Improved Fast Rerouting Using Postprocessing

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

Improved Fast Rerouting Using Postprocessing

Klaus-Tycho Foerster, Andrzej Kamisiński, Yvonne-Anne Pignolet, Stefan Schmid, and Gilles Tredan

Abstract—To provide fast traffic recovery upon failures, most modern networks support static Fast Rerouting (FRR) mechanisms for mission critical services. However, configuring FRR mechanisms to tolerate *multiple* failures poses challenging algorithmic problems. While state-of-the-art solutions leveraging arc-disjoint arborescence-based network decompositions ensure that failover routes always reach their destinations eventually, even under multiple concurrent failures, these routes may be long and introduce unnecessary loads; moreover, they are tailored to worst-case failure scenarios.

This paper presents an algorithmic framework for improving a given FRR network decomposition, *using postprocessing*. In particular, our framework is based on iterative arc swapping strategies and supports a number of use cases, from strengthening the resilience (e.g., in the presence of shared risk link groups) to improving the quality of the resulting routes (e.g., reducing route lengths and induced loads). Our simulations show that postprocessing is indeed beneficial in various scenarios, and can therefore enhance today's approaches.

Index Terms—Resilience, fault-tolerance, computer networks, failover.

1 Introduction

Communication networks have become a critical infrastructure of our digital society: enterprises which outsource their IT infrastructure to the cloud, as well as many applications related to health monitoring, power grid management, or disaster response [1], depend on the uninterrupted availability of such networks. To meet their dependability requirements, most modern networks provide static Fast Rerouting (FRR) mechanisms [2], [3], [4], [5]. Since FRR mechanisms *pre-configure* conditional failover behaviors, they enable a very fast traffic recovery upon failures, which only involves the data plane but not the (typically much slower [6]) control plane.

However, while allowing to pre-configure conditional failover behavior is the key benefit of FRR, enabling the fast response to failures, it is also the key *challenge* when it comes to designing algorithms for such mechanisms: as the conditional failover behavior needs to be configured *before* the failures are known, the algorithmic problem of how to optimally configure the failover rules at the different routers, for all *possible* failures, seems inherently combinatorial. The problem is particularly challenging in scenarios where packet headers cannot be used to carry meta-information about encountered failures: such header rewriting is often undesired and introduces overhead (related to header rewriting itself, but also in terms of additional rules required at the routers to process such information).

While FRR technology has been used for many years already in modern communication networks, a major algorithmic result on how to configure FRR mechanisms is relatively recent: Chiesa et al. [7], [8] showed that by decomposing the network into k arc-disjoint spanning arborescences [9], highly resilient FRR configurations can be defined. Edmonds [10] proved that k-connected graphs always allow for k such arborescences, and they can be computed rapidly [9].

However, Chiesa et al.'s conjecture that for any k-connected graph, there exists a failover routing resilient to any $k-1$ failures, remains an open problem. What is more, while this network decomposition approach ensures connectivity, the failover routes may be far from optimal regarding latency (i.e., route length) and congestion.

The goal of this paper is to improve the network decomposition approach, in terms of resilience, performance, and flexibility. In particular, we are motivated by the observation that in practice, additional information about failure scenarios and failover objectives may be available, e.g., about shared risk link groups [11], [12], [13] or about critical flows for which it is important to be routed along short paths, even after failures. Existing optimizations of arborescence-based failover schemes are *oblivious* to such aspects.

Model. In a nutshell, we consider the problem of pre-defining (static) conditional failover rules at network's *nodes* (i.e., switches or routers), which define to which link to forward an incoming packet. These forwarding rules can only depend on the destination t, the in-port at which a packet arrives at the current node, as well as the status of the links directly incident to the node. At the same time, they should not depend on non-local failures or the packet source. In particular, we do not allow for packet tagging (i.e., header rewriting) or carrying failure information in the header.

More specifically, we consider FRR mechanisms leveraging arc-disjoint arborescence network decompositions [7], [8]: for each destination, a set of arborescences are defined which are rooted at the destination and span the entire network without two arborescences sharing an arc. As long as no failure is encountered, a packet travels along an arbitrary arborescence towards the root, being the destination. When encountering a failure, a packet is rerouted onto the next

• K.-T. Foerster and S. Schmid are with the Faculty of Computer Science at the University of Vienna, Austria.
• Andrzej Kamisiński is with the AGH University of Science and Technology, Poland.
• Yvonne-Anne Pignolet is with DFINITY, Switzerland.
• Gilles Tredan is with LAAS-CNRS, France.

Manuscript submitted December 15, 2019, revised May 11, 2020.

arborescence according to some pre-defined order. The logic of the latter is defined by the *arborescence routing* strategy.

Contribution. This paper presents an algorithmic framework for *postprocessing* state-of-the-art FRR mechanisms based on network decompositions, to improve resilience, performance, and flexibility, of fast rerouting. The framework relies on an iterative swapping of arcs, hence changing the network decompositions towards a certain objective. More specifically, such swapping operations can be used to account for specific failure scenarios (e.g., given by shared risk link groups), to improve traffic engineering properties of failover paths (such as load and stretch), or to flexibly adjust the failover routes to the specific requirements or priorities of flows (and their applications).

We show that we do not limit ourselves by focusing on arc-disjoint arborescence network decompositions by proving that arborescence-based decompositions are as good as any deterministic local failover method. Furthermore, we demonstrate the potential of our arc-swapping framework in four different use cases: two related to routing (i.e., improving stretch and load), and two related to properties of the decomposition (namely, depth and independence of paths). We report on extensive simulations using synthetic network topologies, which illustrate the benefits of our approach. Moreover, we also provide a novel Integer Linear Program (ILP) formulation, to directly create optimized arborescences, instead of postprocessing them.

Organization. The remainder of this paper is organized as follows. Section 2 provides intuition on why focusing on arborescences-based network decompositions is not a limitation. Our postprocessing framework is described in Section 3. We discuss and evaluate case studies in Section 4 and present our ILP in Section 5. After reviewing related work in Section 6 we conclude in Section 7.

2 Impossibility of Beating Arborescences

We first motivate our focus on failover algorithms based on arborescence network decompositions, showing that this approach does not only provide a high resilience but also competitive route qualities (in terms of lengths).

In general, while the static fast rerouting algorithms considered in this paper have the advantage that they do not require header rewriting nor control plane reconvergence, the resulting failover routes may have a high additive stretch. More formally, the (additive) stretch of a failover route from v to t is defined as the difference between the number of hops taken 1) along the failover route from v to t and the hops 2) along the shortest route from v to t. The additive stretch of the routing scheme is then the maximum stretch along all failover routes, i.e., from all v to t.

We will see that this is a feature inherent to *all* local fast failover algorithms, though as the later evaluation sections show, it is more of a rarely occurring worst-case scenario.

We start with some definitions for arborescence-based re-routing. Let (u, v) denote a directed arc from node u to v. A directed subgraph T is an r-rooted spanning arborescence of G if (i) $r \in V(G)$, (ii) $V(T) = V(G)$, (iii) r is the only node without outgoing arcs and (iv), for each $v \in V \setminus \{r\}$, there exists a single directed path from v to r. When it is clear from the context, we use the term "arborescence" to

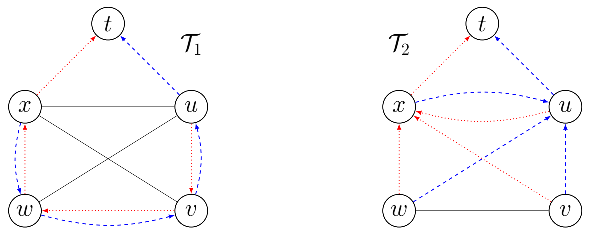


Fig. 1. Example network from [14] with two different t-rooted arc-disjoint spanning arborescence decompositions, $\mathcal{T}_1$ left and $\mathcal{T}_2$ right. In both of them one arborescence is drawn with dotted red arrows, while the second arborescence is depicted with dashed blue arrows. Note that the mean path length of the arborescences of $\mathcal{T}_1$ is 2.5, while it is less than 2 in $\mathcal{T}_2$.

refer to a t-rooted spanning arborescence, where t is the destination node. A set of arborescences $\mathcal{T} = \{T_1, \ldots T_k\}$ is arc-disjoint if no pair of arborescences in $\mathcal{T}$ shares common arcs, i.e., if $(u, v) \in E(T_i)$ then $(u, v) \notin E(T_j)$ for all $i \neq j$. A set of t-rooted arc-disjoint spanning arborescences is a valid arborescence-based decomposition. See Fig. 1 for two examples of such arborescence decompositions. In arborescence-based routing, packets follow an arborescence towards its root. In case of encountering failures on its path to the root, the packet switches to another arborescence. Let the blue dashed arborescence be failover route for x if its direct link to t fails. In this case the additive stretch is 3 for the decomposition $\mathcal{T}_1$, while it is 1 for $\mathcal{T}_2$. This illustrates that the choice of the decomposition has an impact on the quality of service in case of failures.

In the following, we show that the arborescence-based routing scheme depicted in Fig. 2 may lead to a detour of length $\Omega(n)$, even though a constant-length detour is available. In our example, the arborescences (depicted with different colours and line patters in Fig. 2) to be used have been constructed such that a certain set of failures leads to a long detour for packets emitted by node 22, even though 22 is very close to the destination t. The only link out of node 22 belongs to an arborescence that takes this long detour if no other links fail as the packet will stay on this arborescence until it reaches t.

In general though, no failover algorithm can obtain a better stretch than $\Omega(n)$ for three failures: an adversary could fail the links $(22, t), (21, 11), (23, 13)$, in which case even algorithms with global information would take a detour of length $\Omega(n)$.

However, what happens when we strengthen the definition of additive stretch to a *competitive* [15] point of view?

Recall that we so far defined the stretch in comparison to the shortest path in the network *without* failures. In a competitive setting, we compare the stretch under some failure set, to the shortest path in the network *with* failures.

To give some intuition, in the simple network setting of a cycle, a local failover strategy is to switch between clockwise and counterclockwise routing. This strategy induces an additive stretch according to the size of the cycle, for the nodes neighboring the destination, when their direct connection to the destination fails. However, for those nodes, there is no shorter path to the destination after such a failure, and hence from a competitive point of view, their failover route is optimal. We next consider short failover routes.

