Improving the Resilience of Fast Failover Routing: TREE (Tree Routing to Extend Edge disjoint paths)

Oliver Schweiger
University of Vienna
Faculty of Computer Science
Austria
oschweiger@cs.univie.ac.at

Klaus-Tycho Foerster
TU Dortmund
Germany
klaus-tycho.foerster@tu-dortmund.de

Stefan Schmid
TU Berlin, University of Vienna, Fraunhofer SIT
Germany and Austria
stefan.schmid@tu-berlin.de

ABSTRACT

Today's communication networks have stringent availability requirements and hence need to rapidly restore connectivity after failures. Modern networks thus implement various forms of fast reroute mechanisms in the data plane, to bridge the gap to slow global control plane convergence. State-of-the-art fast reroute commonly relies on disjoint route structures, to offer multiple independent paths to the destination.

We propose to leverage the network's path diversity to extend edge disjoint path mechanisms to tree routing, in order to improve the performance of fast rerouting. We present two such tree-mechanisms in detail and show that they boost resilience by up to 12% and 25% respectively on real-world, synthetic, and data center topologies. Whereas the first method retains the stretch of edge disjoint path mechanisms, our second method increases it depending on the use case, just below 8% for networks from Topology Zoo on average, but by up to 56% for random graphs in the Erdős-Rényi model.

CCS CONCEPTS

• **Computer systems organization → Reliability**.

KEYWORDS

routing schemes, fast reroute, network resilience, data plane

ACM Reference Format:
Oliver Schweiger, Klaus-Tycho Foerster, and Stefan Schmid. 2021. Improving the Resilience of Fast Failover Routing: TREE (Tree Routing to Extend Edge disjoint paths). In *Symposium on Architectures for Networking and Communications Systems (ANCS '21), December 13–16, 2021, Layfette, IN, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3493425.3502747

ANCS '21, December 13–16, 2021, Layfette, IN, USA

ACM ISBN 978-1-4503-9168-9/21/12.
https://doi.org/10.1145/3493425.3502747

1 INTRODUCTION

Communication networks form a critical backbone of the digital society. To meet their high availability and dependability requirements, these networks need to be able to deal with failures: especially link failures are unavoidable today, and are likely to become more frequent at increasing scale.

Fast reroute [8] is an attractive solution used in most modern communication networks today to quickly reroute traffic around failed links (henceforth also called fast failover). The fast reroute mechanism is implemented in the dataplane, relying only on local information for fast decision making, and hence avoiding the overheads and delays usually involved in control plane mechanisms (such as path reconvergence, link reversal or the notification of a centralized controller) [8].

Designing fast reroute mechanisms however is challenging, due to the limited information such local solutions can have when deciding to which port to forward a packet. Indeed, even preserving connectivity is challenging [9, 16, 17], and accounting for additional properties such as stretch or congestion only renders the problem more difficult [3, 23].

A powerful and widely-used approach to realize fast reroute mechanisms is to rely on (directed) edge-disjoint spanning trees and paths [6, 9, 10, 14, 18–20, 22, 23]: upon encountering a failure, the packet can be locally steered to the next tree or path, upon which it then reaches the destination; respectively switches again after hitting the next failure. Whereas trees use less routing table space, they are inherently limited by the global network connectivity. Edge-disjoint paths however directly provide routes between source and destination, providing better path lengths and resilience to failures.

This paper is motivated by the recent success of such disjoint path approaches, and especially CASA [23]. In particular, we envision that the resilience and path length quality of such edge-disjoint paths can be improved upon by attaching further subpaths, which can then act as local detours to better leverage the network's path diversity. We hence leverage Tree Routing to Extend Edge disjoint paths, denoted as TREE, and positively evaluate our new mechanism on real-world and synthetic topologies. To this end, we make the following contributions in this paper. In this work, we:

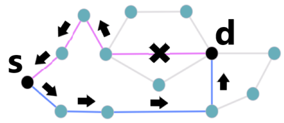

Figure 1: Motivational failover scenario. Left: Edge Disjoint Paths that require lengthy rerouting. Right: Edge Disjoint Paths extended to trees that offer shorter failover routes and improved resilience.

- build upon and expand the idea of edge disjoint path structures for fast failover routing.
- present novel TREE algorithms that utilize previously unused link diversity in the network to create tree-structures to improve the overall network resilience, while still preserving the routing mechanism's locality.
- provide multiple implementations of this idea and analyze them with respect to resilience and path length and study their respective trade-offs.
- conduct a comprehensive evaluation of these algorithms on synthetic and real data-sets. Our results show a resilience boost by up to 12.7% and 25.5% respectively, with acceptable pre-computation overhead. At the same time, our first algorithm keeps the stretch properties of edge disjoint paths, but our second method increases it depending on the use case, just below 8% for networks from Topology Zoo on average, but also by up to 56% for random Erdős-Rényi graphs.
- provide our code at https://github.com/oliver306/TREE.

