Bonsai: Efficient Fast Failover Routing Using Small Arborescences

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

Bonsai: Efficient Fast Failover Routing Using Small Arborescences

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

***Abstract*—To provide high availability despite link failures, many modern communication networks feature fast failover mechanisms in the data plane, which operates orders of magnitude faster than the control plane. While the configuration of highly resilient data planes using the shortest possible back-up routes is known to be a difficult combinatorial problem, over the last years, much progress has been made in the design of algorithms which provably guarantee connectivity even under many concurrent link failures. However, while these algorithms provide *connectivity*, the resulting routes after failures can be very long, which in turn can harm performance.**

In this paper, we propose, analyze, and evaluate methods for fast failover algorithms which account for the *quality* of the routes after failures, in addition to connectivity. In particular, we revisit the existing approach to cover the to-be-protected network with arc-disjoint spanning arborescences to define alternative routes to the destination, aiming to keep the stretch imposed by these trees low (hence the name of our method: *Bonsai*). We show that the underlying problem is NP-hard on general topologies and present lower bound results that are tight for various topologies, for any class of fast failover algorithms. We also present heuristics for general networks and demonstrate their performance benefits in extensive simulations. Finally, we show that failover algorithms using low-stretch arborescences, as a side effect, can provide connectivity under more general failure models than usually considered in the literature.

I. Introduction

Communication networks have become mission critical and reliability is one of the main concerns of network operators today [1]. Ensuring a high network availability however is often non-trivial, especially under frequent and concurrent link failures, which are becoming more likely with the increasing scale of communication networks including datacenters [2], backbones [3], [4] or enterprise [5] networks, but also due to virtualization and shared risk link groups [6].

Fast rerouting in the data plane is an important mechanism to meet availability guarantees: as reaction times to failures in the data plane are several orders of magnitude shorter than in the control plane [7], many communication networks today support statically precomputed conditional failover paths [8] (e.g., using *IP Fast Reroute* [9], [10], *MPLS Fast Reroute* [11], or *OpenFlow fast-failover groups* [12]), along which traffic can be rerouted in case failures are encountered. Interestingly, allocating such conditional failover paths introduces a challenging combinatorial problem: as the forwarding rules on the switches or routers need to be installed *beforehand* and *without knowledge* of the actual failures which may occur, the forwarding decisions must be robust to *all possible* additional failures which may occur downstream.

The underlying algorithmic problem is related to distributed computing problems (due to the switch or router's local view on failures) [13]. Over the last years, much progress has been made (e.g., [4], [14], [15], [16], [17]) toward the design of polynomial-time algorithms to precompute failover rules which provide *optimal connectivity* [18]: failover routes are guaranteed as long as the underlying network remains *physically* connected.

However, while *connectivity* is important, connectivity alone is not sufficient to meet performance requirements, also the *quality* of the resulting failover routes matters [8]. In particular, long failover routes may introduce additional delays: if the length of the old and the new path differ, this can temporarily lead to packet reorderings and an overestimation of the congestion of the network, harming TCP throughput [19]. More importantly, long routes also increase the likelihood of congestion, as bandwidth needs to be allocated on a per-hop basis and longer flows may interfere with more other flows. Thus we focus in this paper on minimizing the maximal stretch which can be introduced by rerouting.

Contributions. We investigate deterministic fast failover mechanisms which not only provide provable connectivity under multiple link failures, but also account for the quality in terms of the resulting route lengths, namely the *stretch*: the actual length of the failover route, minus the shortest originally possible distance. At the same time, previous fast failover works focused mostly on connectivity alone and they often required dynamic routing tables or modifications of packet headers. Reducing the route length and stretch, compared to earlier work, has a positive impact on latency and jitter.

We consider the state-of-the-art approach to resilient routing, which protects a network by covering it with *arc-disjoint spanning arborescences* [18] (arborescences are used to define alternative routes to the destination in the case of failed links). We motivate our approach with the fact that focusing only on the connectivity can lead to very long failover paths, where we also give some appropriate lower bounds. We thus investigate algorithms which compute an arc-disjoint spanning arborescence packing, where each arborescence provides low stretch, hence optimizing the maximum stretch: the idea is that this results in shorter failover routes, without sacrificing connectivity. Since we keep failover arborescences ("trees") small, we call our approach *Bonsai*.

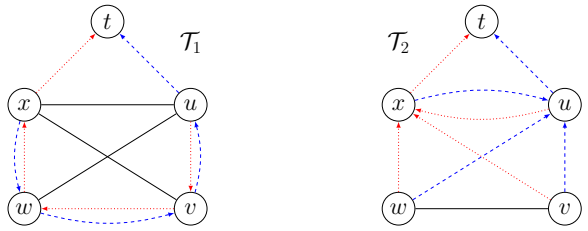

Fig. 1. Example of a five node graph with two different choices of failover routes $\mathcal{T}_1, \mathcal{T}_2$, depicted in *blue* (dashed) and *red* (dotted). In both $\mathcal{T}_1, \mathcal{T}_2$, routing can switch to a second arc-disjoint arborescence if a failure is encountered. $\mathcal{T}_1$ is inspired by routing along a Hamiltonian cycle, either clock- or counter-clockwise. The arborescences in $\mathcal{T}_2$ optimize for short paths.

As we prove the problem to be NP-hard for general network topologies, we present a fast and flexible heuristic that takes stretch optimization criteria into account. We report on extensive experiments of this heuristic, for different random and real-world networks. Our results show that we can significantly improve the stretch for many scenarios, in particular when the network size increases, in comparison to currently used methods. Moreover, our techniques also provide good performance gains in other aspects, such as the computation time or stretch in relation to network connectivity. Finally, we demonstrate the benefits of low-stretch failover algorithms to support more general failure models than usually studied in the literature: when the failures are appropriately clustered, low-stretch arborescences greatly increase network survivability.

Example. To illustrate our problem and show the need for route length-aware arborescences, we consider the network with five nodes in Fig. 1. In this simple example, we consider flows that need to be routed to the destination t, in the top of the figure. In the absence of link failures, the flows could be routed along e.g. the dotted *red* paths. In the case of incident link failures, any node y will apply conditional rules which have been pre-installed. These rules can only be conditioned on the availability of links incident to the node y and the in-port, i.e., rerouting decisions are purely local. We refer to Sec. II (and related work [13], [14], [16], [20]) for a detailed model.

We are interested in strategies to pre-compute such conditional failover rules which ensure that one is still able to route to t, even in the presence of multiple failures. In the example in Fig. 1, node x has a conditional failover rule which reroutes traffic if the dotted *red* arc (x, t) is not available. In the case of the $\mathcal{T}_1$ tree, traffic might be rerouted to node w via the dashed *blue* arc (x, w). If w would not match on the incoming port, then the packet might be forwarded again to x, resulting in a forwarding loop.

Prior work provided important insights in how failover rules should be defined in order to avoid such loops, even under multiple link failures, for example, relying on Hamiltonian cycles [15, §B.6]. However, in the worst case, these schemes can result in very long paths: if x resorts to the alternative counter-clockwise route indicated by dashed *blue* arcs in $\mathcal{T}_1$, the stretch is on the order of the number of nodes, which is especially harmful when larger examples are considered. On the other hand, the routing as described in $\mathcal{T}_2$ on the right side just induces a stretch of one extra hop. Thus, we aim to ensure that failover paths "preserve locality", without sacrificing resiliency. As such, we are interested in algorithms to compute static failover rules which result in paths like the one indicated in $\mathcal{T}_2$.