In the failure example of Fig. 2, an algorithm with global information could simply take a tour of length 5 from node 22

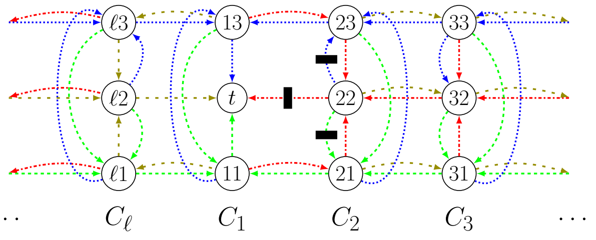


Fig. 2. Example of a $(4,\ell)$-clique-torus (see Definition 1), with 4 t-rooted arc-disjoint arborescences, in *blue* (dotted), *red* (dash-dotted), *green* (dashed), and *olive* (loosely dashed). Three links (striked out) incident to 22 have failed in this scenario, forcing a circular scheme to use the *olive* (loosely dashed) arborescence at 22, which takes a tour of length at least $\ell-1$, even though a short 5-hop alternative is available.

to t, as already pointed out above. Is it possible to find better deterministic local failover algorithms that can outperform arborescence-based routing in this example?

In this context, a deterministic algorithm makes all decisions on which out-port is used by a packet entirely depend on available information only, no randomness is used, e.g., when switching to another arborescence. An algorithm is local if its failover decisions do not take the state of other routers into account, but only the locally available information (inport, dst). In particular a router does not know where in the network other failures have happened.

Succinctly stated, the answer is *no*—all deterministic local fast failover routing schemes perform badly in such cases, i.e., they do not outperform arborescence-based routing. To this end, we will show that there are k-connected k-regular graphs where *every* deterministic local algorithm has to take large detours, even though short routes are available.

The intuition behind this statement lies in the fact that even with the freedom of taking other decisions, there are cases that lead to long detours and/or high load when only local knowhow can be used (i.e., the router does not know where else failures have happened). Thus the power of algorithms that make deterministic decisions without knowing anything about the state of other flows and routers coincides with the power of arborescence based algorithms. For our proof we define the following graph class: start with a cycle of ℓ nodes and replace each link with $k-1$ parallel links. Observe that these graphs are k-connected and k-regular, but have parallel links between neighboring nodes. In order to obtain a simple graph without parallel links, we expand each node into a clique of $k-1$ nodes, preserving connectivity and regularity. For example for $k-1=3$, this results in a $3\times\ell$ torus graph, as in Fig. 2.

Definition 1. *Let $k,\ell\in\mathbb{N}$ with $k\geq 3$, $\ell\geq 3$. A (k,ℓ)-clique-torus is a graph with $(k-1)\ell$ nodes and $(\ell(k-1))+(\ell\frac{(k-1)(k-2)}{2})$ links, constructed as follows: create ℓ cliques C_j, $1\leq j\leq\ell$, of $k-1$ nodes, i.e., so far every node has degree $k-2$. Denote the $k-1$ nodes of each clique C_j as $v_{j,1},v_{j,2},\dots,v_{j,k-1}$. Next and last, for each $1\leq j\leq\ell$ and each $1\leq i\leq k-1$, connect $v_{j,i}$ with $v_{(j \mod \ell)+1,i}$.*

We show that every deterministic local fast failover algorithm sometimes has to take detours with a length in the order of the diameter of the graph, even though a route with a constant number of hops is available. We note that our results here refer to deterministic algorithms.

Theorem 1. *For all $k\geq 3$, $\ell\geq 6$: for deterministic local fast failover algorithms ALG resilient to $k-1$ failures on k-connected k-regular graphs, matching on in-port (from which link the packet arrives) and destination, the competitive additive stretch of ALG vs. a globally optimal algorithm is $\geq\ell-6$.*

We utilize the following Lemma for the theorem proof:

Lemma 1. *Let $G=(V,E)$ be a connected graph with a link failure set $F\subset E$. Let U be a (V_1,V_2)-node separator of G s.t. 1) $F\subseteq E(V_1)$, 2) all links in F are not of the type $(v_1,v_2), v_1\in V_1, v_2\in V_2$, 3) all nodes in V_1, which are adjacent to nodes in V_2 (denoted as $N_{V_2}(V_1)$) have at most degree $|F|+1$. Let $t\in V_2$ s.t. there is no path to t in $E(V_1)\setminus F$ from any node in $N_{V_2}(V_1)$, but in $E\setminus F$. Let $R_{t,F}^{(V_1,V_2)}$ be the shortest path from any node in $N_{V_2}(V_1)$ to t only using edges from $E\setminus F$ and, over all nodes $v\in V_2, v'\in V_1$, with $F'=E(v')$, let $R_{t,F'}^{(V_1,V_2)}$ be the maximum length, of all shortest paths from nodes $v\in N_{V_1}(V_2)$ to t only using edges from $E\setminus E(v')$. Then, the competitive additive stretch of any $|F|$-resilient local fast failover algorithm A, matching on in-port and destination, on G is at least, for all eligible t,V_1,V_2,F:*

$$\max_{t,V_1,V_2,F} R_{t,F}^{(V_1,V_2)}-R_{t,F'}^{(V_1,V_2)}-1\,. \tag{1}$$

Proof: We start by considering fixed t,V_1,V_2,F fulfilling the lemma requirements. Observe that A must provision routes from all nodes in $N_{V_2}(V_1)$ to t when the links F fail, where the shortest of such routes has the length $R_{t,F}^{(V_1,V_2)}$. Let e be the first link used by $R_{t,F}^{(V_1,V_2)}$, i.e., e is from some $v_1\in V_1$ to some $v_2\in V_2$. After traversing e, the route is deterministically predefined, never encountering incident links from F. We can furthermore enforce that e will be traversed by A, by setting the failure set F' as all links incident to v_1 except e, with $|F'|\leq|F|$. Now, being at node v_2, both failure sets F,F' are indistinguishable to A, i.e., the remaining route of $R_{t,F}^{(V_1,V_2)}$ will be used. On the other hand, the globally optimal route from v_2 to t has at most the the length $R_{t,F'}^{(V_1,V_2)}$, with one additional hop from v_1. As such, we proved a competitive additive stretch of $(R_{t,F}^{(V_1,V_2)})-(R_{t,F'}^{(V_1,V_2)}+1)$ for fixed eligible t,V_1,V_2,F, from which the lemma statement follows directly. □

We can now prove Theorem 1 in a succinct fashion, using Lemma 1 as follows for (k,ℓ)-clique-torus graphs: a local algorithm cannot distinguish the situation where 1) all links between two cliques failed, forcing a long detour, and 2) being enforced to take a hop on the long detour, by a dense cluster of failures which leaves a short detour intact.

Proof of Theorem 1: We pick t from clique 1 and set F as the $k-1$ links between clique $1,2$, with V_1 being clique 2 and V_2 being $V\setminus V_1$, where the picked t,V_1,V_2,F fulfill the requirements of Theorem 1. The theorem statement follows from $R_{t,F}^{(V_1,V_2)}\geq\ell-1$ and that $R_{t,F'}^{(V_1,V_2)}+1\leq 5$ (1 hop for e, 1 in C_3, 2 to reach C_1, at most 1 extra to reach $t\in C_1$). □

Combining the fact that no deterministic local algorithm can have a better competitive additive stretch than $\Omega(\ell)$ with the fact that a (k,ℓ)-clique-torus graph has $(k-1)\ell$ nodes, i.e., $\ell\in\Omega(n/(k-1))$, yields the following:

Corollary 1. *For all $k\geq 3$, deterministic local fast failover algorithms resilient to $k-1$ failures, matching destination and in-port, have competitive additive stretch of $\Omega(n/(k-1))$.*

```
Algorithm 1: Basic Arc-Swap Operation
    Input: valid arborescence-based decomposition
    Output: modified valid arborescence-based
            decomposition
1   given a node $v$ and two outgoing arcs $e, e'$
2   if arborescence conditions hold for $e, e'$ then
3       swap arborescences
```

3 The Postprocessing Framework

This section presents our algorithmic framework to post-process arborescence-based network decompositions for improved resilience and performance. In the following, we first present the general framework, before we discuss concrete use cases. In particular, we do not make any assumptions on the given arborescence-based network decompositions nor the (re)routing strategy used, which can be arbitrary (specific examples will be considered in our simulations).

The framework can be used to optimize a large set of objectives. We consider two classes of objectives in this paper and present two examples each. In the first class, we aim to improve traffic-engineering metrics of failover routes (like load or stretch) or account for flow or application *priorities*, given certain assumptions about a traffic scenario and failure model, *without sacrificing maximum resilience*. As a shorthand, we will refer to this first class as the ***traffic scenario***. In the second class, the concrete routing mechanism is ignored and *properties of the decomposition*, e.g., depth and independence, are improved, which can lead to shorter paths and higher resilience respectively.

We will refer to this second class as the ***decomposition***. In the remainder of this section, we introduce our framework without concrete instantiations of objective functions, which we will cover in the next section.

At the heart of our postprocessing framework lies an *arc-swapping* algorithm which can come in different flavors, depending on the use case. All the different variants of the arc-swapping algorithm have in common that they *always preserve connectivity*: if a source-destination pair features a certain property that influences the objective, then this property can only be improved in each arc-swap operation. In particular, these swaps must maintain the arborescence character of the decompositions, i.e., they cannot introduce cycles.

The general principle is quite simple (see Algorithm 1): we only swap the arborescences of two outgoing arcs of the same node v and ensure that no cycles are generated. For simplicity we refer to the set of arcs that do not belong to any arborescence as the 0-*arborescence*, even though they do not form an arborescence. This allows us to treat all arcs in a uniform manner and simplifies the description of our algorithm.