2 MOTIVATION

Prior work has shown that Edge Disjoint Paths (abbreviated as `EDPs` in the following) offer both good resilience and path lengths for fast failover routing [23]. These `EDPs` are pre-computed for every source and destination pair and are then used to route efficiently between them. If a link on any path fails, packets bounce back to the source and another path is used for routing, until the destination is reached. However, merely using these `EDPs` leaves a lot of resilience and path length potential on the table, as the amount of `EDPs` that can be built is limited by the minimum cut between source and destination. For example in the network in Fig. 1, only two `EDPs` can be constructed between s and d, leaving most of the links around d unused.

We propose to enhance the `EDPs` by extending them into trees. By having the tree's leaves adjacent to d, routing along those trees in a *depth first search* (DFS) manner will offer new paths to d that are otherwise left unused by the `EDPs`.

Ideally, these trees then offer fast local failover reroutes that do not require a packet to be routed back to the source, but rather jump into another branch of the tree that leads to d on a short route. Fig. 1 illustrates this scenario: when using just `EDPs` on the left, the link failure adjacent to d leads to a lengthy rerouting process. On the right side, when using trees, the packet can be rerouted locally and reach the destination d quickly. This can also improve network resilience: if the vertical link in Fig. 1 between d and the node beneath it fails as well, the `EDPs` are disconnected from the destination d, whereas the trees would offer multiple failover routes to bridge these gaps. We note that until convergence of routing rules, packets in this context might not be routed along shortest paths, and we discuss in parallel work [34] how to *shortcut* such failover routes.

3 MODEL

The network is represented as a connected graph $G = (V, E)$, with $|V| = n$ nodes connected by $|E|$ links. These links are full-duplex symmetric, meaning they are always bidirectional. We call such a graph k-connected, if it remains connected after the removal of any $k - 1$ links.

We will investigate fast failover routing between source nodes s and destination nodes d, with $s \neq d$. To this end, we assume that the static failover rules are computed ahead of time and deployed at the nodes without a priori knowledge of the link failures. More precisely, the routing rules at a node $v \in V$ may only match on the source s, the destination d, the incident failures at v, and the incoming port. Hence, the failover rules are purely local and come into effect immediately. We do not allow for randomization, packet header modification, dynamic routing table changes, or re-convergence by means of control or data plane communication.

Regarding performance metrics, we consider two major aspects to determine overall routing quality. First, the overall *hop-count* it takes for a packet to reach d from s.

Second, the *resilience* denotes overall routing success rate in terms of whether, in the case of link failures, d could be reached at all via a given routing scheme. Ideally, a routing scheme provides high resilience with low hop count.

4 ALGORITHM

This section describes our two methods to extend `EDPs` to trees, in such a way that the inherent good resilience of `EDPs` [23] is maintained. Two major variants arise, denoted as `One Tree` (§4.1) and `Multiple Trees` (§4.2). `One Tree` retains the hop count quality of `EDPs`, whereas `Multiple Trees` trade in extra hop count for additional resilience.

4.1 `One Tree`

The idea of `One Tree` revolves around creating a single expansive tree out of *one* of the `EDPs`, which acts a fail-safe in case all prior paths visited on the `EDPs` suffer from link failures. In this version, the longest path in the `EDPs` is being extended as outlined in Algorithm 1. After this, we end up with a tree spanning potentially the whole network.

Algorithm 1 Extend - One Tree

```
pathToExtend ← longest path in EDPs as node array
for i = 1 → length(pathToExtend) − 1 do        //WALK DOWN EDP
    nodes ← pathToExtend    //INITIALIZE NEW NODES AS NODES OF EDP
    it ← 0
    while it < length(nodes) do
        neighbors ← neighboring nodes of nodes[i]
        for j = 0 → length(neighbors) − 1 do
            if neighbors[j] not part of tree already then
                nodes.insert(neighbors[j])
                add neighbors[j] to tree
            end if
        end for
        it + +
    end while
end for
```

Since not every branch of this tree will lead to the destination d, this might lead to unnecessary detours when routing. Thus, we afterwards remove such unwanted branches from the tree in linear time, omitted here for brevity.

The final result is a tree that contains no redundant paths with each leaf neighboring the destination. In the case of link failures, the tree's branches can act as a bridge to d where traditional EDPs would fail, and hence is expected to improve the overall network resilience, since it provides additional routes to d. The hop count quality of EDPs can be maintained in this structure, as we will discuss in §4.3.