A large body of work on routing schemes resilient to single failures as well as routing schemes resilient to multiple failures with dynamic failover tables (where link reversal approaches [21] can be used) exists, as we will discuss in more details in the related work section. However, much less is known about the design of resilient static forwarding tables, especially for scenarios where packet-header rewriting or packet-duplication is impossible or undesired (the former consumes header space and the latter introduces additional loads). The most closely related works to ours are by Chiesa et al. [14], [15], [18], [22], Stephens et al. [16], [17], and Pignolet et al. [13], [23], who developed robust failover schemes using static forwarding tables. These approaches provide very high resilience. However, they do not guarantee any non-trivial deterministic bounds on the resulting path lengths. The work of Foerster et al. [8] provides bounds on the resulting path lengths, however just for specialized regular topologies such as the 2d-torus, grids, and data center topologies based on complete bipartite graphs—these constructions are specialized and cannot be extended to more general graph classes.

This paper aims to fill this gap, by considering the study of deterministic local algorithms for short failover paths on general network topologies. We build upon the concepts of *arc-disjoint spanning arborescences*, which were also exploited in prior work [8], [14], [15], [18], [22], [24], [25] and are reminiscent of work on homotopic routing problems [26].

Organization. We present our formal model in Section II, followed by Section III, where we introduce the concepts of arc-disjoint spanning arborescences and derive lower bounds on the possible stretch, along with proving optimal stretch computation to be NP-hard. Section IV investigates the construction of stretch-aware arborescences, where we first introduce prior methods and then present our own round-robin approach, which can efficiently swap links for optimization purposes. In the next Section V, we also provide positive evaluation results which showcase that our new approach performs well in practice, in comparison to prior work. We furthermore show that low-stretch arborescences also have a positive influence on the resiliency, by investigating theoretical guarantees for distributed failure clusters in Section VI. We then discuss related work in Section VII and lastly conclude in Section VIII.

II. Network and Routing Model

We model the network as a graph $G = (V, E)$, connecting n nodes in set V (switches, routers, hosts) using bidirected links E (i.e., a full-duplex symmetric digraph). That is, if there is a link (u, v), then there is also always a link (v, u) in the opposite direction. When focusing on the directed nature of a link, we often use the term *arc* to emphasize this.

We assume that forwarding rules can match the destination field from the packet header as well as the in-port (the port

from which a packet arrives at v)[1], and depending on this match, define the outgoing port to which a packet is forwarded at v. In other words, the focus of this paper is on *oblivious* routing algorithms which do not rely on any dynamic state at nodes (e.g., no counters), nor in packets: we do not allow packet tagging. While marking packets is known to improve the robustness of routing [14], [27], it may not be possible in practice to add additional header fields or to reuse existing fields, as they are needed by other protocols.

The failover mechanisms needs to be *statically preconfigured*: at the time the failover rules are installed, the *set of link failures* F is not known yet. The mechanism must be configured such that for any possible set of *local* link failures, a failover action is taken which provides connectivity and stretch guarantees, *independently* of the additional failures that may be encountered down-stream.

More precisely, we say that a failover algorithm A has a *resiliency* of $|F|$, if for any source-destination pair $s, t \in V$ it holds: the route taken by packets from s according to algorithm A leads to t despite any $|F|$ link failures.

Note that statically preconfigured failover allows a network to react seamlessly to local failures, whether they are transient or permanent. In case of permanent link problems, the reconstruction of routes can be triggered in addition, if necessary. The focus of this work is on the properties of such local failover algorithms and not on the reconstruction of permanent failures.

III. Rooted Spanning Arborescences

In order to obtain efficient local fast failover algorithms, we leverage a known approach for resilient routing [14], [18], [22], [25], based on *rooted arc-disjoing spanning arborescences*[2]: routing is performed along arborescences, where upon hitting a failure, the packet switches to another arborescence and follows it—without modifying the packet header.

A. Arborescence Preliminaries and Prior Work

We now formally define the concept of rooted arc-disjoint spanning arborescences and how to use them for failover routing as discussed in prior work, e.g., by switching between them in a circular order [14], [15], [18], [22].

Arborescence properties. 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.

Generating arc-disjoint arborescences. A packing (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 $i \neq j$. It is known that k arc-disjoint arborescences exist in any k-connected graph [28] and that they can be computed efficiently [29], with a runtime of $O(|E| + nk^3 \log^2 n)$. In a simpler (and slower) version, one creates a spanning arborescence s.t. the remaining graph remains $k-1$ connected, and repeating this process $k-2$, $k-3$, $\dots$, until k arc-disjoint arborescences are obtained [30]. A conceptually different approach is proposed by Chiesa et al. [15], using link-disjoint Hamiltonian cycles: given $k/2$ such cycles, they can be turned into k arc-disjoint arborescences, two for each cycle, in opposite directions. However, k-connectedness does not imply $k/2$ disjoint Hamiltonian cycles.

Resilient routing on arc-disjoint arborescences. Elhourani et al. [24] and Chiesa et al. [15] showed how decompositions of G into $\mathcal{T}$ can be used to define failover routes for packets destined to t. These packets are routed according to an arborescence T_i by forwarding them along the unique directed path of T_i towards the root t. If a link $(u, v) \in E(T_i)$ that should be used for the next hop is not available, the affected packets are forwarded along a different arborescence T_j at u, i.e., the packet switches from T_i to T_j at u.

When arborescences are arc-disjoint, a failed arc only disconnects one arborescence, i.e., in a k-connected graph, $k-1$ failures leave at least one arborescence intact. On the other hand, k failures can physically disconnect the destination from parts of the network. Elhourani et al. [25] obtain a resilience of $k-1$ by indexing the arborescences from 1 to k, with all packets starting in arborescence 1, switching to the arborescence with the next higher index after a failure.

Chiesa et al. [14], [15], [18], [22] propose to generalize this concept by defining a circular order on the arborescences. As such, packets may start in any arborescence, and may also be forwarded after hitting k failures, though without any theoretical guarantees in the latter case.[3] Chiesa et al. also provided other failover strategies, some of them probabilistic. Instead of defining a fixed circular order on the arborescences, the next arborescence may be chosen uniformly at random, hopefully breaking out of forwarding loops. As an extension, motivated by bidirectional link failures, packets can bounce on the failed link e with some fixed probability, picking the other arborescence that was also impacted by failing link e.

Arborescence quality. A defining quality of routing on an arborescence is its stretch, which we now define formally. Let $\ell_{\text{opt}}(v, t)$ denote the minimum distance packets have to travel from v to t without failures (shortest path routing) and let $\ell_T(v, t)$ denote the route length along arborescence T. The *stretch of a node* v in T is defined as the difference between routing optimally and on T, i.e., $\ell_T(v, t) - \ell_{\text{opt}}(v, t)$, whereas the *stretch of an arborescence* T is defined as the maximum stretch over all $v \in V$. Similarly, the *stretch of an arborescence packing* $\mathcal{T}$ is defined as the maximum stretch over all $T \in \mathcal{T}$.

[1]The in-port is crucial for resiliency. E.g., consider a network with a dead-end, e.g., a node v which can only be reached via a link from u after the failures. As packets are forced to return back to u along link (v, u), i.e., the same link from which they arrived, matching the in-port is needed to facilitate a different routing decision at v, avoiding a loop.

[2]Also denoted as branchings or directed rooted trees in the literature.

