*, vol. 16, no. 4, pp. 749–762, 2008.

[42] D. Turner *et al.*, "California fault lines: understanding the causes and impact of network failures," in *SIGCOMM*. ACM, 2010.

[43] R. Govindan *et al.*, "Evolve or die: High-availability design principles drawn from googles network infrastructure," in *SIGCOMM*, 2016.

[44] R. Potharaju and N. Jain, "When the network crumbles: an empirical study of cloud network failures and their impact on services," in *SoCC*. ACM, 2013, pp. 15:1–15:17.

[45] C. Jiang *et al.*, "PCF: provably resilient flexible routing," in *SIGCOMM*. ACM, 2020.

[46] H. H. Liu *et al.*, "Traffic engineering with forward fault correction," in *SIGCOMM*. ACM, 2014.

[47] C. Busch *et al.*, "Analysis of link reversal routing algorithms for mobile ad hoc networks," in *SPAA*. ACM, 2003.

[48] E. Gafni and D. P. Bertsekas, "Distributed algorithms for generating loop-free routes in networks with frequently changing topology," *IEEE Trans. Communications*, vol. 29, no. 1, pp. 11–18, 1981.

[49] M. S. Corson and A. Ephremides, "A distributed routing algorithm for mobile wireless networks," *Wirel. Netw.*, vol. 1, no. 1, pp. 61–81, 1995.

[50] M. Markovitch and S. Schmid, "SHEAR: A highly available and flexible network architecture marrying distributed and logically centralized control planes," in *ICNP*. IEEE, 2015.

[51] A. Vahdat *et al.*, "A purpose-built global network: Google's move to SDN," *Commun. ACM*, vol. 59, no. 3, pp. 46–54, 2016.

[52] J. Liu *et al.*, "Ensuring connectivity via data plane mechanisms," in *NSDI*, 2013.

[53] A. Atlas and A. Zinin, "Basic specification for IP fast reroute: Loop-free alternates," *RFC*, vol. 5286, pp. 1–31, 2008.

[54] G. Rétvári *et al.*, "IP fast reroute: Loop free alternates revisited," in *INFOCOM*. IEEE, 2011.

[55] A. Bashandy *et al.*, "Topology independent fast reroute using segment routing," *Working Draft, Internet-Draft draft-bashandy-rtgwg-segment-routing-tilfa-05*, 2018.

[56] K.-T. Foerster *et al.*, "TI-MFA: keep calm and reroute segments fast," in *Global Internet Symposium*. IEEE, 2018.

[57] ——, "Local fast segment rerouting on hypercubes," in *OPODIS*, 2018.

[58] J. S. Jensen *et al.*, "P-rex: fast verification of MPLS networks with multiple link failures," in *CoNEXT*. ACM, 2018.

[59] C. Filsfils *et al.*, "Bgp prefix independent convergence (pic)," *Cisco, San Jose, CA, Tech. Rep*, 2011.

[60] M. Chiesa *et al.*, "PURR: a primitive for reconfigurable fast reroute: hope for the best and program for the worst," in *CoNEXT*. ACM, 2019.

[61] M. Canini *et al.*, "A distributed and robust SDN control plane for transactional network updates," in *INFOCOM*. IEEE, 2015.

[62] M. Borokhovich *et al.*, "Provable data plane connectivity with local fast failover: introducing openflow graph algorithms," in *HotSDN*. ACM, 2014.

[63] O. Reingold, "Undirected connectivity in log-space," *J. ACM*, vol. 55, no. 4, pp. 17:1–17:24, 2008.

[64] D. Dereniowski *et al.*, "Bounds on the cover time of parallel rotor walks," in *STACS*, 2014.

[65] M. Chiesa *et al.*, "Exploring the limits of static failover routing (v4)," *CoRR*, vol. abs/1409.0034.v4, 2016.

[66] ——, "On the resiliency of randomized routing against multiple edge failures," in *ICALP*, 2016.

[67] G. Bankhamer *et al.*, "Local fast rerouting with low congestion: A randomized approach," in *ICNP*. IEEE, 2019.

[68] M. Borokhovich and S. Schmid, "How (not) to shoot in your foot with SDN local fast failover - A load-connectivity tradeoff," in *OPODIS*, 2013.

[69] K.-T. Foerster *et al.*, "On the feasibility of perfect resilience with local fast failover," in *Symposium on Algorithmic Principles of Computer Systems (APOCS)*, 2021.

[70] J. Tapolcai, "Sufficient conditions for protection routing in ip networks," *Optimization Letters*, vol. 7, no. 4, pp. 723–730, 2013.