4.2 Multiple Trees

While One Tree attempts to construct only a single tree that acts as a fail-safe attached to one the the EDPs, there might still remain links left unutilized in the network. This is because a single tree has limitations to its growth, as shown in Algorithm 1: newly created branches cannot cross over existing ones, as this would violate the tree's property of being acyclic. However, this could be alleviated by employing multiple trees that all act as separate structures. This idea lends itself especially well to our setup, as there are often multiple EDPs to work with. Thus, we propose the construction of multiple trees on top of the EDPs as outlined in Algorithm 2.

When examining the differences between Algorithm 1 and 2, the former one builds its routing structure on a *node*-basis, whereas the latter one uses edge markings to form its trees in the network. This differentiation is needed, as in Algorithm 1 nodes can only belong to one single structure, whereas Algorithm 2 only requires the *edges* of its trees to be disjoint. Note also how the destination d is not a part of any of these trees, as the tree leaves are placed at the nodes that

Algorithm 2 Extend - Multiple Trees

```
EDPs ← get all EDPs between s and d, sorted desc. by their length
for i = 0 → length(EDPs) − 1 do
    pathToExtend ← EDPs[i]
    for j = 1 → length(pathToExtend) − 1 do  //WALK DOWN EDP
        nodes ← pathToExtend        //INITIALIZE NEW NODES AS NODES IN CURRENT EDP
        it ← 0
        while it < length(nodes) do
            neighbors ← neighboring nodes of nodes[j]
            for k = 0 → length(neighbors) − 1 do
                if edge to neighbors[k] not part of any tree already then
                    nodes.insert(neighbors[k])
                    add edge to neighbors[k] to tree
                end if
            end for
            it + +
        end while
    end for
end for
```

are *neighbors* to d. As for Algorithm 1, one needs to perform afterwards the tree truncation to every constructed tree to remove redundant branches, again omitted here for brevity.

4.3 Routing

Now that we have established how both structures are computed, the question remains how to efficiently route along them. Summing up, the following steps are to be carried out:

(1) compute the set of EDPs from s to d
(2) sort the set of EDPs in descending order[1]
(3) apply either Algorithm 1 or 2 to the EDPs
(4) truncate the trees to remove unwanted branches
(5) sort the original set of EDPs in ascending order
(6) for each of the EDP trees one after another while d is not reached, route along the tree in a *depth-first* search (henceforth called *Depth First Routing* (*DFR*)). By the previous sort, we route to d by the shortest EDP first.

Notice how depth-first traversal of the EDPs also makes sense for the EDPs left untouched, as the normal routing-behaviour of going down and up (in case of failure) a single path is equivalent to a depth-first traversal. There is one additional trick that we apply to DFR. While traversing the tree depth-first, at nodes with a degree greater than 2, we

[1]By doing this, we start applying the tree formation to the longest EDP first. This is useful, as the shortest EDPs tend to remain in their original form and thus offer the fastest route to d in case they do not suffer from any failure.

make use of an internal per-node ranking that determines the order at which the individual branches of the tree are visited. This rank indicates which branch has the shortest distance to any leaf below it. This has the effect that, when routing, the shortest potential path to d within the tree is always picked, i.e., a shortest branch depth-first order.

Moreover, we extend DFR for `One Tree` to retain the hop count quality of `EDPs`, under the assumption that the `EDPs` can successfully reach the destination d. The single tree of `One Tree` still contains the original `EDP` as one of its branches and we adapt the routing to first try this path, before using additional tree branches, i.e., `One Tree` first uses the `EDPs`. In this way, the hop count of `EDPs` and `One Tree` is identical, as long as the `EDPs` still connect source and destination.

Ranked DFR can be achieved by static routing tables. As the `EDPs` and tree-extensions are pre-computed, these rules need only be configured at the individual nodes, where for packets from s to d, the outgoing port can be uniquely identified by the incoming port and incident failures.

5 EVALUATION

This section evaluates how both tree algorithms perform in comparison to `EDPs` [23]. For this, we carried out a multitude of experiments, both on real and synthetic data sets and investigate two failure models that help with examining the strong and weak points of the respective techniques. We consider two performance measures to evaluate performance: the **hop-count** h_{rs} of an employed routing scheme rs, and its **resilience** r_{rs}. Both metrics have been introduced in §3.