More formally, we revisit the approach use to generate arborescences as in e.g. [9], where arcs are added to arborescences incrementally until no further arcs can be added. When this situation is reached, arcs belonging to different arborescences and possibly the 0-arborescences are swapped to allow the process to continue. The (incomplete) arborescence set is denoted by $\{T_1, \ldots, T_k\}$. When growing an arborescence T_i, the following minimal conditions must

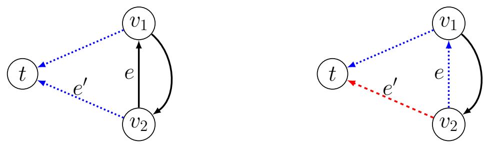


Fig. 3. Introductory example with three nodes where growing arborescences sequentially can end in a deadlock. On the left side, the dotted *blue* arborescence uses both arcs to t, leaving no possibility for the remaining dashed *red* arborescence to route to the destination. However, after swapping (v_2, t) with (v_2, v_1), the dashed *red* arborescence may use the link (v_2, t) on the right side (and subsequentially, the link (v_1, v_2) to complete the construction).

hold when swapping $e = (u, v) \in 0 -$ arborescence with $e' = (u, v') \in E(T_j)$ to ensure that the resulting arborescences are valid [14]: [1]

(1) u has a neighbor $v' \in V(T_i)$
(2) $e = (u, v)$ does not belong to any arborescence yet, i.e., $e \notin \cup_{\rho=1..k} E(T_\rho)$
(3) $u \notin V(T_i)$
(4) $\exists j,\ s.t.\ e' = (u, v') \in E(T_j)$
(5) $v \in V(T_j)$
(6) v' is not on the path to from v to the root in T_j.

Let us consider an example. Let us assume that arborescences have different colors. In Fig. 3, when we swap the dotted blue arc (v_1, t) to the unused arc (v_1, v_2), the dashed red arborescence may now take over (v_1, t), removing the current deadlock situation. In general, when we cannot add an arc to T_i in the normal round-robin fashion as explained in [14] and discussed further in Section 4.4, we can check for candidate arc pairs $e = (u, v), e' = (u, v')$ leaving node u if we could perform a swapping operation. Analogously to the above conditions, we can formulate the criteria for swapping two arcs belonging to arborescences T_i and T_j.

In contrast to the swapping checks necessary when constructing arborescences, we do not have to test whether each node is incident to an arborescence in this case: this is guaranteed already by the existing decomposition (condition (1,4,5)). Thus, in contrast to the swapping conditions during the arborescence decomposition, there are two cases to consider.

Case (i): $e = (u, v) \in E(T_i), e' = (u, v') \in E(T_j)$. From the above correctness conditions (1,4,5) are always satisfied, while (2,3) are irrelevant. In addition to (6), it must hold that v is not on the path to from v' to the root in T_i. If these conditions are satisfied, then e can be added to T_j and e' can be added to T_i. An example is provided in Figure 4, which improves the depth of both arborescences.

Case (ii), $e = (u, v) \in E(T_i)$ and $e' = (u, v')$ does not belong to any (real) arborescence, $e' \notin \cup_{\rho=1..k} E(T_\rho)$. In this case, (1) is always satisfied and (2-6) are irrelevant. Instead, to be able to remove e from T_i and replace it with e' it must hold that v does not belong to the path from v' to the root in T_i. An example is provided in Figure 5, which gives a better depth for the dashed red arborescence.

If the conditions are met, then the arborescence set after the swap is still valid. The time complexity of picking all

[1] [9] and similar approaches use additional criteria which are immaterial to this discussion

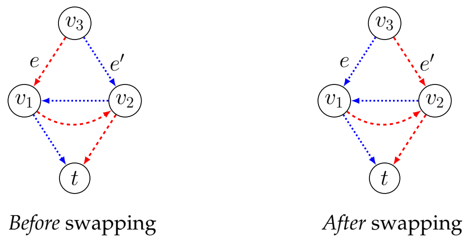


Fig. 4. Example for the swapping in Case (i).

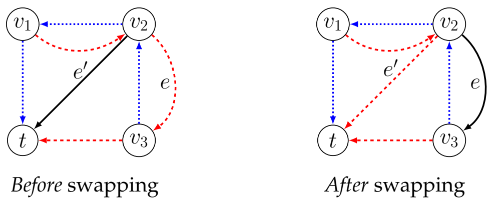


Fig. 5. Example for the swapping in Case (ii).

candidate pairs is in $O(n^2\Delta)$, where Δ is the maximum node degree.

Based on this arc-swap operation, the idea of our algorithmic framework is then to swap arcs only if it improves a certain objective function, see Algorithm 2. Recall that we denote the set of all arcs that do not belong to an arborescence the 0-arborescence and consider it to be always valid (satisfying the condition of line 2 in Algorithm 1). Observe that when two arcs are swapped, they must be outgoing from the same node v: else, if v loses an outgoing arc from some arborescence T, then node v no longer has a way to route to t.

Observation 1. *When exactly two arcs e, e' are swapped in a given valid arborescence decomposition, both must be outgoing from the same node v, else at least one arborescence will be disconnected, i.e., the decomposition is invalid.*

Note that we do not need to check the validity of the arborescence conditions from scratch. It suffices to check if following the outgoing arcs from v, a sink (node without outgoing arcs) before or beyond the destination was created or a cycle was added, as we started with a valid arborescence-based decomposition. In order to generate a new sink, namely a node without any outgoing arborescence links, some node besides the destination must be in some arborescence without a corresponding outgoing edge. Algorithm 1 does not violate this condition as both edges are outgoing from v, i.e., no new sink will be generated. It remains to check for cycles, which can only appear in the two affected arborescences and must contain the arc e or e'. Recall that we do not need to check the 0-arborescence. Hence, we can efficiently validate a swap by simply following the unique outgoing edges of the respective arborescence outgoing from v, terminating if either the original v (cycle) or the destination t is reached. The length of such a path is limited by the prior depth of the arborescence, plus an extra hop for the arc e or e'.

Observation 2. *Checking if a swap is valid in Algorithm 1 can be performed in a runtime in the order of the depth of the arborescence, hence in $O(n)$.*

Algorithm 2: Generic Post-Processing Algorithm

```
    Input: valid arborescence decomposition
    Output: improved (if possible) decomposition
1   improved ← $\top$;
2   while improved do
3       improved ← $\bot$;
4       for all nodes $v \in V$ do
5           for all pairs of outgoing edges $e, e'$ of $v$ do
6               apply Algorithm 1 to $e, e'$;
7               if objective function improves then
8                   improved ← $\top$;
9               else
10                  undo swap of $e, e'$
```

We now show that Algorithm 2 is correct:

Theorem 2. *The algorithmic framework in Algorithm 2 never introduces cycles and always converges.*

Proof: We start with the convergence. During the execution of the algorithm, the sequence of "best" decompositions so far, does not contain any repetitions, since a swap is only kept if the objective function improves. The number of swapping candidates examined in line 4 and 5 is bounded by the number of edge pairs at all nodes, i.e., $n \cdot \binom{n}{2} \in O(n^3)$. Hence, as the number of different arborescence decompositions are finite, convergence is guaranteed. Lastly, as edges may only be swapped if the arborescence conditions are not violated, we do not introduce any cycles. □

As our algorithm framework allows for *any* objective function, it might in the worst case try all possible arborescences. For k arborescences, each edge has $k+1$ options, one extra for not belonging to any arborescence, bounding the number of decompositions from above by $O((k+1)^{|E|})$.

However, we only perform swaps if the objective function improves, which is beneficial for, e.g., arborescence depth optimization, where the maximum depth is less than $|V| = n$. Hence, even if we need to check all $O(n^3)$ edge swap possibilities each time, at a runtime of $O(n)$ each, we obtain a polynomial runtime.

Corollary 2. *Given some objective function x and a graph $G = (V, E)$, let $x_{\#values}(G)$ be the number of different values x evaluates to on G over all possible valid arborescence decompositions. If x can be evaluated in $x_{runtime}(n)$, then the runtime of Algorithm 2 on G for x is bounded by $O(n^3) \cdot \max\{x_{runtime}(n), O(n)\} \cdot x_{\#values}(G)$.*

Depending on the type of optimizations performed, the complexity of computing the gains can differ: in the *traffic scenario* case, the procedure and routing can be simulated; in the *decomposition* case, calculating the improvement per swap involves only the two affected arborescences.

Moreover, we note that our algorithmic framework can also be generalized to swap *multiple* (i.e., more than two) arcs before an improvement of the objective function is required, even from multiple nodes at once. While the validity checking remains tractable[2], the search space grows non-polynomially in Algorithm 1. As thus, we limit ourselves to swapping two edges at once.

[2] E.g., check each of the k arborescences from scratch by DFS in $\theta(kn)$.

4 Use Cases and Evaluation

Our framework for postprocessing a decomposition can be configured with different objective functions, depending on the specific needs. In the following, we discuss and evaluate different use cases, namely two traffic scenario optimization use cases (for stretch/load) and two pure network decomposition optimizations (SRLG and independent paths).

For the experimental evaluation we generate 100 instances of undirected (bi-directional) 5-regular random graphs with 100 nodes with the NetworkX library[3] implementation of Steger and Wormwald's algorithm [16]. We then generate the corresponding arborescences, i.e., five for each graph, picking a random root, and finally optimize those arborescences for the use cases mentioned above. We then compare the unoptimized and optimized arborescences by failing a fraction of the network links picked at random, and simulate a circular arborescence routing process on the resulting infrastructure. In circular arborescence routing, packets follow an arborescence towards a destination until they either reach their target or encounter a failed link. In the latter case, they continue on the next available arborescence, i.e., if a packet has used arborescence T_i up to the failed link, it will then follow arborescence T_{i+1} provided that the corresponding outgoing link is available, or try arborescences $T_{i+2}, \ldots$ otherwise.

4.1 Impact of the Original Network Decomposition

We first study the impact of the network arborescence decomposition algorithm (that is, the input of the optimization process) on the optimization efficiency, before analyzing the optimization scenario in more detail. To this end, we first compare a *Random* and a *Greedy* decomposition generation algorithm, both always find a valid decomposition.[4] *Random* produces a valid yet randomized set of k arbitrary arborescences, whereas *Greedy* constructs the arborescences in a way that ensures that routes are shorter on arborescences with low indices, while longer detours are accepted for higher indices, which will only come into play for several failures. Both of them are described next in more detail.