[3]We show how to extend the theoretical guarantees beyond k failures in Section VI for failure distributions where the failures are clustered [31].

When it is clear from the context, we will just use the term stretch, without specifying v, T, or $\mathcal{T}$.

However, prior work does not provide theoretical guarantees on the stretch when using arborescences, except on how often one might switch the arborescences [14], [15], [18], or for some specialized regular topologies [8]. In extreme cases, the stretch might even be worst possible. For example, using arc-disjoint arborescences generated from disjoint Hamiltonian cycles [15], can induce a stretch of up to $\Omega(n)$, as indicated in Fig. 1. Still, sometimes very low stretch arborescences are not possible, as we investigate in the next section.

B. Lower Bounds on Arborescence Stretch

We start with the observation that once we have more than one arborescence in $\mathcal{T}$, its stretch is at least 1. Consider any neighbor v of the destination t: its distance to t is 1, but the corresponding arc may only be used in a single arborescence.

Matching bounds on complete graphs. Already this simple lower bound allows the optimal stretch to be obtained for complete graphs: we can directly decompose a complete graph into $n-1$ arc-disjoint arborescences of depth 2, where each arborescence uses a different arc (v,t) to the destination t as their first link followed by all arcs from nodes in $V \setminus \{v,t\}$ to node v. Hence, each node has a stretch of at most 1 in every arborescence.

Arc-disjoint paths and lower bounds. We can broaden our initial idea to a more general setting. As all the arborescences need to be arc-disjoint, not all can take the shortest path to the root, but rather need to take k different paths, which gives first stretch bounds: some arborescences must take longer detours.

Observation 1: Let $\mathcal{T}$ be any packing of k arc-disjoint arborescences rooted in $t \in V$. Consider the set of all k arc-disjoint $v-t$-paths, denote $\ell^k(v,t)$ as the minimum length of the kth-shortest (i.e., longest) path from this set. The stretch of $\mathcal{T}$ is at least $\max_{v \in V} \ell^k(v,t) - \ell_{\text{opt}}(v,t)$.

Girth bounds. The network girth is the length of the graph's shortest undirected cycle. Again, we pick any neighbor u of the root t, and note that the girth can be used to derive a lower bound on $\max_{v \in V} \ell^k(v,t) - \ell_{\text{opt}}(v,t)$: if an alternate arc-disjoint route is to be taken, instead of the direct link (u,v), this alternate path has at least the length of the graph's girth minus 1 for (u,v). For example, in the case of complete graphs, the girth is 3, hence the alternate path has at least a length of 2. As the stretch is computed by subtracting the shortest path, we obtain a lower bound of girth minus 2.

Corollary 1: The network's girth minus 2 is a lower bound for the stretch of an arborescence packing $\mathcal{T}$, for $|\mathcal{T}| > 1$.

Optimality for regular topologies. For regular graphs, this bound can suffice for optimality, as seen above on complete graphs. Further examples are e.g. torus graphs, grids, hypercubes, and topologies that can be decomposed into trees of connected complete bipartite graphs [8], all examples where the girth based bound is tight. For general topologies, the situation is more difficult, as discussed in the next section.

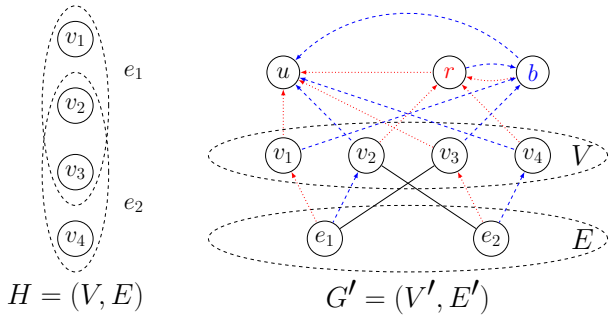

Fig. 2. G' graph construction for a hypergraph $H = (V = \{v_1, v_2, v_3, v_4\}, E = \{e_1, e_2\})$, $e_1 = \{v_1, v_2, v_3\}, e_2 = \{v_2, v_3, v_4\}$, using the ideas from [32]. The nodes r_1, r_2 and b_1, b_2 are merged, respectively, to make the figure less cluttered. In this example, we provide two arborescences of depth 2, in *red* (dotted) and *blue* (dashed), corresponding to a 2-coloring of the hypergraph with v_1, v_3 in *red* and v_2, v_4 in *blue*.

C. Arborescence Decomposition Complexity

Even though computing a maximum amount of arc-disjoint arborescences can be done efficiently, optimizing them regarding stretch is intractable. Hence, due to the NP-completeness result stated next, we will focus on heuristics in the following Section IV and compare them to each other.

Theorem 1: Let $G = (V, E)$ be a k-connected symmetric digraph with a root $r \in V$. Computing k r-rooted arc-disjoint spanning arborescences, s.t. the maximum stretch is minimized, is NP-complete for all $k \geq 2, k \in \mathbb{N}$.

From depth to stretch. Alon, Bermond, and Fraigniaud [32] showed that minimizing the maximum depth for more than one disjoint arborescence is NP-hard, via reduction from hypergraph[4] colorability. However, their result does not suffice to show hardness for the stretch: in fact, in their construction the optimal stretch is easy to compute. Notwithstanding, we adapt their ideas to show NP-hardness for optimal stretch.

Proof of Theorem 1: As the stretch problem is clearly in NP, it remains to show NP-hardness. To this end, we utilize and extend [32, Theorem 2] and subsequent comments.

The authors in [32] perform a reduction from the NP-complete problem of hypergraph colorability [33]: given a hypergraph $H = (V, E)$, can its node set be 2-colored (e.g., red and blue), s.t. every hyperedge $e \in E$ is incident to nodes of both colors? They construct the following graph $G' = (V', E')$, showing that finding two undirected link-disjoint u-spanning arborescences of depth at most 2 is NP-hard: V' consists of V, each hyperedge $e \in E$ is represented as a node, and the five nodes u, r_1, r_2 (red), and b_1, b_2 (blue). The undirected link set is constructed in four steps: 1) the nodes $v \in V$ are connected to those representing $e \in E$ if v is incident to e in H, 2) all nodes in V are connected to all nodes u, r_1, r_2, b_1, b_2 and 3) u is connected to r_1, r_2, b_1, b_2. Lastly, 4) r_1 and r_2 are connected to b_1, b_2 as well. We provide an example in Fig. 2.

We briefly note at this point that their and our following proof construction directly translate to bidirected graphs, by replacing every undirected link with two directed opposite arcs:

[4]Hypergraphs are a generalization of graphs, where the edges are replaced by hyperedges, which in turn can join any number of nodes, not just two.

essentially, using the "back"-direction of a link is not useful at all. For ease of readability, we will use undirected links as well, just denoting them as links in the remainder of the proof.

We analyze the depth and stretch properties of the nodes in V', when they are contained in two arborescences of depth 2. Herein, the depth of a node in an arborescence is denoted w.r.t. the root u, and the stretch w.r.t. the shortest path to u:

- for nodes in E to have a depth of 2 in an arborescence, the stretch must be 0, i.e., only two hops to u;
- subsequently, the nodes in V, connected to the nodes in E, must have a depth of 1 and a stretch of 0;
- as the only option for nodes in V to have a depth of 1 is to be connected (via the arborescence links) directly to the root u, which only provides $|V|$ of the $2|V|$ necessary links, at least half the nodes in V have a depth of 2 and stretch of 1 in some arborescence — no higher depth or stretch is needed, as the detour via r_1, r_2, b_1, b_2 provides ample connectivity;
- the remaining nodes r_1, r_2, b_1, b_2 can be connected to the root with a depth of at most 2 and a stretch of at most 1.

