Improved Fast Rerouting Using Postprocessing

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

***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.

I. 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]–[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 arc-disjoint spanning arborescences [9], highly resilient FRR configurations can be defined. 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 [10]–[12] 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 can travel along an arbitrary arborescence towards the root, being the destination. When encountering a failure, a packet is rerouted onto the next arborescence according to some arbitrary 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.

Organization. The remainder of this paper is organized as follows. Section II provides intuition on why focusing on arborescences-based network decompositions is not a limitation. Our postprocessing framework is described in Section III. We discuss and evaluate case studies in Section IV. After reviewing related work in Section V we conclude in Section VI.

Due to space constraints, and in order to improve the paper's structure, some parts are deferred to the Appendix or to the full version.

II. 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.

We will see that this 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 rerouting. 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 refer to a t-rooted spanning arborescence, where t is the destination node. A set of arborescences $\mathcal{T} = \{T_1, \dots T_k\}$ 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

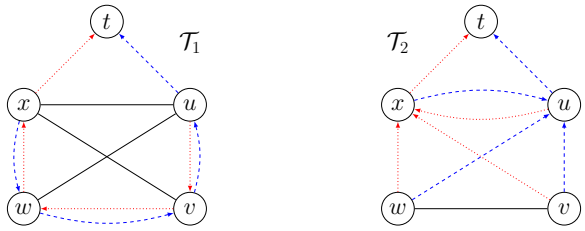


Fig. 1. Example network from [13] with two different t-rooted arc-disjoint spanning arborescence decompositions, $\mathcal{T}_1$ left and $\mathcal{T}_2$ right. In both of them one arborescence is drawn with dotted red arrows, while the second arborescence is depicted with dashed blue arrows. Note that the mean path length of the arborescences of $\mathcal{T}_1$ is 2.5, while it is less than 2 in $\mathcal{T}_2$.

$\mathcal{T}_2$. 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* [14] point of view? In the failure example of Fig. 2, an algorithm with global information could simply take a tour of length 5 from 22 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 (src, 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., router do not know where else failures have happened). Thus the power of algorithms that make deterministic decisions without knowing anything about

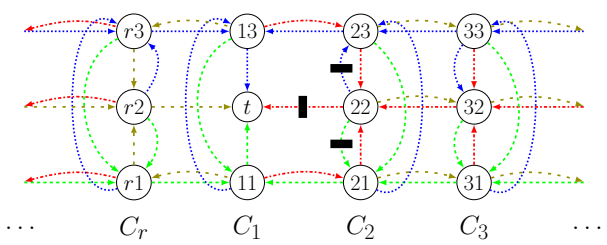


Fig. 2. Example of a $(4, r)$-clique-torus (see Definition 1), with 4 t-rooted arc-disjoint arborescences, in *blue* (dotted), *red* (dash-dotted), *green* (dashed), and *olive* (loosely dashed). Three links (striked out) incident to 22 have failed in this scenario, forcing a circular scheme to use the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

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 r 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 $3 \times r$ 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 $v_{1,j}, v_{2,j}, \ldots, v_{k-1,j}$. Next and last, for each $1 \leq i \leq k - 1$ and each $1 \leq j \leq \ell$, connect $v_{i,j}$ with $v_{(i \mod \ell)+1,j}$.*

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. The technical proof of Theorem 1 is deferred to the full version due to space constraints.

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 $\geq \ell - 6$.*

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 $(k - 1)r$ nodes, i.e., $r \in \Omega(n/(k - 1))$, 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 $\Omega(n/(k - 1))$.*

III. The Postprocessing Framework

This section presents our algorithmic framework to postprocess arborescence-based network decompositions for improved resilience and performance. In the following, we first present the general framework, before we discuss concrete use cases.

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

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, the following minimal conditions must hold when swapping $e = (u, v) \in 0 -$ arborescence with $e' = (u, v') \in E(T_j)$ to

Before swapping *After* swapping

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).

ensure that the resulting arborescences are valid [13]:[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, 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 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.