5.1 Link Failure Models

Our setup revolves around applying failures to the network, i.e., the removal of links. We use two failure models:

- **Random**: edges in the network fail at random.
- **Clustered** simulates failures around d with a chance of p^f. Additionally, failures propagate to adjacent edges, with a probability being reduced per hop by p^f_Δ. For our experiments, we set $p^f_\Delta = 0.3$. This model aims to simulate the failure behavior of zones affected by some kind of natural disaster [32, 38], such as earth quakes or other sources of extensive power outages, where network links might fail in an area of larger scale around an, e.g., epicenter [1, 30].

Note: the meaning of *failure rate* differs between clustered and random failures. Clustered failures describe a certain percentage of failing links around d (which then propagate further). For random failures, we treat the failure rate proportional to the general network connectivity (described by k), as to better generalize this metric for both dense and sparse graphs. As such, a failure rate p^f for k-connected graphs means that $p^f * k$ links fail, chosen uniformly at random.

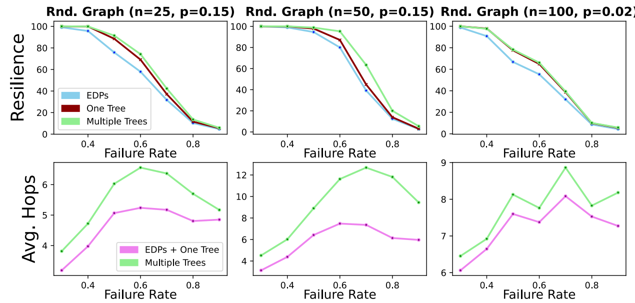

Figure 2: Random graphs with clustered failures.

5.2 Evaluation Setup and Metrics

We examine both real-world topologies by routing on graphs from the Internet Topology Zoo [27], as well as artificial random Erdős-Rényi graphs [15, 24] and data center topologies [35]. The construction of these Erdős-Rényi graphs is defined by two essential parameters n denoting the amount of nodes in the graph, and p denoting the probability of an edge appearing between two nodes.

For generating meaningful measurements of h_{rs} and r_{rs}, we simulated 1000 routing runs per data point in our plots in the next Sections 5.3 and 5.4. Each run, for random graphs, a new Erdős-Rényi graph is generated with the same parameters n and p, and also s and d are newly chosen at random for every run. For the real-world topologies, we keep the same graph over all runs, but also pick s and d randomly.Lastly, for the data center topologies, we performed 200 runs per failure rate per topology size, each time picking s and d randomly.

We compare our `One Tree` and `Multiple Trees` with the state-of-the-art fast failover routing scheme for `EDPs` [23], which we implement in Python 3.8.6 using NetworkX 2.5 [25].

For the three schemes, we compute and compare the average resilience and hop count for given failure rates in §5.3 and §5.4, as well as the average pre-computation time in §5.6. For comparing schemes in terms of their hop count, we only take into account the runs where the `EDPs` succeeded.[2]

5.3 Routing in Random Graphs

5.3.1 Random Failures. `Multiple Trees` outperforms `One Tree` and `EDPs` in terms of resilience, offering resilience gains of up to 21.5 percent, while `One Tree` also outperforms the `EDPs`, though less drastically. The resilience gains of `One Tree` of up to 5 percent is still noteworthy, since it offers the exact same average hop count as the `EDPs` (for this reason, their blue and red lines from the top row are joined into a single violet one for the bottom row in Figures 2 and 3). On overall average, it takes `Multiple Trees` 1.51 hops longer, compared to `EDPs` or `One Tree`. The largest average increase (in both §5.3.1 and §5.3.2) is at $n = 100$ nodes at an extra 56%.

[2]By construction, when `EDPs` reach the destination, both `One Tree` and `Multiple Trees` do so as well, as they extend the `EDPs`.

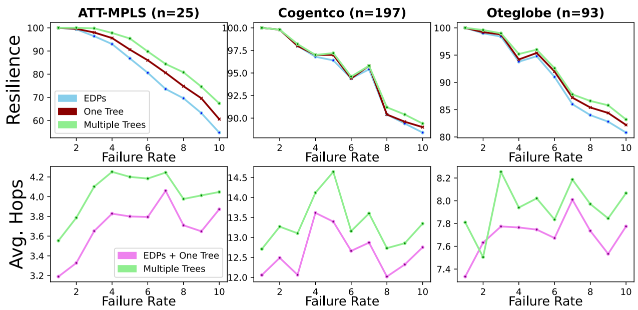

Figure 3: Real-world graphs with random failures.

5.3.2 Clustered Failures. Similar results (Fig. 2) arise when comparing routing performance on the random graphs generated with the same parameters n and p as before, but now with clustered failures. `One Tree` outperforms the `EDPs` in terms of resilience by 12.7 percent, while offering the same average hop count. `Multiple Trees` manages to improve resilience by up to 24 percent, while still offering a very comparable hop count (an average increase of 1.6 hops), especially for larger, less dense graphs (i.e., $n = 100$, $p = 0.02$).