If no arborescences with depth 2 are possible, then some node has at least depth 3. Note that a depth of 3 is always possible. To this end, we first connect all nodes in V via r_1 (red arborescence) and b_1 (blue arborescence) resulting in a depth of 2 and stretch of 1 so far. Next, we connect the nodes in E to those nodes in V, resulting in a depth of 3 and stretch of 1 for both arborescences. However, this construction, while increasing the depth, maintains the stretch of 1: the maximum stretch in V is still 1, but nodes in E have a stretch of at most 1 as well now. If we could enforce that some node were to have a stretch of 2 with depth 3, then we would have shown NP-hardness for the stretch of 2 arborescences. We will next show how to achieve this feat.

We take the graph G', clone it $|E|$ times, and in each of those $|E|$ clones G'_i, $1 \leq i \leq |E|$, we merge the node e_i with u_i, denoting the (polynomially created) graph by $G'' = (V'', E'')$. We obtain new node sets V'_i (distance to u: 3) and E'_i (distance to u: 4). If the original graph G' had two u-rooted link-disjoint arborescences of depth 2, then the new graph G'' has two such arborescences of depth 4 as well. Firstly, the nodes in V'_i (and the four extra nodes $r_{1,i}, \ldots$) have a depth of $3, 4$ and a stretch of $0, 1$. Secondly, the nodes in E''_i have a depth of 4 and as thus a stretch of 0. Observe that in this case of depth 4, all nodes have a stretch of 0 or 1 in the two arborescences.

We now assume that the original graph G' does not allow for two arborescences of depth 2, i.e., following the previous arguments, at least one node in E has a depth of 3 with stretch of 1, w.l.o.g. $e_j \in E$. However, then at least one node in E'_j has a depth of 5 in some arborescence, with a stretch of 2. Hence, minimizing the maximum stretch is NP-hard, as it is NP-hard to decide if two rooted disjoint arborescences with stretch 1 exist. The above NP-hardness construction can be directly extended to any $k \in \mathbb{N}$ number of arborescences for k-connected graphs, applying the ideas from [32, p.5]. ■

IV. Bonsai: How to Build Better Arborescences for Arbitrary Networks

A crucial question studied in this paper is how to avoid the black-box modeling of arborescences, which does not provide any stretch properties. We therefore analyze the complexity of general arborescence decompositions for arbitrary networks, with the goal of obtaining minimum stretch.

Since the problem was shown to be NP-complete in Section III-C, we now describe polynomial-time heuristics to decompose arbitrary graphs into "Bonsai" arborescences efficiently while striving to keep their stretch low. The simplest way to decompose a k-connected graph into k arc-disjoint arborescences constructs one arborescence after each other, as in the two approaches described next.

Random decomposition. When building the i^{th} arborescence T_i, the following method ensures that the graph with all the arcs belonging to the trees $T_1, \ldots, T_i$ are 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 and serves as our baseline in the comparison with construction that take the depth of the resulting arborescences into account.

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 [14]. 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 very good practical choice. However, a bad combination of failures might lead to long detours on $T_i, i > 1$.

Round-robin approach. Instead of building one arborescence after the other, the round-robin approach constructs all of them in parallel. After the j^{th} edge has been added to the first arborescence, chosen from all the unused edges, the second arborescence obtains its j^{th} edge and so on. To make the procedure simpler, we may omit the connectivity test described

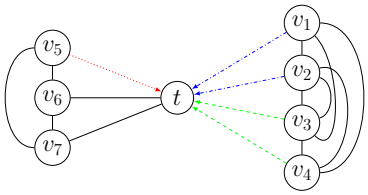

Fig. 3. Example: 3-connected graph. The Round-Robin approach performed two rounds for the first *blue* (dash-dotted) and second *green* (dashed) arborescences, and one round for the third *red* (dotted) arborescence. Even though the graph remains $k-1=2$-connected when removing any single arborescence, the *red* (dotted) arborescence cannot reach the nodes v_1 to v_4.

for the greedy approach[5]. By increasing the depth in each of these decisions only if strictly necessary, this leads to a much more balanced arborescence packing with respect to the length of the detours they entail. Unfortunately, however, this procedure does not always succeed for general graphs. In some cases, there is no unused arc left that can be added to arborescence T_i even though it is i's turn, we reached a dead-end. An example of such a situation for three arborescences is provided in Fig. 3.

Refined connectivity test. In some cases, such dead-ends can be avoided by the following connectivity test. Before adding an unused arc (u,v) to T_i, we count the number of arc-disjoint paths from u to the root in the graph $H=(V',E')$, which represents the unused arcs in case of i failures, i.e., $V'=V$ and $E'=\{(v,w)|(v,w)\notin \cup_{j\leq i}T_i\}$. This represents the number of potentially usable arc-disjoint paths left for the arborescences after i failures. If this number is not at least $k-i$, the corresponding edge is not added to T_i.

Swapping arcs when growing arborescences. Even with this refined connectivity test, we might end up in a dead-end. To get out of it, we can try if exchanging arcs already chosen by an arborescence might mitigate the problem. For example in Fig. 3, when we swap the blue arc (v_1,t) to the unused arc (v_1,v_2), the 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.

To this end, the following conditions must hold.

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

These conditions are sufficient and necessary to obtain valid arborescences by assigning e to $E(T_j)$ and e' to $E(T_i)$. When testing if a set of arcs forms an arborescence, there must be exactly one directed path from every involved node to the root. (4) ensures that no arc leaving the root can be added to an arborescence, while (3,5,6) ensure that valid paths to the root exist in both involved arborescences afterwards, i.e., v and v' have the necessary outgoing links in the appropriate arborescence and no cycles can be created by the swap. We can address the case where e and e' do not originate from the same node analogously by adjusting the conditions above.

When adding the l^{th} edge, each arborescence $T_j, j<i$ contains l nodes and all arborescences $T_j, j\geq i$ contain $l-1$ nodes. As a consequence, the above procedure can be implemented with time complexity $O(l^2\delta)$ on average, for a graph of maximum degree δ and an implementation of set operations of $O(1)$ on average. Thus, testing for swapping possibilities takes more and more time as the arborescences grow. Observe, however, that this approach is much more efficient than the naive approach, which checks for all pairs of edges if the involved graphs T_i, T_j are still valid arborescences after the swap ($O(ln^2\delta^2)$).

For many graphs, the extension of the round-robin approach with the refined connectivity test and swapping arcs finds an arc-disjoint arborescence packing with very low depth. However, there are still cases when the approach reaches a dead-end.

V. Decomposition Comparison

In this section we compare the properties of the approaches described above in Section IV. We construct arc-disjoint arborescence packings using (1) a *random* decomposition, (2) a *greedy* decomposition, (3) our *round-robin (RR)* approach, and (4) our RR aproach using *swapping (RR-swap)*. We then compare their (a) success rate, (b) stretch, and (c) running time. In our evaluation, we use random regular graphs of varying size and connectivity and the well-connected cores of various ASes. We refer to [34] for further details concerning reproducibility.