Random decomposition. An approach, which successfully builds the arborescences $T_1, \ldots, T_k$ in order, needs to maintain the invariant that the remaining graph has sufficient connectivity. To this end, when we add an arc (u, v) to the (partial) arborescence T_i, where (u, v) does not belong to any $T_j,\ j < i$ and $u \notin T_i, v \in T_i$, we check that $k - i$ arc-disjoint paths remain from u to the root [14]. In the random decomposition, the choice of the arc (u, v) is random from all valid candidates, and is turn iterated until the arborescence T_i contains all nodes, afterwards starting with T_{i+1}, finishing when k arborescences are completed. With respect to the runtime, we can leverage maximum flow algorithms, where for k arc-disjoint paths, we need k augmenting paths, each in $O(|E|)$. Hence, as each arc of the $O(E)$ arcs might get tested $O(k)$ times, the construction finishes in $O(|E|^2k^2)$.

Greedy decomposition. The greedy decomposition is analogous to the random decomposition and is used for the experimental evaluation in [17]. The only conceptual difference is that instead of choosing randomly from all possible arc candidates, we pick one of the arcs that increases the depth of the current arborescence T_i the least (ideally by 0). Prior work [17] showed that the greedy decomposition, in combination with circular rerouting, leads to good failover performance, as the depth of the arborescences $T_1, \ldots, T_k$ increases monotonically. For the runtime, the arc testing can again be performed in $O(|E|k)$, leading to a total runtime of $O(|E|^2k^2)$ as well. This includes the greedy aspect, as each node can store its current depth and we only have unit arc weights, else one could implement, e.g., a priority queue.

[3] https://networkx.github.io/ [4] Evaluation results for a recent heuristic [14] are discussed in Section 4.4.

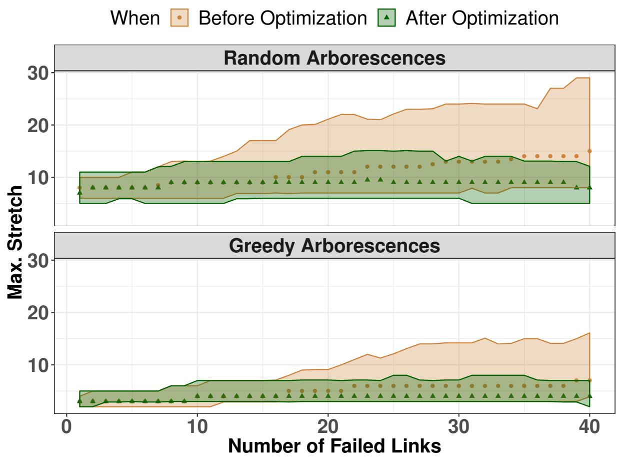


Fig. 6. Efficiency of stretch optimization when optimizing Random (*top*) or Greedy (*bottom*) arborescences, facing random failures. Each point represents the median metric value over 100 independent trials. The shaded area delimits the 10 (resp. 90) quantile values.

Stretch comparison. Figure 6 presents the maximum stretch values recorded before and after stretch optimization. First, one can observe that the *Random* arborescence decomposition (*top*) performs worse than the *Greedy* arborescences decomposition before optimization (*bottom*): for instance facing $x = 20$ random link failures, the median stretch is 11 for *Random* and only 5 for *Greedy*, and 10% of the samples have a stretch above 22 for *Random*, and only above 9 for *Greedy*. Interestingly, the stretch optimization is efficient on both structures, producing arborescences that maintain a lower stretch compared to unoptimized arborescences, especially under high numbers of failures. However, even when optimized, arborescences originally produced by *Random* still perform worse than *Greedy* arborescences: the original performance gap between *Greedy* and *Random* is not completely filled by the optimization process. In the following, we therefore focus on optimizing *Greedy* network decompositions.

4.2 Optimization Use Cases

Reducing Route Stretch. A first fundamental objective is to ensure that failover routes are *short*. The idea is hence to perform edge swapping such that route lengths under failures are reduced. To this end, given a set $\mathcal{F}$ of possible link failures, we postprocess the network decomposition to minimize the maximum additive route stretch of a subset of "important" nodes for the case the links $\mathcal{F}$ fail.

More specifically, the objective function ensures the following. Given a subset of nodes that are deemed crucial and need to send packets to some destination node (the root of the arborescence) as well as a set of links highly susceptible

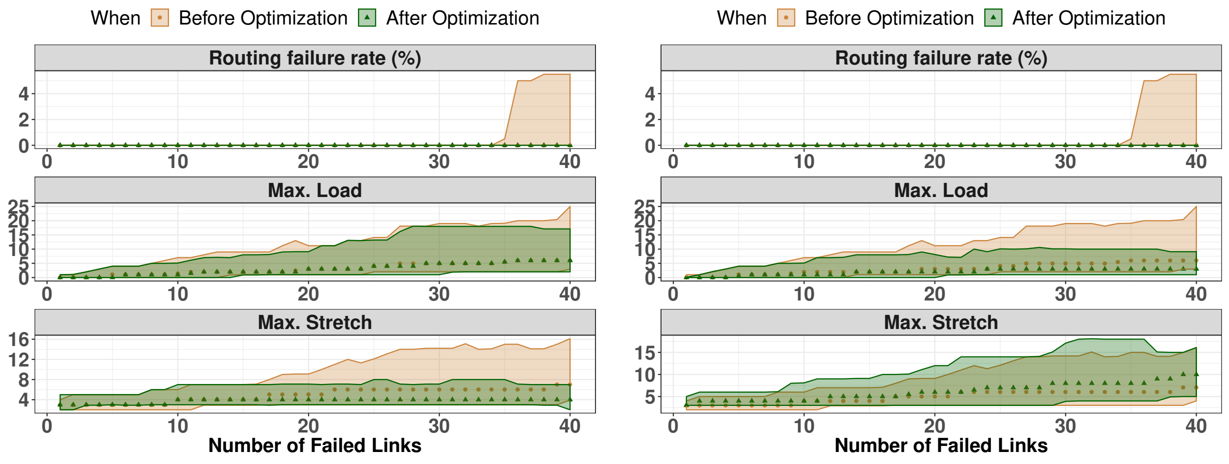


Fig. 7. Resilience, stretch and load worst case performances of greedy structures, before and after improvement, when improvement targets stretch (*left*) and load (*right*), facing random failures. Each point represents the median metric value over 100 indep. trials. Ribbon delimits the 10 (resp. 90) quantile values.

to failures, the packets should reach the destination even if all these links or a subset of them failed with short detours. In other words, as long as an edge swap does not reduce connectivity (i.e., no pair is disconnected) of circular routing, we execute the swap if it strictly reduces the maximum (route length - shortest path) over all pairs under this set of failures $\mathcal{F}$.

Figure 7 (*left*) presents the impact of this optimization approach, measured by three metrics capturing the performance of the circular arborescence routing scheme exploiting the unoptimized and optimized versions of the arborescence decomposition. The traffic we simulate consist in the sending of a flow of size 1 by 20% of the network nodes to a random destination despite a set of 0 to 40 random link failures.[5]

The first metric is the number of routing failures encountered. It shows that both unoptimized and optimized arborescences manage to keep the number of failures very low. Even under a high number of failures (e.g. 40), the median of routing failures is 0 in both optimized and unoptimized arborescences, only the 10% worst unoptimized arborescences seem to raise to a low 5% failure rate. The two next metrics are maximum load and maximum stretch under the same traffic. While both optimized and unoptimized arborescences exhibit roughly the same loads, the stretch of the optimized arborescences is consistently lower than in the unoptimized case and the quantiles are much narrower. This shows the efficiency of our stretch optimization. Interestingly, optimizing the stretch only induces a slight increase in the load, though one expects a trade-off between stretch and load. Intuitively, lower stretch induces higher load, as for low stretch many flows use the same "good" links. For low load some flows must take detours, so in general optimizing for low load leads to higher stretch, as we will see in our next experiments.

Reducing Load. A second fundamental objective is to ensure that failover routes do not overload the network. To that end, we propose an objective function similar to the one used above. Given a set $\mathcal{F}$ of possible link failures, we postprocess the network decomposition to minimize the load, defined as the maximum number of times a link is used due to rerouting messages for the case the links $\mathcal{F}$ fail.

Thus, as long as an edge swap does not reduce connectivity (i.e., no pair is disconnected) of circular routing, we execute the swap if it strictly reduces the maximum of additional flow on edges when re-routing over all pairs under the failure set $\mathcal{F}$.

Figure 7 (*right*) presents the impact of this optimization, captured again along 3 metrics assessed by simulating the sending by 20% of the network nodes of a flow of unit size to a random node in the network. One can first observe (*top*) that this optimization has an impact on the routing failure rate: before optimizing, some packets do not reach their destination, but after swapping, the failure rate is 0.

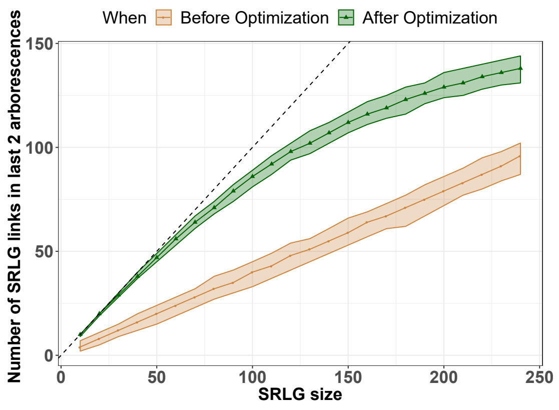
The two next metrics exhibit a mirrored trend compared to the figure of stretch: optimizing load efficiently reduces the load in both median and 10% worst cases. This effect increases with the failure rate, and under a high number of failures (e.g. 40) the median maximum load drops from 5 to 2 thanks to the optimization. This optimization however has a slight impact on the stretch, and load-optimized arborescences exhibit a stretch distribution globally above the stretch distribution of non-optimized arborescences.

We conclude from both Figures 7 left and right that optimizing arborescences for load or stretch is efficient, in the sense that the optimized metric is effectively reduced by the optimization. Overall, there is only a very small loss in the un-optimized criteria (stretch degrades when optimizing load, load degrades when optimizing stretch).

Note that these objective functions were chosen to illustrate the power of our framework for optimizing the rerouting in a certain traffic scenario. While we picked a random set of nodes and links to be sources and error-prone respectively, this approach can be used in traffic engineering to optimize important scenarios with varying failure sets etc. Instead of focussing on one optimization criterion like load or stretch, more complicated objectives that give weights to certain outcomes can be constructed. Theorem 2 guarantees that the optimization converges.