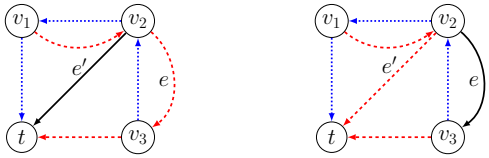
If the conditions are met, then the arborescence set after the swap is still valid. The time complexity of picking all candidate pairs is in $O(n^2\Delta)$, where Δ is the maximum node degree.

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.

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

Before swapping *After* swapping

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



Before swapping *After* swapping

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

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 $O(n)$.*

We now show that Algorithm 2 is correct:

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

Algorithm 2: Generic Post-Processing Algorithm

Input: valid arborescence decomposition
Output: improved (if possible) decomposition

```
1  improved ← ⊤;
2  while improved do
3  |  improved ← ⊥;
4  |  for all nodes v ∈ 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 ← ⊤;
9  |  |  |  else
10 |  |  |  |  undo swap of e, e'
```

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 and 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. ∎

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.

IV. 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 [15]. We then generate the corresponding arborescences, 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 they then continue on the next available arborescence, i.e., if a packet has used T_i up to the failed link, it will follow T_{i+1} next, if the corresponding outgoing link is available or try $T_{i+2}, \ldots$ otherwise.

[2] E.g., check each of the k arborescences from scratch by DFS in $\theta(kn)$.
[3] https://networkx.github.io/

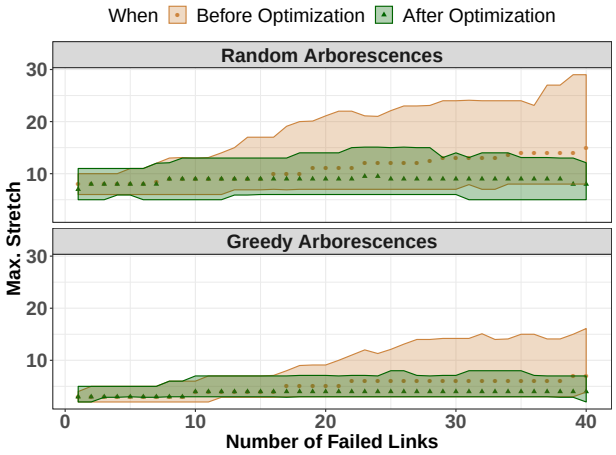


Fig. 6. Efficiency of stretch optimization when optimizing Random (*top*) or Greedy (*bottom*) arborescences, facing random failures. Each point represents the median metric value over 100 independent trials. The shaded area delimits the 10 (resp. 90) quantile values.

A. Impact of the Original Network Decomposition Algorithm

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 in the Appendix in more detail.

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*

[4] Evaluation results for the heuristic are discussed in Section IV-C.

and *Random* is not completely filled by the optimization process. In the following, we therefore focus on optimizing *Greedy* network decompositions.

B. Optimization Use Cases

a) 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 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.

b) 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.

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

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.

c) Shared Risk Link Groups: Link failures are often interdependent, if links in a network share a common fiber or other physical attributes (e.g., if they are close geographically [11]), and thus may fail simultaneously. Such dependencies are known as shared risk link groups *SRLGs* [10]–[12].