A. Experiments on Random Regular Graphs

Our first set of experiments was on random graphs, which were used for the experimental evaluation of the greedy method in [14], [22]. More precisely, we generate random k-regular graphs, sampled in an asymptotically uniform way [35], which are almost surely k-connected [36, p. 195ff.]. In all generated graphs, the degree and connectivity matched. We pick seven different graph sizes (up to 1000 nodes) and six different connectivities (from 5 to 30) to simulate a wide spectrum of parameters, where each combination is generated 100 times, picking a random node as the root. We then generate arborescence packings $\mathcal{T}$ of size k with the random, greedy, RR, and RR-swap method.

Construction success on random regular graphs. As displayed in Fig. 4, all three of the random, greedy, and swapping approaches succeed in 100% of the experiments, whereas the success rate of the round-robin approach greatly drops, failing completely beyond ≈ 200 nodes.

Such behavior is to be expected from the random and greedy approaches, as both utilize sequential constructions, maintaining connectivity for the following arborescences. The swapping numbers are more interesting — while the algorithm can fail in theory, it did not do so over the course of our experiments.

[5]For two arborescences, it is easy to show that this connectivity test suffices to always finish the construction on $\geq$ 2-connected graphs: the remaining subgraph always contains a rooted spanning tree, i.e., the arborescences can continue to add arcs until all nodes are in them.

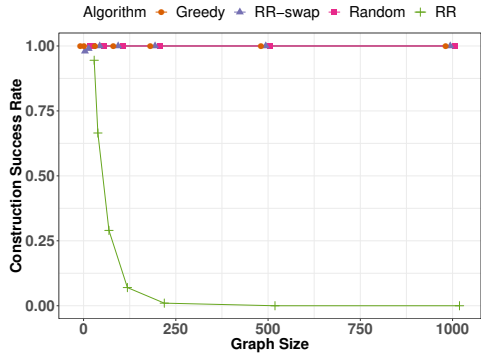

Fig. 4. Success rate for the arborescence packings $\mathcal{T}$ of the different algorithms on random k-regular graphs, aiming for k disjoint arborescences in each trial. Each data point represents the average success rate over 100 attempts.

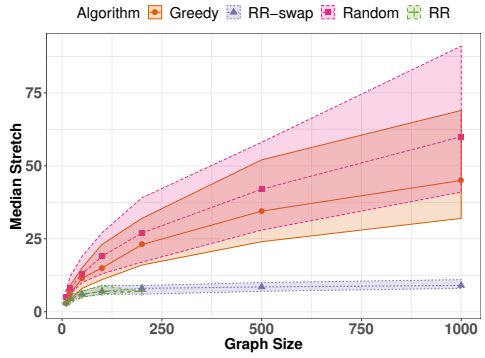

Fig. 5. Median stretch for the arborescence packings $\mathcal{T}$ (100 each) of the different algorithms in random regular graphs, plotted over the number of nodes of the graphs. The shaded areas display the 5th to 95th percentile.

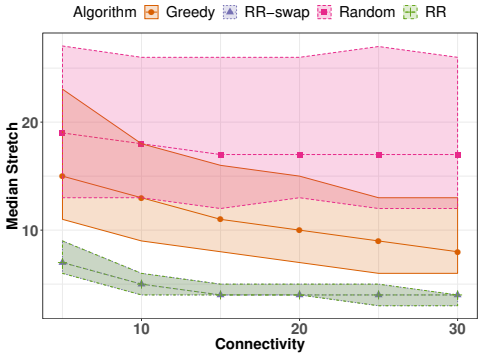

Fig. 6. Median stretch for the arborescence packings $\mathcal{T}$ (100 each) of the different algorithms in random regular graphs, plotted over the connectivity of the graphs. The shaded areas display the 5th to 95th percentile.

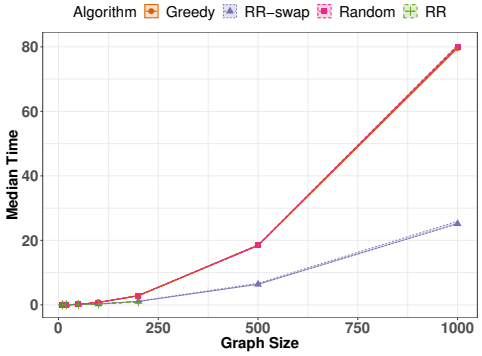

Fig. 7. Median computation time (seconds) for the arborescence packings $\mathcal{T}$ (100 each) of the different algorithms in random regular graphs, plotted over the graph size. The (small) shaded areas display the 5th to 95th percentile.

We thus believe that on random regular graphs, our swapping method nearly always finds alternatives, with deadlocks being extremely rare.

Stretch on random regular graphs. The stretch performance on random regular graphs is shown in the Figs. 5 (plotted over different graph sizes) and 6 (for various connectivities). In both cases, RR and RR-swap perform quite similarly, in Fig. 6 their results overlap. However, RR without swapping fails early to finish its constructions. Both algorithms vastly outperform the random and greedy approaches in the median stretch, as well as the 5th to 95th percentile, of the arborescences $\mathcal{T}$, in particular when considering larger graph sizes where, e.g., not even the percentiles for RR-swap and greedy overlap. While the performance of RR-swap stays similar for larger random regular graphs, it degrades for greedy and random. As the connectivity increases, the median performance of all four algorithms improves slightly, but random and greedy still always stay far behind the RR versions. The situation is similar for the 5th to 95th percentile, with the exception of the random method: here, the performance does not change much with varying connectivity.

Computation time for random regular graphs. The computation time (in seconds) of the arborescence packings $\mathcal{T}$ is shown in the Figs. 7 (different sizes) and 8 (different connectivities). We used an Intel i5-4570 @3.2 GHz with 8GB ram for the experiments. In general, the median and 5th to 95th percentile values barely differ for each individual algorithm. For RR and RR-swap, the dominating factor is picking the next arc, which increases linearly with the number of arcs. Their computation time overlap, though RR fails to find solutions beyond 200 nodes. In general, RR-swap only needed to utilize very few swaps to resolve deadlocks, meaning that the computation time does not change much due to swapping in most cases. For greedy and random, the dominating factor was the ongoing connectivity checks, which result in much slower runtime than RR-swap, the plotted values grow quadratically. As such, their time values are essentially identical as well, with random being slightly faster for larger connectivities.

B. Experiments on Well-Connected Cores of ASes

Our second set of experiments was on the well-connected cores of various ASes, taken from the commonly used *Rocketfuel* data set [37]. We trim the AS graphs s.t. only the well-connected cores remain, as follows. We first *contract* nodes of bidirected degree 2 into a single bidirected link. Next, we remove nodes that have a degree less than $4/5/6/7$, contracting the graph again later. This process is repeated until no more nodes can be contracted or removed. If the trimming resulted in less than 20 nodes, we omit them. For each such topology (ranging from 20 to roughly 700 nodes), we pick 20 different

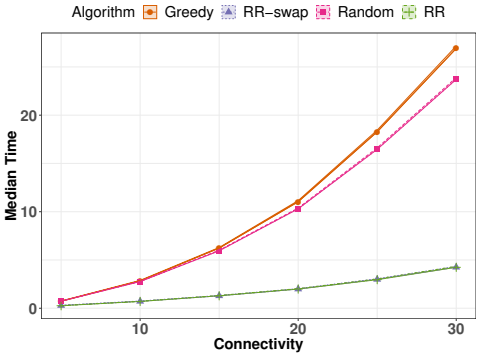

Fig. 8. Median computation time (seconds) for the arborescence packings $\mathcal{T}$ (100 each) of the different algorithms in random regular graphs, plotted over the connectivity. The (small) shaded areas display the 5th to 95th percentile.