Shared Risk Link Groups. Link failures are often interdependent, if links in a network share a common fiber or other

[5] We omit results from other failure set sizes as they exhibit the same properties



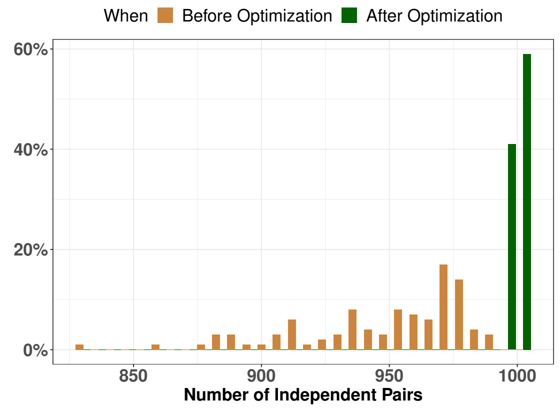


Fig. 8. *(left)* Number of SRLG links in the last two arborescences before and after SRLG optimization, for varying SRLG sizes. Dashed line represents the ideal case where all SRLG links end up in the last two arborescences. *(right)* Distribution of the number of independent pairs before and after optimization on 4800 graphs.

physical attributes (e.g., if they are close geographically [12]), and thus may fail simultaneously. Such dependencies are known as shared risk link groups *SRLGs* [11], [12], [13].

Existing arborescence-based network decompositions do not account for SRLG. In fact, the main objective of most existing FRR algorithms is to preserve connectivity of up to $k-1$ failures in k-connected networks, no matter *where* these failures occur and *even if the remaining network remained highly connected with more failures*. Note that a network may keep the same connectivity despite additional failures, if the failures do not affect nodes that suffer from reduced connectivity already.

Accordingly, we show how to use our algorithmic framework to improve the connectivity provided by an arborescence-based network decomposition (based on circular routing), exploiting information about SRLGs. A basic observation one can make is the following. Since in a k-connected network, a decomposition can only consist of k edge-disjoint arborescences, it is generally not possible to tolerate more than $k-1$ link failures. Furthermore, if the links of a large SRLG are distributed across many arborescences, the failure of this SRLG can easily disrupt connectivity: there are simply no arborescences left that provide a valid route to the destination.

However, the situation looks different if we are able to collocate the links of a SRLG in a small number of arborescences.[6] The failure of this SRLG, even if it is large, will only affect this arborescence set. When encountering the first failure in this SRLG, the routing mechanism can then simply re-route traffic to other arborescences unaffected by any failures in the same group. The beauty of this approach lies in the fact that the routing mechanism does not enter the objective function in this case.

To implement this idea, we can hence exploit our algorithmic framework to swap edges in such a way that links from an SRLG are assigned to the same arborescences. Thus, if all these links fail simultaneously, they just affect a small number of arborescences, *independently* of the number of failures. All other arborescences stay intact, and are available in the case of further failures. In particular, we can simply use the following approach. Given a SRLG, we select two or more arborescences to contain the SRLG arcs and then we iterate over all edges in this set, e.g, the two with the highest index T_{k-1}, T_k. In this case, we pick an edge (u, v) in SRLG but not yet in the SRLG arborescences T_{k-1} and T_k and we check for all outgoing links of u whether swapping the two edges will increase the number of links in the SRLG to be assigned to T_{k-1} or T_k(Algorithm 3). After running this algorithm, we will have a maximal number of SRLG links in T_{k-1} and T_k. In circular arborescence routing these two arborescences will be selected last, i.e., after $k-1$ failures occurred and once a packet has encountered two links from this set it will not be routed to another link of this set unless $k-2$ more failures happen or not all SRLG links have been assigned to T_{k-1}, T_k.

Algorithm 3: Postprocessing for SRLG

```
  Input: valid decomposition, SRLG $S$
  Output: $s \in S$ belong to $T_{k-1}, T_k$ (if possible)
1 for each link $(u, v) \in S$ do
2     for each $v'$ s.t. $(u, v') \in T_{k-1}$ or $T_k$ do
3         if $(u, v') \notin S$ swap$((u, v),(u, v'))$ valid then
4             break                          /*inner for-loop*/
```

Note that the runtime of Algorithm 3 is linear in the SRLG's size (which can be much smaller than the network size n): we can simply try to swap an SRLG link with one of the outgoing links belonging to the SRLG arborescences.

This scheme can be combined with circular routing by ensuring that all SRLG arborescences are indexed consecutively and can thus be skipped once a failed SRLG link is encountered. Moreover at nodes that belong to network parts with a high likelihood of SRLG link failures can always skip these arborescences even if no failures occur to increase the chances of staying clear of them.

To evaluate the effects of our approach, we randomly generated networks as above and selected random SRLGs of varying size. We then measured the capacity of our scheme to move those links towards the last two arborescences. These

[6] In general it is not possible to put an arbitrary set of links into one single arborescence, e.g., only one directed link of a symmetric connection can be in one arborescence and from each node there is only one outgoing link for each arborescence.

results are represented Figure 8 (*left*): The fraction of SRLG links in the last arborescences is constant before optimization (since SRLG links are picked at random). One can observe that the optimized arborescences all manage to pack a greater fraction of the SRLG links into the last arborescences. More precisely, the algorithm nearly reaches a perfect optimization when there are few SRLG links (e.g. less than 50). When the number of SRLG links increases, the algorithms manages to put proportionally less such links in the last arborescences. This is due to the last arborescences "saturating": recall that those 2 last trees can only contain 200 links in total, and each must still connect all the nodes, therefore increasing the difficulty of the optimization.

These results confirm the ability of our algorithmic framework to optimize arborescences based on SRLG criteria.

Note that SRLGs are not random subsets of edges in practice. Unfortunately, we could not find a suitable dataset or other information to construct a more meaningful experiment. Observe that the performance of arborescence-based FRR for SRLGs depends on the density of the subgraph the SRLG forms. The more arcs of a node belong to an SRLG the more arborescences will contain SRLG arcs, even after the optimization.

Providing Independence. Lastly, we focus on a case study that concerns the independence of the decomposition. Two paths are independent if they do not share any nodes except their source and destination. This property is useful to deal with node failures where all incident links to a node fail simultaneously. The more such independent path pairs exist the more (node failures)-resilient arborescence-based rerouting can be. In particular, maximal resiliency of $k-1$ can be achieved with circular routing if all node pairs use independent paths in each arborescence.

Note that this node failure case usually cannot be modelled with shared risk link groups as we cannot cover all incident links to a node with a single arborescence in general.

Figure 8 (*right*) presents the results of swapping edges with the objective of increasing the number of independent paths from all nodes in all arborescence pairs. To measure how efficiently our approach manages to ensure independent paths, we generated 4800 random 5-regular graphs with 100 nodes and counted the number of independent paths before and after optimization. Per instance $\binom{k}{2} \cdot n = 1000$ pairs are evaluated. This figure shows that before optimization, all topologies have at least 828 independent paths out of 1000. Thus paths are already independent with a high probability (949/1000 on average), and that this quantity varies considerably across networks (high dispersion of values). After the optimization, pairs are independent with a high probability (999/1000 on average) in nearly all the cases (lowest recorded value is 997, 1000 reached in 59% of the networks). This confirms the ability of our optimization to produce independent (and therefore more resilient) paths.

4.3 Runtime Analysis

We now turn our attention to the runtime of our optimization framework. To test this, we measured the wall clock times of the optimization processes over random regular graphs of varying sizes and connectivities. The single-threaded code is executed on a 24-core Intel Xeon E5-2620 platform with 32Gb memory. Figure 9 presents the distribution of those results. It shows that optimizing stretch or load on a 80-nodes topology takes on average around 750 seconds. Quite surprisingly, connectivity only has a slight impact on runtime. Runtimes varies roughly quadratically with topology size ($R^2 = .89$).

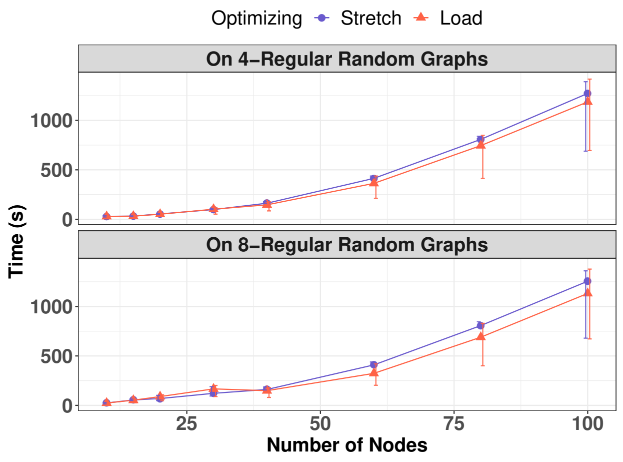


Fig. 9. Impact of topology size on stretch and load optimization wall clock runtimes. Each point represents the mean time in second established over 100 independent random regular graphs. Vertical bars (shifted for readability) represent the the 10 (resp. 90) quantile values.

4.4 Optimizing Network Decomposition Heuristics

So far in this section, we evaluated our postprocessing framework on network decomposition algorithms that *always* yield a valid output. Recent work [14] also proposed a heuristic called *Bonsai* that attempts to generate arborescences of small depth, with no guarantees if a valid output may be produced. Notwithstanding, if successful, the question arises if said arborescences can further benefit by our approach.

Heuristic round-robin decomposition. *Bonsai* [14] proposes to build the k arborescences in parallel (*round-robin*), with the goal of achieving small (i.e., low depth respectively stretch) arborescences. This is in contrast to the random and greedy schemes, which build arborescences sequentially. However, even though the *Bonsai* round-robin scheme outperforms the greedy and random schemes regarding stretch quality in evaluations in [14], it has the downside that it might not produce a valid decomposition. To this end, the authors in [14] propose a further additional swapping heuristic for *Bonsai* to boost their success rate, which however cannot guarantee the output to be valid—unlike the swapping performed in our postprocessing framework, which guarantees valid arborescences.