5.4 Routing in Real-World Topologies

For testing our algorithms on real-world topologies, we opted to use graphs from the Internet Topology Zoo [27], as common in related work. Overall, we chose three different topologies with varying n, in order to evaluate performance on both larger and smaller real-world networks.

As with the artificial graphs from §5.3, we apply both random and clustered failures to the graph, while pushing up the failure rate, and compare resilience and average hop count of the three schemes. On average, over all experiments in §5.4, `Multiple Trees` requires less than 8% extra hops.

5.4.1 Random Failures. Fig. 3 shows that, similarly to the artificial networks, both `One Tree` (resilience boost up to 7 percent) and `Multiple Trees` (resilience boost up to 12.6 percent) outperform the `EDPs` on real-world graph for random failures, `One Tree` often times offering resilience values close to those of `Multiple Trees`. `One Tree` and `EDPs` take the same amount of hops to reach d, with `Multiple Trees` exhibiting an average increase in hop count by 0.44 hops.

5.4.2 Clustered Failures. Next, we employ clustered failures to the real-world topologies. Yet again, a similar pattern emerges. `One Tree` and `Multiple Trees` boost resilience by up to 11.9 and 18.7 percent respectively, with `One Tree` being just as fast as the `EDPs`. `Multiple Trees` also shows to be efficient here, as it offers high resilience improvements while only increasing the hop count by 0.52 on average.

5.5 Routing in Data Center Topologies

Recent research [12, 13, 26, 35, 39, 40] has shown significant performance benefits for flat topology designs in data center networks and we as hence also investigate our tree algorithms in these settings. For easy scalability of our evaluations, we choose topologies as in Jellyfish [35], modeled as random δ-regular graphs. Multiple test runs with $\delta = 8$ and $n \in \{25, 50, 100\}$ were performed, again both with clustered and random failures. For `Multiple Trees`, in these experiments resilience could be improved by up to 25.5% at an average hop increase of 4.5 for $n = 50$ for clustered failures, and a resilience-increase of up to 18.5% at an average hop increase of 1.96 for $n = 25$ for random failures. However, `One Tree` performed slightly worse on these graphs, as performance was only marginally better than its `EDP` counterpart most of the time. Still, from a percentage point of view, `Multiple Trees` increased the stretch in average by up to 74% for random and by up to 82% for clustered failures.

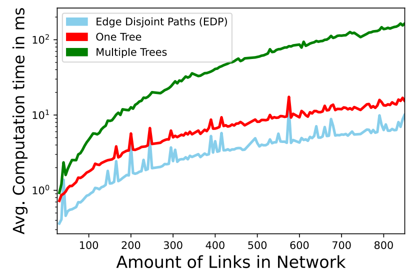

Figure 4: The relationship between computation time in milliseconds and the amount links in the network for the three compared algorithms

5.6 Computational Runtime Comparison

We next analyze the runtime that it takes for the `EDPs`, `One Tree`, and `Multiple Trees` structures to be computed. Our single-threaded computations were performed on a Ryzen 9 3900X CPU @3.8GHz with 32GB of DDR4 RAM, using Python 3.8.6 with NetworkX 2.5 [25]. We use Erdős-Rényi graphs with n rising from 25 to 105 in steps of 10, a constant $p = 0.15$ and taking 200 runs for each n. Fig. 4 plots the runtime of the three algorithms w.r.t. to the number of links in the network, taking the average runtime for ranges of size 5, i.e., 1 to 5 links, 6 to 10 links, etc. We observe a much heavier workload for the creation and truncation of `Multiple Trees`, where the computation times for `Multiple Trees` scales linearly with the number of links multiplied with the number of trees built in the network, while `One Tree` computes much faster, even in larger networks, as there is only a single tree to build and, consequently, truncate: there is not much overhead when comparing `One Tree` to `EDPs`.

5.7 Discussion

Overall, these results showcase the unused potential in the random and real-world graphs that our techniques leverage. By building trees from the `EDPs`, we manage to establish local failover mechanism that increase resilience. One Tree retains the path length of EDPs, while still providing noticeable resilience gains. One Tree keeps the efficiency of the EDPs in terms of hop count, while still leaving some potential links unused. Consequently, Multiple Trees does use these unused links to further increase resilience, with the downside of exhibiting larger hop-counts: while the relative increase is not large for graphs from Topology Zoo, it can be significant for very dense graphs. Thus, we perceive the use of either One Tree or Multiple Trees to be highly dependable of the exact use-case and underlying network properties. If the speed of packet arrival is not a top priority, but a higher resilience is desired, one might use Multiple Trees. Otherwise, if one might want to keep hop-counts short but still leverage some unused link-potential and computation time for the routing structures is critical, we propose to construct One Tree on top of the existing EDPs. Notwithstanding, it would be interesting to further investigate the resilience-stretch trade-off in the design space between One Tree and Multiple Trees. Lastly, however a downside of our new algorithms is the additional routing table space consumption, and it would also be interesting to adapt our algorithms to find a good trade-off respectively to use compression [36, 37].