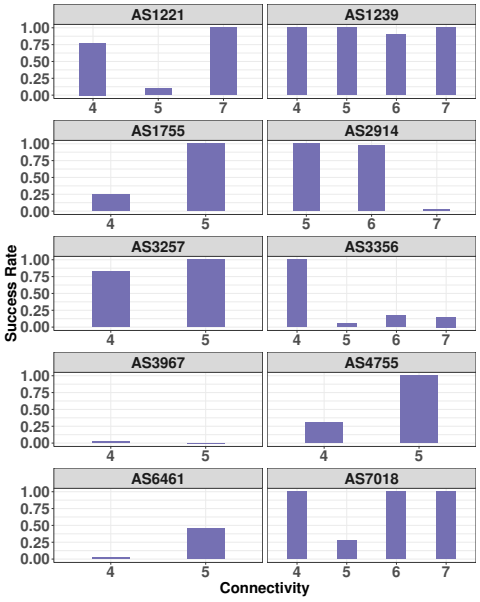

Fig. 9. Success rates for the arborescence packings $\mathcal{T}$ for RR-swap, aiming for 4 to 7 disjoint arborescences Each item represents the average success rate over 20 attempts, picking a random root each time. Greedy always succeeds.

nodes uniformly at random, selecting them as a root for the arborescence packings. We generate arborescence packings $\mathcal{T}$ of size $k \in \{4, 5, 6, 7\}$ with the greedy and RR-swap methods.

Construction success on AS graphs. Greedy always finishes, meaning that our trimming operation generated $4/5/6/7$-connected graphs. Different to the random regular graphs, RR-swap struggles to complete some arborescence constructions, as seen in Fig. 9: in total, RR-swap succeeds 61% of the time. Upon further further investigation, we saw that RR-swap created partial arborescences that could not be fixed locally, i.e., which would require a non-trivial amount of restructuring.

Stretch on AS graphs. The stretch of the arborescence packings $\mathcal{T}$ generated by greedy and RR-swap is shown in Fig. 10, where we depict the median stretch for the different connectivities, along with the 5th to 95th percentiles. As can be seen, RR-swap performs clearly better than greedy, where in many cases not even the 5th to 95th percentile ranges overlap. On the other hand, the differences are not as prominent as for

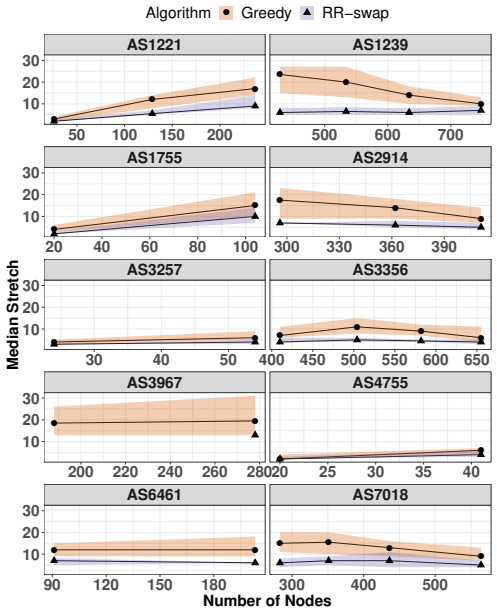

Fig. 10. Median stretch (and 5th to 95 percentiles) for the arborescence packings $\mathcal{T}$ (up to 20 each) of the different algorithms on the well-connected cores of different ASes, plotted over the size of the graphs.

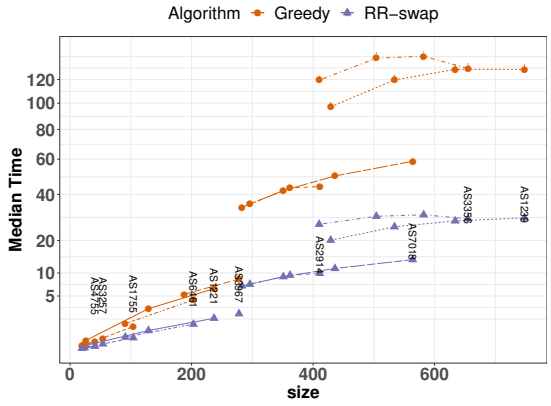

Fig. 11. Median computation time for the arborescence packings $\mathcal{T}$ (up to 20 each) of the different algorithms on the well-connected cores of different ASes, plotted over the size of the graphs. The (very small) bars display the 5th to 95th percentile. For each AS and algorithm, we have up to four data points, representing the different sizes for $|\mathcal{T}| \in \{4, 5, 6, 7\}$.

random regular graphs, and in some cases, the RR-swap results are missing. Another interesting observation is that higher connectivity not always implies better stretch. For example in AS1221, the stretch goes up with lower connectivity whereas in AS1239, lower connectivity implies lower stretch.

Computation time for AS graphs. The computation time (in seconds) of the arborescence packings $\mathcal{T}$ is shown in Fig. 11. We used the machine employed for the random regular graphs. We note that for better visibility, the y-axis (the computation time) is depicted on a logarithmic scale. Similar as for random regular graphs, the RR-swap method greatly outperforms the greedy method: the reason is (again) that the runtime of greedy is dominated by the connectivity computations, whereas RR-swap needs to perform relatively few swaps, as swaps are heavily optimized as well, recall the end of Section IV.

C. Concluding Remarks

While the round robin approach can still get stuck despite swapping, it reaches much better stretch than the random and greedy approach. Consequently, it makes sense to combine them in practice: if round robin with swapping returns k complete arborescences, they are taken (this is also the fastest algorithm without the connectivity condition); otherwise greedy is run, which in turn outperforms the random approach.

VI. Guarantees beyond $k-1$ Failures

In the first few sections, we implicitly restricted ourselves to a resiliency of at most $k-1$ failures in in k-connected graphs. For a general adversarial failure model this is a natural choice, as no more than k disjoint arborescences can be used if the graph is not $(k+1)$-connected, along with the observation that k failures might disconnect the graph.

In this section, we will theoretically analyze how good stretch properties can allow us to cope with situations that go beyond $k-1$ failures, by assuming that these failures are distributed over the network, in the sense that they are clustered [31], respectively only few appear locally.[6] This will allow us to leverage the locality properties of our failover schemes, as, in some sense, past failures stop to matter after a certain point.

To this end, we will next first define a parametrized spatial failure distribution in Section VI-A, followed by its application to circular arborescence routing schemes in Section VI-B.

A. Failure Ball Distribution

We begin by introducing so-called b-*balls*, which will contain at most b failed links, also defining their pairwise distance.

Definition 1: Let $G=(V,E)$ be a graph with a set of failed links $F \subset E$. We call a set of at most $b \leq |F|$ links from F a b-*ball*. The distance $\mathrm{dist}(B,B')$ between two b-balls B, B' is defined as $\min_{e\in B, e'\in B'} \mathrm{dist}(e,e')$.

We now use these b-balls to cover all failures of a graph, where the existence of such a cover requires that the balls can be chosen in such a way that they maintain a minimum specified distance to each other. The idea is to parametrize the spatial distribution of failures over the graph, so only a pre-defined amount of failures may be locally clustered.

Definition 2: Let $G=(V,E)$ be a graph with a set of failed links $F \subset E$. We say that G has a (b,c)-*failure cover*, if there exists a set $\mathcal{B}$ of b-balls, with $\cup_{B\in\mathcal{B}} B = F$, such that for any two $b-$balls $B, B' \in \mathcal{B}$, $B \neq B'$, the following two conditions hold: 1) $B \cap B' = \emptyset$, 2) $\mathrm{dist}(B,B') \geq c$.