Results. Maybe surprisingly, the rate of improvement is quite similar to the effects described in Section 4.2 for the *Greedy* approach. In other words, even though the heuristics in [14] were already optimized ahead of time, our postprocessing framework still yielded similar significant improvements for *Bonsai*. The plots are shown in Figures 10 and 11.

4.5 Experiments on Real World Graphs

To complement our experiments on synthetic graphs, we also ran them on well-connected cores of network topologies, taken from the *Topology Zoo* data set [18]. We trim the Topology Zoo graphs s.t. only the well-connected cores

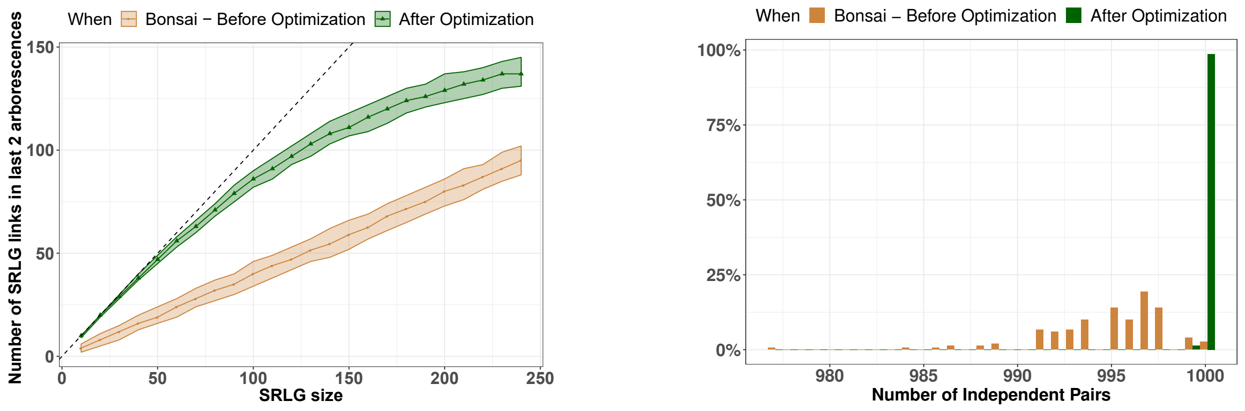


Fig. 10. Effect on *Bonsai* [14]: *(left)* Number of SRLG links in the last two arborescences before/after SRLG optimization, for varying SRLG sizes. Dashed line represents the ideal case where all SRLG links end up in the last two arborescences. *(right)* Distribution of the number of independent pairs before and after optimization, established over 3600 independent random graphs.

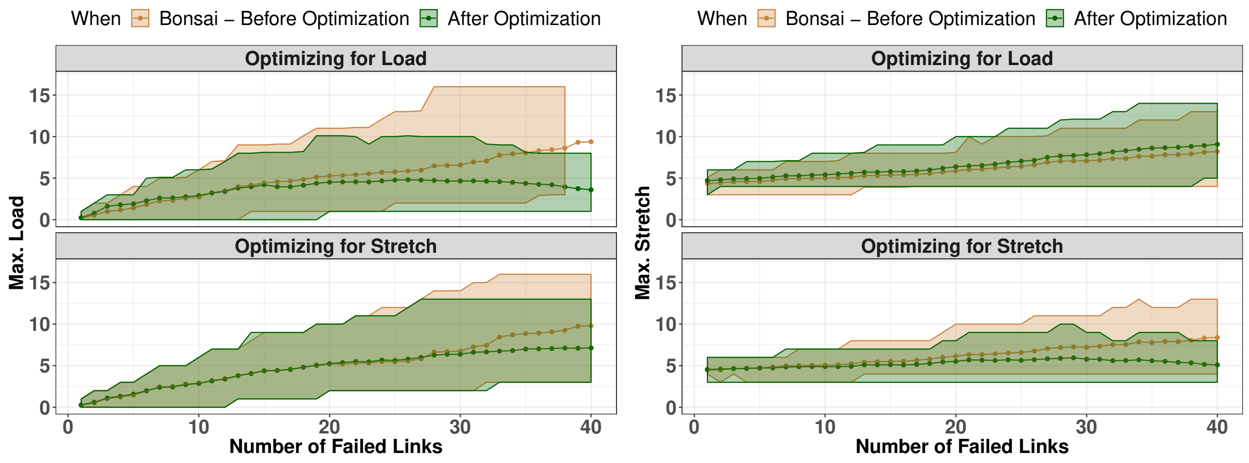


Fig. 11. *Load* (left) and *stretch* (right) performances of *Bonsai* arborescences [14], before and after improvement, when improvement targets load (top) or stretch (bottom), facing random failures. Each point represents the median metric value over 100 indep. trials. Ribbon delimits the 10 (resp. 90) quantile values.

remain, as follows. We first *contract* nodes of bidirected degree 2 into a single bidirected link. Next, we replace nodes that have a degree 3 with three edges between the three affected neighbors. This process is repeated until no more nodes can be contracted or removed. If the trimming resulted in less than 10 nodes, we omit them. All 11 topologies generated like this are of connectivity at least 4. For each such topology (with 10 to 83 nodes and 54 to 111 edges), we pick 20 different nodes uniformly at random, selecting them as a root for the arborescence packings. The results of the experiments are very similar to the results on synthetic graphs. In all cases, the optimizations are computed quickly and yield improvements in the same percentage range as we have observed on synthetic graphs. As a consequence, we do not present plots for these graphs.

5 An Example ILP Model for the Circular Routing Scheme

The existence of a valid circular routing scheme based on k arc-disjoint spanning arborescences in a given network graph containing a known set of failed links can also be analyzed with the aid of Integer Linear Programming (ILP) tools. To illustrate one of the possible approaches, we formulate an example mathematical model of the corresponding ILP optimization problem for path lengths and stretch below. Please note that the proposed formulation may not be suitable for medium and large problems due to significant computational complexity.

Constants:

- A — set of arcs;
- F — set of failed arcs;
- T — set of arc-disjoint arborescences;
- D — set of traffic demands;
- s_d — the source node of traffic demand d.

Variables:

- $f_{vwit} \in \mathbb{B}$ — presence of a unit virtual flow on arc $(v,w) \in A$ associated with arborescence t and source i;
- $a_{vwt} \in \mathbb{B}$ — assignment of arcs to arborescences;
- $x_{vwd} \in \mathbb{B}$ — presence of a unit data flow associated with demand d on arc $(v,w) \in A$;
- $y_{vwd} \in \mathbb{B}$ — presence of a unit virtual flow associated with demand d on arc $(v,w) \in A$;
- $ua_{dt} \in \mathbb{B}$ — 1 if arborescence t is used by demand d;
- $ua\_in_{dwt} \in \mathbb{B}$ — 1 if arborescence t is used by demand d to enter transit node w;

- $ua\_out_{dwt} \in \mathbb{B}$ — 1 if arborescence t is used by demand d to leave transit node w;
- $ua\_pairs_{dwt_at_b} \in \mathbb{B}$ — 1 if arborescences t_a and t_b are used by demand d to enter/leave transit node w, respectively;
- $l_d \in \mathbb{Z}_+$ — # hops/arcs included in the path used by demand d;
- $l_{\max} \in \mathbb{Z}_+$ — # hops/arcs included in the longest path among all demands;
- $ps_{\max} \in \mathbb{Z}_+$ — maximum path stretch.

The objective function:

$$\min z = l_{\max} + ps_{\max} + \sum_{(v,w)\in A}\sum_{d\in D} y_{vwd} + \sum_{d\in D}\sum_{t\in T} ua_{dt} + \sum_{d\in D}\sum_{w\in V\setminus\{s_d,r\}}\sum_{t\in T}(ua\_in_{dwt} + ua\_out_{dwt}) \quad (2)$$

Constraints:

$$\forall_{(v,w)\in A} \sum_{t\in T} a_{vwt} \le 1$$
(1: Arc in one tree)

$$\forall_{t\in T} \sum_{(v,w)\in A} a_{vwt} = n-1$$
(2: Spanning arborescence)

$$\forall_{i\in V\setminus\{r\}}\forall_{t\in T} \sum_{(w,i)\in A} f_{wiit} - \sum_{(i,w)\in A} f_{iwit} = -1$$
(3: Tree sources)

$$\forall_{i\in V\setminus\{r\}}\forall_{v\in V\setminus\{i,r\}}\forall_{t\in T}: \sum_{(w,v)\in A} f_{wvit} - \sum_{(v,w)\in A} f_{vwit} = 0$$
(4: Tree balance)

$$\forall_{i\in V\setminus\{r\}}\forall_{t\in T} \sum_{(w,r)\in A} f_{writ} - \sum_{(r,w)\in A} f_{rwit} = 1$$
(5: Tree sink)

$$\forall_{i\in V\setminus\{r\}}\forall_{(v,w)\in A}\forall_{t\in T} f_{vwit} \le a_{vwt}$$
(6: Tree assignment)

$$\forall_{d\in D} \sum_{(v,w)\in A:w=s_d} x_{vwd} - \sum_{(w,v)\in A:w=s_d} x_{wvd} = -1$$
(7: Traffic sources)

$$\forall_{d\in D}\forall_{w\in V\setminus\{r,s_d\}} \sum_{(v,w)\in A} x_{vwd} - \sum_{(w,v)\in A} x_{wvd} = 0$$
(8: Traffic balance)

$$\forall_{d\in D} \sum_{(v,r)\in A} x_{vrd} - \sum_{(r,v)\in A} x_{rvd} = 1$$
(9: Traffic destination)

$$\forall_{d\in D}\forall_{t\in T}\forall_{(v,w)\in A} \quad x_{vwd} + a_{vwt} \le 1 + ua_{dt}$$
(10: Traffic used trees)

$$\forall_{d\in D}\forall_{t\in T}\forall_{(v,w)\in A:w\notin\{s_d,r\}}: x_{vwd} + a_{vwt} \le 1 + ua\_in_{dwt}$$
(11: Traffic inbound trees)

$$\forall_{d\in D}\forall_{t\in T}\forall_{(w,v)\in A:w\notin\{s_d,r\}}: x_{wvd} + a_{wvt} \le 1 + ua\_out_{dwt}$$
(12: Traffic outbound trees)

$$\forall_{(v,w)\in F} \sum_{d\in D} x_{vwd} = 0$$
(13: Traffic skip failed arcs)