6 RELATED WORK

Failures are common in computer networks [32] and hence fast reroute mechanisms have been studied comprehensively over the last decades, we refer to a recent survey for an overview [8]. In general, for rapid connectivity restoration, the computation & distribution of new routes is too slow [29], but even convergence mechanisms in the data plane such as DDC [29] can introduce non-trivial delays [5].

As such many lines of research have considered how to overcome the delay induced by convergence mechanisms by means of pre-computation; our work falls into this regime.

Some of these fast failover algorithms leverage packet header modification, e.g., to carry failure information [21, 28], by employing MPLS [31] or to leverage graph exploration approaches [4], but they can be disruptive to other network functions and moreover require specialized equipment or in-node computation abilities, similar arguments can be made for mechanisms that require dynamic state on the forwarding devices, such as, e.g., resilience via a rotor router [11]. Recent work also investigated the use of randomization [2, 7], but they require some means of random number generation on the routers and can lead to packet reordering.

In contrast, many deterministic static fast failover algorithms, as in our work, do not suffer from the above issues, however at the price of reduced resilience. Conceptually, in this framework, achieving perfect resilience is impossible [16, 17], i.e., to guarantee reaching the destination under the assumption of connectivity. State-of-the-art deterministic fast failover algorithms hence leverage underlying pre-failure connectivity assumptions to pre-compute disjoint routing structures, between which the packets can be locally migrated after hitting a failure. For destination-based routing, the standard approach is by arc-disjoint rooted spanning trees (arborescences) [9, 10, 14], yielding worst-case resilience guarantees related to the number of such trees that can be constructed. Recent work expanded upon their construction in heterogeneous networks in a heuristic fashion, by, e.g., adding local non-spanning arborescences [20], or by employing partial structural networks [41]. Nonetheless, due to the paradigm of destination-based routing, their path length falls behind source-destination-based approaches [23].

When routing based on both source and destination, CASA showed that the use of edge disjoint paths (EDPs) can improve worst-case resilience guarantees and improve the hop count, at the expense of additional routing table space. Our work is most related to CASA and builds upon it, by extending their approach of EDPs with one or multiple trees. Our One Tree approach retains the hop count quality of CASA, but adds additional resilience, whereas our Multiple Trees approach shows longer path lengths in our evaluation, but improves even upon the resilience of One Tree.

7 CONCLUSION

We introduced a method that builds upon established static and deterministic local fast failover routing rules and further modifies and enhances them. Two algorithms were introduced: One Tree manages to keep the hop count low as in traditional approaches using Edge Disjoints Paths, but boosts overall network resilience. Multiple Trees further enhances the network resilience by extending all EDPs to trees, at the cost of slightly larger hop counts. We analyzed these trade-offs in evaluations on real-world, random, and data center topologies and saw good performance benefits over previous work, with, e.g., Multiple Trees providing up to over 25 percentage points of additional routing success.

Reproducibility. In order to foster reproducibility and to enable comparisons with future work, our source code will be made available at https://github.com/oliver306/TREE. An extended technical report for this paper can be found at [33].

Acknowledgments. We thank our shepherd Brent Stephens and the reviewers for their valuable feedback. We also thank Paula-Elena Gheorghe for an earlier implementation of a tree algorithm. This research received funding from the Vienna Science and Technology Fund (Wiener Wissenschafts-, Forschungs- und Technologiefonds WWTF) project, Fast and Quantitative What-if Analysis for Dependable Communication Networks (WHATIF), ICT19-045, 2020-2024.

REFERENCES

[1] Pankaj K. Agarwal, Alon Efrat, Shashidhara K. Ganjugunte, David Hay, Swaminathan Sankararaman, and Gil Zussman. 2013. The Resilience of WDM Networks to Probabilistic Geographical Failures. *IEEE/ACM Trans. Netw.* 21, 5 (2013), 1525–1538.

[2] Gregor Bankhamer, Robert Elsässer, and Stefan Schmid. 2019. Local Fast Rerouting with Low Congestion: A Randomized Approach. In *ICNP*. IEEE, 1–11.