In other words, the b-balls in $\mathcal{B}$ cover all failures, are pairwise disjoint, and have minimum distance of c from each other. An example is shown in Figure 12. Furthermore, if $c>0$, then Condition 2) also includes Condition 1), i.e., $B \cap B' = \emptyset$.

Note that so far, we only specified the term resiliency for arbitrary link failures. In addition, we can also define resiliency of an algorithm to a specific set of failures on a given graph.

[6]For example in data centers, "*Corruption has weak spatial locality*" [38].

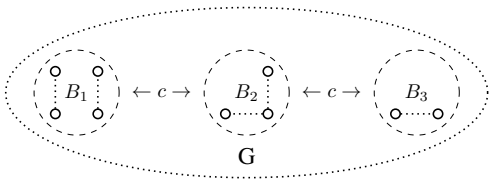

Fig. 12. Illustration of a (b,c)-failure cover with $b=2$ and $\{B_1,B_2,B_3\} = \mathcal{B}$. Only the failed links (dotted) and their incident nodes are depicted. All remaining links in the graph G are free from failures.

Definition 3: Let A be a routing scheme with default and (conditional) failover flow rules, as specified in Section II. Let $G=(V,E)$ be a graph with a set of failed links $F \subset E$. If A successfully routes packets on G under link failure set F, then we say A is F-resilient for G.

We now apply these definitions to circular arborescence routing schemes to be resilient to arbitrarily many failures under certain conditions.

B. Usage in Circular Arborescence Routing

To cope with spatially distributed failures, we need routing schemes that do not deviate too much from the shortest path after encountering a failed link. An illustration for the contrary is the Hamiltonian cycle technique described in [15, §B.6]. Consider that packets follow Hamiltonian cycles. When encountering the first failure on the current cycle, they start traversing it in the opposite direction. This approach can induce an additive stretch of $\Omega(n)$ after a single failure, as a single failure close to the destination can lead to touring nearly the entire graph.

Instead, we desire circular arborescence techniques that impose a small additive stretch of $x \in \mathbb{N}$, i.e., for each arborescence it holds that routing from a node v to t along an arborescence leads to the traversal of at most $\mathrm{dist}(v,t)+x$ hops, if no failures occur. We now show that circular arborescence routing with small stretch is resilient to arbitrarily many failures, as long as we can partition the failures into balls of large enough distance. Conceptually, our ideas take some inspiration from geometric ad-hoc/local routing paradigms [39], [40]

Theorem 2: Let A be a r-resilient circular arc-disjoint arborescence routing scheme for a graph family $\mathcal{G}$, where each of its $r' > r$ arborescences imposes an additive stretch of at most x. Let $G \in \mathcal{G}$ with a link failure set F. If there exists an $(r, xr+x+r+1)$-failure cover for G w.r.t. F, then A is F-resilient for G.

The proof idea can be roughly summarized as follows: each hit failure can derail a packet only by a certain amount from its course to the destination, namely at most the additive stretch of the next arborescence picked. Thus, if there exists a failure ball coverage s.t. the failure balls have a distance slightly greater than the total detour possible by a single ball, the packet will not return to a previous failure.

At first glance, it might seem as if a distance between balls that is only slightly larger than the detour caused by a single arborescence would be a sufficient distance between two failure balls. However, note that the Definition 1 of failure balls did

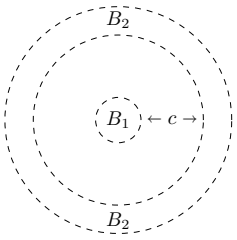

Fig. 13. Visual representation of a (b, c)-failure cover where a b-ball B_2 surrounds another b-ball B_1. A valid routing, going from top to bottom for example, might hit errors in B_2, then route towards B_1 where additional failures are encountered, and then hit failures in B_2 again. If the routing scheme and the (b, c)-failure cover are improperly specified, this could result in an infinite loop where packets bounce between failures inside of B_1, B_2.

not include any convexity structure, only an upper limit on the number of contained failures. Hence, one failure ball B_1 could surround a link failure from another failure ball B_2, inducing a loop where the packet bounces between failure sequences in B_1 and B_2, see Figure 13 for an example. Similarly, we would like to point out that Theorem 2 holds as long as *any* such specified failure cover exists.

Proof of Theorem 2: We prove in the following that A is F-resilient for G. To this end, we show that A cannot loop indefinitely on G if an $(r, xr + x + r + 1)$-failure cover exists for F. Observe that if G is finite, the only way to prevent a packet from looping is allowing it to reach its destination. We now consider for contradictory purposes the last loop that a packet ends up in, that is, the shortest closed walk W that is eventually repeated indefinitely. As A is deterministic, such a walk W must exist. If the packet never hits a failed link in W, triggering a conditional failover flow rule, then the packet must reach the destination: as no failure is hit, the packet never changes its arborescence, but that implies that the arborescence has a loop, a contradiction. Thus, the remaining case to be analyzed concerns a packet hitting failures, and changing arborescences accordingly. Should the walk W be confined to only hitting failures from a single r-ball, then we arrive at a contradiction again: the ball has at most r failures, i.e., by starting appropriately at one of these failures, we must either reach the destination or hit a failure from another r-ball. Recall that we have $r' > r$ arc-disjoint arborescences.

Hence, if a packet loops, it must hit failures from different r-balls, where W may hit at most r failures, possibly once from each of the up to $2r$ incident nodes, in a single ball before encountering a failure in another ball. For each such r-ball traversal, we can define the (node, in-port[7])-pair where the last error is hit before a change to another ball occurs. Note that such a pair is unique in W, as A is deterministic. So far, we may have many of these pairs for a single ball, as W could traverse a single ball multiple times, especially since we did not require the r-balls to form some sort of local cluster. From this pair set, pick w_1 where the contained node, w.l.o.g. $v_1 \in B_1 \in \mathcal{B}$[8] is of minimum distance to the destination, and denote the next such pair when following W by w_2, where its contained node is $v_2 \in B_1 \in \mathcal{B}$ (not necessarily of minimum distance to t).

After w_1, let the next pair along W which contains a node $v_2' \in B_2$ be denoted by w_2'. As B_2 has at most ℓ failures, where each may be hit once from each incident node, we obtain $\mathrm{dist}(v_2, t) \leq \mathrm{dist}(v_2', t) + 2xr$, due to the fact that every arborescence change can increase the distance to the destination by at most x. We can improve this bound. While each failure hit can detour the packet by x, summing up these hops is only important for the stretch (see the later Corollary 2), not for the distance from v_2'. When a failed link $(u_1, v_1) = e_1$ is hit from u_1, and we later hit it again from v_1, we increased the distance at most by one. Hence, we achieve the following improved bound: $\mathrm{dist}(v_2, t) \leq \mathrm{dist}(v_2', t) + xr + r$. A will route (uninterrupted by hitting failures) along an arborescence between w_1 and w_2', with $\mathrm{dist}(v_1, v_2') \geq xr + x + r + 1$. As each arborescence has a stretch of at most x it holds that: $\mathrm{dist}(v_2', t) \leq \mathrm{dist}(v_1, t) - (xr + x + r + 1) + x = \mathrm{dist}(v_1, t) - xr - r - 1$.