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

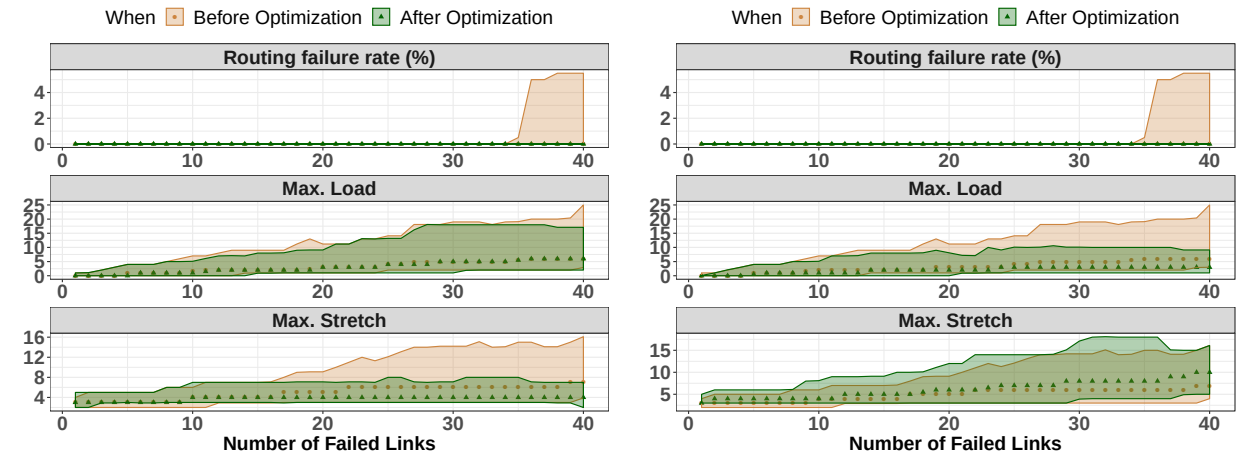


Fig. 7. Resilience, stretch and load worst case performances of greedy structures, before and after improvement, when improvement targets stretch (*left*) and load (*right*), facing random failures. Each point represents the median metric value over 100 indep. trials. Ribbon delimits the 10 (resp. 90)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

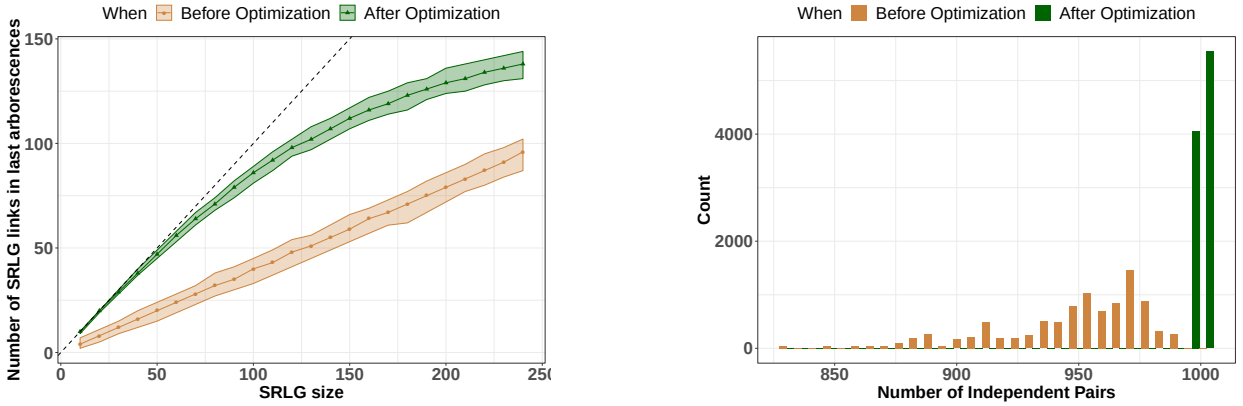


Fig. 8. *(left)* Number of SRLG links in last arborescences before and after SRLG optimization, for varying SRLG sizes. Dashed line represents the ideal case where all SRLG links end up in last arborescences. *(right)* Distribution of the number of independent pairs before and after optimization on 7200 graphs.

$k-1$ 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)
for *each link* $(u, v) \in S$ **do**
  for *each* v' *s.t.* $(u, v') \in T_{k-1}$ *or* T_k **do**
    if $(u, v') \notin S$ ***swap***$((u, v),(u, v'))$ *valid* **then**
      break /*inner for-loop*/

[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.

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 SRLGs of varying size. We then measured the capacity of our scheme to move those links towards the last two arborescences. These 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.

d) 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. A set of arborescences is α-independent for $0 \leq \alpha \leq 1$ if an α fraction of all paths from all nodes in all arborescence pairs to the destination are independent. The more such independent path pairs exist the more resilient arborescence-based rerouting can be. In particular, maximal resiliency of $k-1$ can be achieved with circular routing if the arborescence set used is 1-independent. Moreover, this property is useful to deal with node failures where all incident links to a node fail simultaneously. Note that this 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*) present 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 7200 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, 2 pairs are independent with a relatively high probability (94.9% of times on average), and that this quantity varies considerably across networks (high dispersion of values). After the optimization, pairs are independent with a high probability (99.9% on average) in nearly all the cases (lowest recorded value is 96%, 100% reached in 57.6% of the networks). This confirms the ability of our optimization to produce independent (and therefore more resilient) paths.