$$\forall_{d\in D}\ l_d = \sum_{(v,w)\in A} x_{vwd}$$
(14: Traffic path length)

$$\forall_{d\in D}\ l_{\max} \ge l_d$$
(15: Traffic the longest path)

$$\forall_{d\in D}\forall_{w\in V\setminus\{s_d,r\}}\forall_{t_a,t_b\in T:t_a<t_b\wedge t_b-t_a>1}: ua\_in_{dwt_a} + ua\_out_{dwt_b} \le 1 + ua\_pairs_{dwt_at_b}$$
(16: Non-consecutive trees A)

$$\forall_{d\in D}\forall_{w\in V\setminus\{s_d,r\}}\forall_{t_a,t_b\in T:t_a>t_b\wedge k-(t_a-t_b)>1}: ua\_in_{dwt_a} + ua\_out_{dwt_b} \le 1 + ua\_pairs_{dwt_at_b}$$
(17: Non-consecutive trees B)

$$\forall_{d\in D}\forall_{w\in V\setminus\{s_d,r\}}\forall_{t_a,t_b\in T:t_a<t_b\wedge t_b-t_a>1}: \sum_{(w,v)\in A\cap F}\sum_{t_i\in T:t_i>t_a\wedge t_i<t_b} a_{wvt_i} \ge (t_b - t_a - 1)\cdot ua\_pairs_{dwt_at_b}$$
(18: Prohibited rerouting A)

$$\forall_{d\in D}\forall_{w\in V\setminus\{s_d,r\}}\forall_{t_a,t_b\in T:t_a>t_b\wedge k-(t_a-t_b)>1}: \sum_{(w,v)\in A\cap F}\sum_{t_i\in T:t_i>t_a\vee t_i<t_b} a_{wvt_i} \ge (k - [t_a - t_b] - 1)\cdot ua\_pairs_{dwt_at_b}$$
(19: Prohibited rerouting B)

$$\forall_{d\in D} \sum_{(v,w)\in A:w=s_d} y_{vwd} - \sum_{(w,v)\in A:w=s_d} y_{wvd} = -1$$
(20: Reference paths sources)

$$\forall_{d\in D}\forall_{w\in V\setminus\{r,s_d\}} \sum_{(v,w)\in A} y_{vwd} - \sum_{(w,v)\in A} y_{wvd} = 0$$
(21: Reference paths balance)

$$\forall_{d\in D} \sum_{(v,r)\in A} y_{vrd} - \sum_{(r,v)\in A} y_{rvd} = 1$$
(22: Reference paths destination)

$$\forall_{(v,w)\in F} \sum_{d\in D} y_{vwd} = 0$$
(23: Reference paths skip failed arcs)

$$\forall_{d\in D}\ ps_{\max} \ge l_d - \sum_{(v,w)\in A} y_{vwd}$$
(24: Max path stretch)

The mathematical formulation presented above includes the necessary elements to model the following features:

- finding the set of k arc-disjoint spanning arborescences rooted at node r in a given network graph;
- routing traffic flows associated with user demands d, originating at the corresponding source nodes s_d and terminating at the root node;
- the circular routing scheme;
- computation of the shortest paths in the graph containing failed links;
- minimization of both the maximum path stretch and the maximum path length.

In this example, the primary objective is to minimize the maximum path length among all traffic demands ($l_{\max}$), while also minimizing the maximum path stretch, $ps_{\max}$. The remaining terms in Formula (2) guarantee that the corresponding binary variables are set to 0, unless the positive value is required to satisfy the constraints.

The first group of constraints (1: Arc in one tree) guarantees that each arc in the network graph belongs to at most one of k arc-disjoint spanning arborescences covering the graph. Then, constraints (2: Spanning arborescence) ensure that the arborescences contain exactly $n-1$ arcs each. To be able to construct valid arborescences in the graph, we rely on virtual unit flows and we introduce the corresponding flow conservation constraints in groups (3: Tree sources)-(5: Tree sink). The first of the three groups is related to sources of virtual flows, the second — to transit nodes (virtual flows must be forwarded without losses), and the third group — to the destination. As an example, if variable f_{vwit} equals 1, it means that arc $(v,w)\in A$ associated with arborescence t carries the unit flow from the source node i to the destination (the root node of the arborescence,

r). Constraints (6: Tree assignment) assign arcs to particular arborescences based on the values of f_{vwit} for all possible indices.

Transmission of user data is modeled in a similar way — we introduce the corresponding flow conservation constraints in groups (7: Traffic sources)-(9: Traffic destination), relying on binary variables x_{vwd} describing the presence of the unit data flow associated with demand d on arc $(v, w) \in A$. Note that we do not model the load of particular arcs in the current ILP formulation. Variables x_{vwd} may also be used to determine the path associated with each demand. Failed links cannot be used by data flows, which is guaranteed by constraints (13: Traffic skip failed arcs). The lengths of the resulting forwarding paths are set by constraints (14: Traffic path length), whereas the maximum observed path length among all traffic demands is determined based on inequalities (15: Traffic the longest path). Note that the value of $l_{\max}$ is minimized.

To be able to enforce the deterministic circular routing scheme in the network, we first determine which arborescences are used to forward traffic associated with particular demands — see constraints (10: Traffic used trees). In the next step, as it is important to track which arborescences are used to enter and leave a transit node on the flow's path, we introduce the corresponding constraints in groups (11: Traffic inbound trees)-(12: Traffic outbound trees). Then, we eliminate the forbidden combinations of used arborescences, which is enforced by the following groups of constraints (16: Non-consecutive trees A)-(19: Prohibited rerouting B).

Finally, to be able to minimize the maximum path stretch among all user demands d in the network graph containing failed links (arcs belonging to the set F), we first introduce additional virtual unit flows to find the shortest paths between the source nodes s_d and the root node r, and then, we determine the maximum path stretch based on the difference in length between the actually used paths (circular routing) and the reference paths (the shortest paths avoiding the failed links). The corresponding constraints are defined in groups (20: Reference paths sources)-(24: Max path stretch).

The example formulation presented above can be modified in different ways, according to specific needs. For instance, only a subset of constraints and variables might be considered to find k arc-disjoint spanning arborescences in a given graph.

6 Related Work

Link failures are the norm rather than the exception in large networks, and have recently led to several outages, as reported in, e.g., [19], [20], [21], [22]. Many existing robust routing mechanisms in the literature, while tolerating *multiple* concurrent failures, involve the control plane and are hence slow. A well-known example are link reversal algorithms [23], [24], [25], [26], which require dynamic router tables and long convergence times, quadratic in the number of nodes [27]. While there exist interesting approaches to implement link reversal algorithms also in the data plane [6], they do not affect the other main drawbacks of link reversal algorithms. Many solutions in the literature also rely on packet-header rewriting [28], [29], [30], [31] or packet-duplication [32]. However, the former consumes header space and the latter introduces additional loads, which is undesirable. Another approach in the literature pre-computes multiple paths s.t. even in the event of multiple failures, the ingress switches can reroute the traffic efficiently without additional computational overhead [33]. Notwithstanding, packets currently en route on a failure-ridden path are not protected by such schemes.

We in this paper are interested in *static* fast rerouting algorithms in the data plane, which rely on precomputed failover rules and do not require packet header rewriting. Our model is hence closely related to the papers by Feigenbaum et al. [34], Chiesa et al. [7], [8], Elhourani et al. [29] and Stephens et al. [35], [36] which all study reachability even under multiple failures. In contrast to our work, however, they do not account for performance aspects of the computed failover paths.

The work in [37] provides stretch guarantees for some special graph classes, such as Hypercubes, Tori, Grids, and Clos-/BCube-topologies, but does not apply to general networks, the focus of this paper. There is also work on algorithms for *constructing* (implicit) network decompositions with certain properties from scratch [14], [38], [39], [40]. Except [14], all approaches require to match also the packet source, not only the in-port, and some of them rely on computationally expensive preprocessing (to compute block designs). [14] proposes a heuristic that attempts to produce arborescences of small depth, which may not always succeed. That said, we see both works as orthogonal, as our approach in this paper could be leveraged to optimize also the network decompositions described in these papers. The approach in [38] is also less general than our framework, and can e.g., not be used to account for special failure scenarios, such as shared risk link groups, simultaneous geographically-correlated failures of multiple network elements [41], or requirements of communication pairs. Indeed, while shared risk link groups (and their characterization) has been studied intensively in the literature, e.g., see [12]. We are not aware of any work on accounting for such risk groups in state-of-the-art decomposition-based FRR mechanisms.

Finally, we note that our results also have applications in other contexts, such as multicasting, which also rely on arborescence decompositions [42], [43], [44].

7 Conclusion

This paper was motivated by the computational challenges involved in computing network decompositions which do not only provide basic connectivity but also account for the quality of routes after failures. We proposed and evaluated a simple solution which improves an arbitrary network decomposition, using fast postprocessing, in terms of basic traffic engineering metrics such as route length and load. Furthermore, we showed that our framework can also be used to improve resiliency for shared risk link groups: an important extension in practice.

We understand our work as the first step and believe that it opens several interesting avenues for future research. In particular, it will be interesting to study alternative postprocessing algorithms, and derive formal performance guarantees for them. It would also be interesting to study further use cases for our framework, beyond the ones given in this paper, e.g., for SRLGs combined with load and stretch.

Lastly, in order to guarantee reproducibility and facilitate other researchers to build upon our algorithms, our code is publicly available at https://gitlab.cs.univie.ac.at/ct-papers/fast-failover.

Acknowledgments

Research supported by the WWTF project WHATIF (ICT19-045). A preliminary version of this work appeared in the proceedings of the 38th International Symposium on Reliable Distributed Systems (SRDS) [45]. We would like to thank the reviewers of SRDS and TDSC for their helpful comments.

References

[1] S. Peter, U. Javed, Q. Zhang, D. Woos, T. Anderson, and A. Krishnamurthy, "One tunnel is (often) enough," in *ACM SIGCOMM CCR*, vol. 44, no. 4. ACM, 2014, pp. 99–110.

[2] A. K. Atlas and A. Zinin, "Basic specification for IP fast-reroute: loop-free alternates," *IETF RFC 5286*, 2008.