[3] Gregor Bankhamer, Robert Elsässer, and Stefan Schmid. 2021. Randomized Local Fast Rerouting for Datacenter Networks with Almost Optimal Congestion. In *DISC (LIPIcs)*.

[4] Michael Borokhovich, Liron Schiff, and Stefan Schmid. 2014. Provable data plane connectivity with local fast failover: introducing openflow graph algorithms. In *HotSDN*. ACM, 121–126.

[5] Costas Busch, Srikanth Surapaneni, and Srikanta Tirthapura. 2003. Analysis of link reversal routing algorithms for mobile ad hoc networks. In *SPAA*. ACM, 210–219.

[6] Marco Chiesa, Andrei V. Gurtov, Aleksander Madry, Slobodan Mitrovic, Ilya Nikolaevskiy, Michael Schapira, and Scott Shenker. 2016. On the Resiliency of Randomized Routing Against Multiple Edge Failures. In *ICALP (LIPIcs, Vol. 55)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 134:1–134:15.

[7] Marco Chiesa, Andrei V. Gurtov, Aleksander Madry, Slobodan Mitrovic, Ilya Nikolaevskiy, Michael Schapira, and Scott Shenker. 2016. On the Resiliency of Randomized Routing Against Multiple Edge Failures. In *ICALP*. 134:1–134:15.

[8] Marco Chiesa, Andrzej Kamisinski, Jacek Rak, Gábor Rétvári, and Stefan Schmid. 2021. A Survey of Fast-Recovery Mechanisms in Packet-Switched Networks. *IEEE Commun. Surv. Tutorials* 23, 2 (2021).

[9] Marco Chiesa, Ilya Nikolaevskiy, Slobodan Mitrovic, Andrei V. Gurtov, Aleksander Madry, Michael Schapira, and Scott Shenker. 2017. On the Resiliency of Static Forwarding Tables. *IEEE/ACM Trans. Netw.* 25, 2 (2017), 1133–1146.

[10] Marco Chiesa, Ilya Nikolaevskiy, Aurojit Panda, Andrei V. Gurtov, Michael Schapira, and Scott Shenker. 2016. Exploring the Limits of Static Failover Routing. *CoRR* abs/1409.0034.v4 (2016).

[11] Dariusz Dereniowski, Adrian Kosowski, Dominik Pajak, and Przemyslaw Uznanski. 2014. Bounds on the Cover Time of Parallel Rotor Walks. In *STACS*. 263–275.

[12] Michael Dinitz, Michael Schapira, and Asaf Valadarsky. 2015. Explicit Expanding Expanders. In *ESA*, Vol. 9294. Springer, 399–410.

[13] Michael Dinitz, Michael Schapira, and Asaf Valadarsky. 2017. Explicit Expanding Expanders. *Algorithmica* 78, 4 (2017), 1225–1245.

[14] Theodore Elhourani, Abishek Gopalan, and Srinivasan Ramasubramanian. 2016. IP Fast Rerouting for Multi-Link Failures. *IEEE/ACM Trans. Netw.* 24, 5 (2016), 3014–3025.

[15] P Erdös and A Rényi. 1959. On Random Graphs I. *Publicationes Mathematicae Debrecen* 6 (1959), 290–297.

[16] Joan Feigenbaum, Brighten Godfrey, Aurojit Panda, Michael Schapira, Scott Shenker, and Ankit Singla. 2012. Brief announcement: on the resilience of routing tables. In *PODC*. ACM, 237–238.

[17] Klaus-Tycho Foerster, Juho Hirvonen, Yvonne-Anne Pignolet, Stefan Schmid, and Gilles Trédan. 2021. On the Feasibility of Perfect Resilience with Local Fast Failover. In *APOCS*. SIAM, 55–69.

[18] Klaus-Tycho Foerster, Andrzej Kamisinski, Yvonne-Anne Pignolet, Stefan Schmid, and Gilles Trédan. 2019. Bonsai: Efficient Fast Failover Routing Using Small Arborescences. In *DSN*. IEEE, 276–288.

[19] Klaus-Tycho Foerster, Andrzej Kamisinski, Yvonne-Anne Pignolet, Stefan Schmid, and Gilles Trédan. 2019. Improved Fast Rerouting Using Postprocessing. In *SRDS*. IEEE, 173–182.

[20] Klaus-Tycho Foerster, Andrzej Kamisinski, Yvonne-Anne Pignolet, Stefan Schmid, and Gilles Trédan. 2021. Grafting Arborescences for Extra Resilience of Fast Rerouting Schemes. In *INFOCOM*. IEEE, 1–10.