C. 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 [13] also proposed a heuristic called *Bonsai* that attempts to generate arborescences of small depth, with no guarantees if a valid output may be produced (see Appendix for a more detailed description). Notwithstanding, if successful, the question arises if said arborescences can further benefit by our approach in this paper.

Maybe surprisingly, the rate of improvement is quite similar to the effects described in Section IV-B for the *Greedy* approach. In other words, even though the heuristics in [13] were already optimized ahead of time, our postprocessing framework still yielded similar significant improvements for *Bonsai*. The plots are deferred to the Appendix to improve the paper's structure.

V. 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., [16]–[19]. 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 [20]–[23], which require dynamic router tables and long convergence times, quadratic in the number of nodes [24]. 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 [25], [26] or packet-duplication [27]. 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 [28]. 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. [29], Chiesa et al. [7], [8], Elhourani et al. [25] and Stephens et al. [30], [31] 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 [32] 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 [13], [33]–[35]. Except [13], 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). [13] 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 [33] 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 [36], or requirements of communication pairs. Indeed, while shared risk link groups (and their characterization) has been studied intensively in the literature, e.g., see [11]. 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 [37]–[39].

VI. 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.

References

[1] S. Peter, U. Javed, Q. Zhang, D. Woos, T. Anderson, and A. Krishnamurthy, "One tunnel is (often) enough," in *ACM SIGCOMM Computer Communication Review*, 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] 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.

[11] 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.

[12] 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.

[13] 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.

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

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

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

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

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

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

[20] 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.

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

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

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

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

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

[26] M. Canini *et al.*, "A Distributed and Robust SDN Control Plane for Transactional Network Updates," in *Proc. INFOCOM*, 2015.

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

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

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

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

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

[32] 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.

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

[34] 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.

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

[36] 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.

[37] 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.

[38] 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.

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

[40] 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.

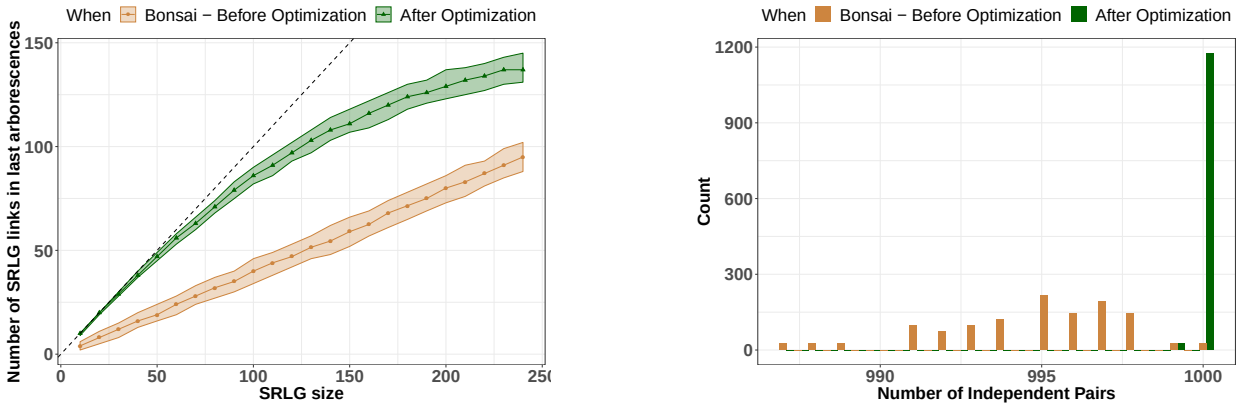


Fig. 9. Effect on *Bonsai* [13]: *(left)* Number of SRLG links in last arborescences before/after SRLG optimization, for varying SRLG sizes. Dashed line represents the ideal case where all SRLG links end up in last arborescences. *(right)* Distribution of the number of independent pairs before and after optimization.