[3] A. Kamisiński, "Evolution of IP fast-reroute strategies," in *Proc. International Workshop on Resilient Networks Design and Modeling*, 2018.

[4] P. Pan, G. Swallow, and A. Atlas, "Fast reroute extensions to RSVP-TE for LSP tunnels," in *Request for Comments (RFC) 4090*, 2005.

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

[6] J. Liu, A. Panda, A. Singla, B. Godfrey, M. Schapira, and S. Shenker, "Ensuring connectivity via data plane mechanisms," in *Proc. NSDI*, 2013.

[7] M. Chiesa, A. V. Gurtov, A. Madry, S. Mitrovic, I. Nikolaevskiy, M. Schapira, and S. Shenker, "On the resiliency of randomized routing against multiple edge failures," in *Proc. ICALP*, 2016.

[8] M. Chiesa, I. Nikolaevskiy, S. Mitrovic, A. Panda, A. Gurtov, A. Madry, M. Schapira, and S. Shenker, "The quest for resilient (static) forwarding tables," in *Proc. IEEE INFOCOM*, 2016.

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

[10] J. Edmonds, "Edge-disjoint branchings," *Combinatorial algorithms*, vol. 9, no. 91-96, p. 2, 1973.

[11] L. Shen, X. Yang, and B. Ramamurthy, "Shared risk link group (SRLG)-diverse path provisioning under hybrid service level agreements in wavelength-routed optical mesh networks," *IEEE/ACM Transactions on Networking*, vol. 13, no. 4, pp. 918–931, 2005.

[12] J. Tapolcai, B. Vass, Z. Heszberger, J. Bıró, D. Hay, F. A. Kuipers, and L. Rónyai, "A tractable stochastic model of correlated link failures caused by disasters," in *Proc. IEEE INFOCOM*, 2018.

[13] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: measurement, analysis, and implications," *ACM SIGCOMM CCR*, vol. 41, pp. 350–361, 2011.

[14] K.-T. Foerster, A. Kamisiński, Y.-A. Pignolet, S. Schmid, and G. Tredan, "Bonsai: Efficient fast failover routing using small arborescences," in *Proc. IEEE/IFIP DSN*, 2019.

[15] A. Borodin and R. El-Yaniv, *Online computation and competitive analysis*. Cambridge University Press, 1998.

[16] A. Steger and N. C. Wormald, "Generating random regular graphs quickly," *Combinatorics, Probability and Computing*, vol. 8, no. 4, pp. 377–396, 1999.

[17] M. Chiesa, I. Nikolaevskiy, S. Mitrovic, A. V. Gurtov, A. Madry, M. Schapira, and S. Shenker, "On the resiliency of static forwarding tables," *IEEE/ACM Trans. Netw.*, vol. 25, no. 2, pp. 1133–1146, 2017.

[18] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, 2011.

[19] D. Madory, "Renesys blog: Large outage in pakistan," https://dyn.com/blog/large-outage-in-pakistan/.

[20] R. Singel, "Fiber optic cable cuts isolate millions from internet, future cuts likely," https://www.wired.com/2008/01/fiber-optic-cab/, 2008.

[21] Wikitech, "Site issue Aug 6 2012," http://wikitech.wikimedia.org/view/Site_issue_Aug_6_2012, 2012, (last accessed in April 2019).

[22] C. Wilson, "'Dual' fiber cut causes Sprint outage," https://web.archive.org/web/20080906210432/http://telephonyonline.com/access/news/Sprint_service_outage_011006/, 2006.

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

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

[25] V. D. Park and M. S. Corson, "A highly adaptive distributed routing algorithm for mobile wireless networks," in *Proc. INFOCOM*, 1997.

[26] J. L. Welch and J. E. Walter, "Link reversal algorithms," *Synthesis Lectures on Distributed Comp. Theory*, vol. 2, no. 3, pp. 1–103, 2011.

[27] C. Busch, S. Surapaneni, and S. Tirthapura, "Analysis of link reversal routing algorithms for mobile ad hoc networks," in *Proc. SPAA*, 2003.

[28] K. Lakshminarayanan, M. Caesar, M. Rangan, T. Anderson, S. Shenker, and I. Stoica, "Achieving convergence-free routing using failure-carrying packets," in *Proc. ACM SIGCOMM*, 2007.

[29] T. Elhourani, A. Gopalan, and S. Ramasubramanian, "IP fast rerouting for multi-link failures," in *Proc. IEEE INFOCOM*, 2014.

[30] M. Canini, P. Kuznetsov, D. Levin, and S. Schmid, "A distributed and robust SDN control plane for transactional network updates," in *Proc. IEEE Conference on Computer Communications (INFOCOM)*, 2015, pp. 190–198.

[31] K.-T. Foerster, M. Parham, M. Chiesa, and S. Schmid, "TI-MFA: Keep calm and reroute segments fast," in *Proc. IEEE Global Internet Symposium (GI)*, 2018.

[32] P. Hande *et al.*, "Network pricing and rate allocation with content-provider participation," in *Proc. INFOCOM*, 2010.

[33] H. H. Liu, S. Kandula, R. Mahajan, M. Zhang, and D. Gelernter, "Traffic engineering with forward fault correction," in *Proc. SIGCOMM*, 2014.

[34] J. Feigenbaum et al., "BA: On the resilience of routing tables," in *Proc. PODC*, 2012.

[35] B. Stephens, A. L. Cox, and S. Rixner, "Plinko: Building provably resilient forwarding tables," in *Proc. ACM HotNets*, 2013.

[36] ——, "Scalable multi-failure fast failover via forwarding table compression," in *Proc ACM SOSR*, 2016.

[37] K.-T. Foerster, Y.-A. Pignolet, S. Schmid, and G. Tredan, "Local fast failover routing with low stretch," *ACM SIGCOMM CCR*, vol. 1, pp. 35–41, Jan. 2018.

[38] ——, "Casa: Congestion and stretch aware static fast rerouting," in *Proc. IEEE INFOCOM*, 2019.

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

[40] Y.-A. Pignolet, S. Schmid, and G. Tredan, "Load-optimal local fast rerouting for dependable networks," in *Proc. DSN*, 2017.

[41] P. K. Agarwal, A. Efrat, S. K. Ganjugunte, D. Hay, S. Sankararaman, and G. Zussman, "The resilience of WDM networks to probabilistic geographical failures," *IEEE/ACM Trans. Netw.*, vol. 21, no. 5, pp. 1525–1538, 2013.

[42] P. Fraigniaud and E. Lazard, "Methods and problems of communication in usual networks," *Discrete Applied Mathematics*, vol. 53, no. 1-3, pp. 79–133, 1994.

[43] S. M. Hedetniemi, S. T. Hedetniemi, and A. L. Liestman, "A survey of gossiping and broadcasting in communication networks," *Networks*, vol. 18, no. 4, pp. 319–349, 1988.

[44] A. Pelc, "Fault-tolerant broadcasting and gossiping in communication networks," *Networks*, vol. 28, no. 3, pp. 143–156, 1996.

[45] K.-T. Foerster, A. Kamisiński, Y. A. Pignolet, S. Schmid, and G. Trédan, "Improved fast rerouting using postprocessing," in *SRDS*. IEEE, 2019.

Klaus-Tycho Foerster is a PostDoc at the Faculty of Computer Science at the University of Vienna, Austria since 2018. He received his Diplomas in Mathematics (2007) & Computer Science (2011) from Braunschweig University of Technology, Germany, and his PhD degree (2016) from ETH Zurich, Switzerland, advised by Roger Wattenhofer. He spent autumn 2016 as a Visiting Researcher at Microsoft Research Redmond with Ratul Mahajan, joining Aalborg University, Denmark as a PostDoc with Stefan Schmid in 2017. His research interests revolve around algorithms and complexity in the areas of networking and distributed computing.

Andrzej Kamisiński is an Assistant Professor in the Department of Telecommunications at the AGH University of Science and Technology in Kraków, Poland. He received his B.Sc., M.Sc., and Ph.D. degrees from the same University in 2012, 2013, and 2017, respectively. In 2015, Andrzej Kamisiński was a Visiting Ph.D. Student at NTNU (Trondheim, Norway) where he worked with Bjarne E. Helvik and with Telenor Research on dependability of Software-Defined Networks. In summer 2018, he was a Visiting Research Fellow in the Communication Technologies group led by Stefan Schmid at the Faculty of Computer Science, University of Vienna, Austria. Between 2018 and 2020, he was a member of the Management Committee of the *Resilient Communication Services Protecting End-User Applications From Disaster-Based Failures* European COST Action, and in 2020, a Research Associate in the Networked Systems Research Laboratory at the School of Computing Science, University of Glasgow, Scotland. His primary research interests span dependability and security of computer and communication networks.

Yvonne-Anne Pignolet Yvonne-Anne Pignolet received the M.Sc. and Ph.D. degrees from ETH Zurich, Switzerland, in 2006 and 2009, respectively. After her PhD at ETH Zurich in 2009 she was a postdoc at IBM Research Zurich and Ben Gurion University, Be'er Sheva. Yvonne-Anne joined DFINITY in 2019, after 8 years at ABB Corporate Research as a Principal Scientist. Yvonne-Anne Pignolet's work is centered around networked systems, ranging from the design of algorithms for reliable and efficient distributed systems despite failures and malicious behaviour to the analysis of complex network evolution.

Stefan Schmid is a Professor at the Faculty of Computer Science at the University of Vienna, Austria. He received his MSc (2004) and PhD degrees (2008) from ETH Zurich, Switzerland. In 2009, Stefan Schmid was a postdoc at TU Munich and the University of Paderborn, between 2009 and 2015, a senior research scientist at the Telekom Innovations Laboratories (T-Labs) in Berlin, Germany, and from the end of 2015 till early 2018, an Associate Professor at Aalborg University, Denmark. His research interests revolve around fundamental and algorithmic problems arising in networked and distributed systems.

Gilles Tredan received the Ph.D. degree in computer science from University of Rennes 1 in 2009, under the supervision of A. Mostefaoui. From 2010 to 2011, he held a post-doctoral position at the FG INET Group, Berlin. He is currently a Researcher with the Laboratoire d'Architecture et d'Analyse des Systèmes, Centre national de la recherche scientifique, Toulouse, France. He likes graphs and algorithms.