[21] Klaus-Tycho Foerster, Mahmoud Parham, Marco Chiesa, and Stefan Schmid. 2018. TI-MFA: Keep calm and reroute segments fast. In *INFOCOM Workshops*. IEEE, 415–420.

[22] Klaus-Tycho Foerster, Yvonne-Anne Pignolet, Stefan Schmid, and Gilles Trédan. 2018. Local Fast Failover Routing With Low Stretch. *Comput. Commun. Rev.* 48, 1 (2018), 35–41.

[23] Klaus-Tycho Foerster, Yvonne-Anne Pignolet, Stefan Schmid, and Gilles Trédan. 2019. CASA: Congestion and Stretch Aware Static Fast Rerouting. In *INFOCOM*. IEEE, 469–477.

[24] Edgar N Gilbert. 1959. Random graphs. *The Annals of Mathematical Statistics* 30, 4 (1959), 1141–1144.

[25] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. 2008. Exploring Network Structure, Dynamics, and Function using NetworkX. In *Proceedings of the 7th Python in Science Conference*, Gaël Varoquaux, Travis Vaught, and Jarrod Millman (Eds.). Pasadena, CA USA, 11 – 15.

[26] Simon Kassing, Asaf Valadarsky, Gal Shahaf, Michael Schapira, and Ankit Singla. 2017. Beyond fat-trees without antennae, mirrors, and disco-balls. In *SIGCOMM*. ACM, 281–294.

[27] Simon Knight, Hung X. Nguyen, Nick Falkner, Rhys Alistair Bowden, and Matthew Roughan. 2011. The Internet Topology Zoo. *IEEE J. Sel. Areas Commun.* 29, 9 (2011), 1765–1775.

[28] Karthik Lakshminarayanan, Matthew Caesar, Murali Rangan, Tom Anderson, Scott Shenker, and Ion Stoica. 2007. Achieving convergence-free routing using failure-carrying packets. In *SIGCOMM*. ACM.

[29] Junda Liu, Aurojit Panda, Ankit Singla, Brighten Godfrey, Michael Schapira, and Scott Shenker. 2013. Ensuring Connectivity via Data Plane Mechanisms. In *NSDI*. USENIX Association, 113–126.

[30] Sebastian Neumayer, Gil Zussman, Reuven Cohen, and Eytan H. Modiano. 2011. Assessing the Vulnerability of the Fiber Infrastructure to Disasters. *IEEE/ACM Trans. Netw.* 19, 6 (2011), 1610–1623.

[31] Ping Pan, George Swallow, and Alia Atlas. 2005. Fast Reroute Extensions to RSVP-TE for LSP Tunnels. *RFC* 4090 (2005), 1–38.

[32] Jacek Rak and David Hutchison (Eds.). 2020. *Guide to Disaster-Resilient Communication Networks*. Springer.

[33] Oliver Schweiger, Klaus-Tycho Foerster, and Stefan Schmid. 2021. Improving the Resilience of Fast Failover Routing:TREE (Tree Routing to Extend Edge disjoint paths). *CoRR* abs/2111.14123 (2021). https://arxiv.org/abs/2111.14123

[34] Apoorv Shukla and Klaus-Tycho Foerster. 2021. Shortcutting Fast Failover Routes in the Data Plane. In *ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*.

[35] Ankit Singla, Chi-Yao Hong, Lucian Popa, and Philip Brighten Godfrey. 2012. Jellyfish: Networking Data Centers Randomly. In *NSDI*. USENIX Association, 225–238.

[36] Brent E. Stephens, Alan L. Cox, and Scott Rixner. 2013. Plinko: building provably resilient forwarding tables. In *HotNets*. ACM, 26:1–26:7.

[37] Brent E. Stephens, Alan L. Cox, and Scott Rixner. 2016. Scalable Multi-Failure Fast Failover via Forwarding Table Compression. In *SOSR*. ACM, 9.

[38] János Tapolcai, Lajos Rónyai, Bálazs Vass, and Laszlo Gyimothi. 2017. List of shared risk link groups representing regional failures with limited size. In *INFOCOM*. IEEE, 1–9.

[39] Asaf Valadarsky, Michael Dinitz, and Michael Schapira. 2015. Xpander: Unveiling the Secrets of High-Performance Datacenters. In *HotNets*. ACM, 16:1–16:7.

[40] Asaf Valadarsky, Gal Shahaf, Michael Dinitz, and Michael Schapira. 2016. Xpander: Towards Optimal-Performance Datacenters. In *CoNEXT*. ACM, 205–219.

[41] Baohua Yang, Junda Liu, Scott Shenker, Jun Li, and Kai Zheng. 2014. Keep Forwarding: Towards k-link failure resilient routing. In *INFOCOM*. IEEE, 1617–1625.