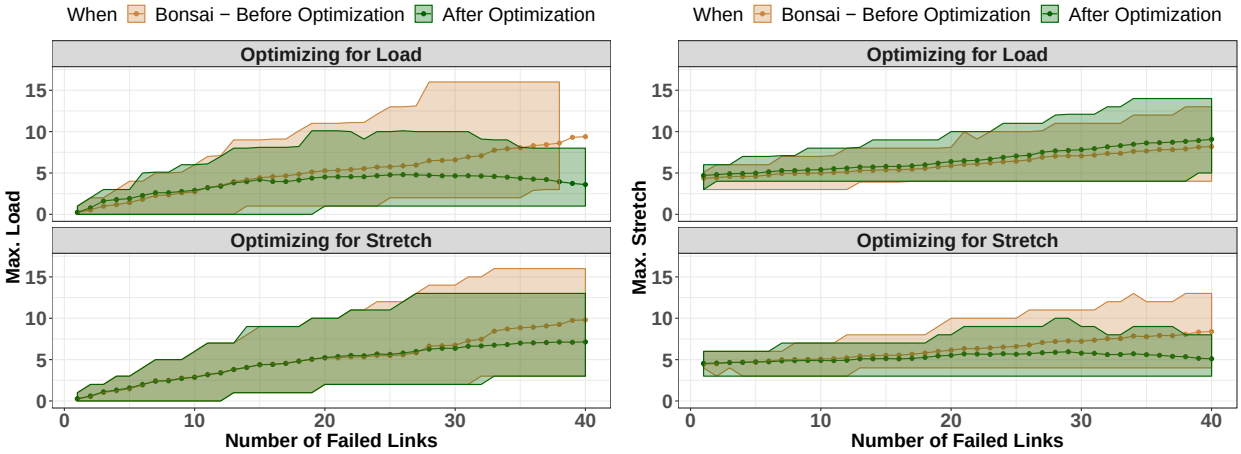


Fig. 10. *Load* (left) and *stretch* (right) performances of *Bonsai* arborescences [13], before and after improvement, when improvement targets load (top) or stretch (bottom), facing random failures. Each point represents the median metric value over 100 indep. trials. Ribbon delimits the 10 (resp. 90) quantile values.

APPENDIX

A. Decomposition Generation

Random decomposition. When building the i^{th} arborescence T_i, the following method ensures that the graph with all the arcs belonging to the arborescences $T_1, \ldots, T_i$ removed, is still $k-i$ connected. We start at the root and insert a random unused arc (not belonging to any $T_j,\ j<i$) towards the root into T_i. Now a recursive procedure is used to add new arcs (u,v) to T_i which extend T_i (i.e. $v \in T_i$) while maintaining the arborescence structure (i.e., $v \notin T_i$). For each edge to be added we test whether there are still $k-i$ arc-disjoint paths from u to the root on the unused arc (excluding (u,v)). If yes the arc is added to T_i and we proceed recursively. Otherwise the next arc is tested. This algorithm always succeeds to construct k arc-disjoint arborescences.

Greedy decomposition. The following greedy approach ensures that T_i has the lowest depth of all possible arborescences on the arcs that have not been used yet. As in the random decomposition, we start at the root and insert one of the unused arcs (not belonging to any $T_j,\ j<i$) towards the root into T_i. All candidate arcs (u,v) are tested until a suitable one is found, ordered by the depth the arborescences would exhibit with (u,v). Analogously to the above, we test whether there are still $k-i$ arc-disjoint paths from u to the root on the unused arc. This approach is used for the experimental evaluation in [40]. This algorithm also always succeeds to construct k arc-disjoint arborescences. The depth of the first arborescence is the smallest of all arborescences and the depth of the other arborescences increases monotonically. For networks with very few failures, this construction combined with circular routing starting with the first arborescence is a good practical choice.

Heuristic round-robin decomposition. *Bonsai* [13] 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 [13], it has the downside that it might not produce a valid decomposition. To this end, the authors in [13] 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.

The postprocessing results for the improved heuristic round-robin decompositions are shown in the Figures 9 and 10.