Combining the inequalities yields: $\mathrm{dist}(v_2, t) \leq \mathrm{dist}(v_1, t) - xr - r - 1 + xr + r = \mathrm{dist}(v_1, t) - 1$.

However, we chose w_1 s.t. $\mathrm{dist}(v_1, t) \leq \mathrm{dist}(v_2, t)$, a contradiction to earlier inequalities. Hence, we have shown that A cannot loop as w_1 is the last (node,in-port)-pair visited where a failure is hit, i.e., we obtained F-resiliency for G. ■

As such, circular arborescence routing can tolerate arbitrarily many failures, assuming that the failure distribution is spatially appropriate. The total additive stretch remains bounded by the product of the total number of possible failure hits and the additive stretch of the arborescences:

Corollary 2: Let A be a F-resilient circular arc-disjoint arborescence routing scheme for G, where each arborescence imposes an additive stretch of at most x. The additive stretch of A on G under F is at most $2x|F|$.

Proof: As circular arborescence routing changes the arborescence after each failure hit, A incurs only an additive stretch of at most x each time. Note that the combination of node and in-port results in a unique next routing decision, for a given set of failures. Assume that we change arborescences more than $2|F|$ times. Then, due to the pigeonhole principle, we hit some link failure twice with the same (node,in-port) combination, which results in a forwarding loop. However, we showed in Theorem 2 that A is F-resilient for G, a contradiction which completes the proof of Corollary 2. ■

We would like to note that the circular property is crucial to the results of this section, as it properly defines how the routing scheme reacts after r failures were hit. Observe that by just following the definition of resilience, no proper routing has to be defined after r failures were encountered. As such, we can envision to extend the resiliency beyond r failures for other routing schemes, by appropriately specifying the routing behavior after r failures.

[7]When starting at a node emitting a packet, we assume the in-port to belong to the initially chosen arborescence.

[8]We slightly abuse notation here, by also including incident nodes.

VII. Related Work

Failures are common. Many network outages have been attributed to link failures and are well-documented in the literature, e.g., [41], [42], [43], [44], and there is also much literature on the empirical characterization of failures, e.g., in backbone networks [3], [45], [46], [47].

***Fast* failover.** A key challenge in designing resilient networks is that ensuring connectivity via the control plane can be slow [7], [48]. Hence, many networks (additionally) provide data plane mechanisms to preserve connectivity [9], [11], [12] and allowing the definition of planned backup paths. The preferred mechanism depends on the network: datacenters often rely on the ECMP data plane algorithm (see also [49] for a recent study of datacenter network reliability) while many WAN networks rely on MPLS Fast Reroute [7]; supporting a faster failover was one of the main reasons for Google's move to SDN [50].

Single vs multiple failures. Much existing fault-tolerant and robust routing solutions revolve around single failures [51], [52], [53], [54], which however may be insufficient to provide high availability in more complex scenarios, e.g., shared risk link groups [55], attacks [56], node failures [2], [5], [9], [24]. A well-known example in the literature is the Internet outage in Pakistan in 2011 which was due to a failure in both a link and its backup [41].

Dynamic tables. A classic approach to deal with *multiple* failures, which has been studied intensively in the literature, is to use Gafni and Bertsekas' link reversal algorithm [21], or one of its extensions [57], [58], [59]. However, link reversal algorithms require dynamic tables which are not always supported; furthermore, the approach can introduce non-trivial delays [60]. An interesting recent contribution in this area is by Liu et al. [7] who proposed a link reversal algorithm suited *for the data plane*: DCC provides provable connectivity and can tolerate arbitrary delays and losses.

Alternative approaches: duplication, header modification, scaling. Other approaches to deal with multiple link failures require packet header rewriting [25], [61] (e.g., packets relay failure information [62], [63]) or packet-duplication [64], which however consumes header space or introduces additional loads, respectively. Yet another approach is to pre-compute multiple flow paths s.t. even in the event of multiple failures, the ingress switches can rescale the traffic load efficiently without additional computational overhead [65].

What-if analysis tools. Not only the design of robust algorithms has been studied intensively in the literature, but also the question how to efficiently *verify* the robustness of a *given* algorithm or configuration. Recently, polynomial-time verification algorithms have been presented for MPLS networks, based on automata theory [4], [66], and supporting arbitrarily large header sizes as they may result from header rewriting during failover (see above).

Ideal resilience. Our paper is motivated by robust routing algorithms which provide static resiliency in the data plane without requiring header rewriting, initially proposed in [48] and [20] and subsequently studied intensively by Stephens et al. [16], [17], Chiesa et al. [14], [15], [18], [22], Elhourani et al. [25], and Pignolet et al. [8], [13], among others. Feigenbaum et al. [48] raised the question of whether it is possible to achieve an "ideal (static) resilience": Is it always possible to define failover rules such that connectivity is preserved as long as the network is physically connected? The authors already proved that this is not possible. Simultaneously, Borokhovich and Schmid showed that this is not even possible in an initially completely connected network which is still highly connected after the failures [20]. Chiesa et al. [15] then raised the interesting (and so far only partially answered) question whether it is at least always possible to preserve connectivity in a k-(edge-)connected network if there are at most $k-1$ link failures. Pignolet et al. [13], [23] observed and exploited a connection of fast rerouting problems to combinatorial block designs, in order to minimize congestion on failover routes.

Approaches going beyond connectivity. Most of the existing literature on fast rerouting under multiple link failures focuses on *connectivity*, which alone however is insufficient to meet the availability and performance guarantees of emerging communication networks. The study of the *quality* of failover paths has been stated as an open problem in the literature [7], and except for some notable first studies on the load on failover paths in very dense networks [13], the space is largely unexplored. In particular, not only the arborescence based solutions in [14], [15], [18], [22], without further optimizations, may result in long failover paths, but also many other algorithms, e.g., based on graph search which rely on packet header rewriting [27]. To the best of our knowledge, the work by Foerster et al. [8] is the only one that generates arborescences with small stretch, however only for the special cases of torus, hypercube, and grid graphs, as well as trees of complete bipartite graphs. Unlike in our work, no approaches for (more) general graphs are provided nor practical evaluations or complexity results.

Arborescences with good depth and other application domains. We note that the problem of finding arborescences also arises in other domains, e.g., in broadcast applications [67]. There also already exists work on arborescences of improved depth, either by reducing the number of arborescences [68] or by considering restricted topologies, such as hypercubes [69] or tori [70]. We refer to the surveys in [67], [71], [72] for further references.

VIII. Conclusion

In this paper, we studied local fast failover algorithms which guarantee not only resiliency but also provide routes with bounded stretch. In addition to analytical results regarding the complexity of the problem and the impact of failure distributions on the resilience and stretch, we proposed and evaluated arc-selection methods to create suitable arborescences. Our solutions can lead to significantly shorter failover routes than with state of the art methods, without modifying the packet headers and without incurring rule convergence delays.

Other costs or capacities could be included in the balanced arc-selection algorithm with a vector instead of scalar based approach. To keep the evaluation simple, we focus on scalar values, yet the algorithm can be extended to further constraints and multi-objective scenarios.

Our work opens several interesting avenues for future research. For example, this paper has focused on a hop distance metric, which is natural to capture resource allocations; however, it would be interesting to generalize our results to arbitrary link weights (e.g., representing latencies). Another open question in terms of algorithms concerns provably low-stretch failover algorithms for further special graph classes. Along the same lines, the study of rerouting techniques with header re-writing constitutes another interesting direction for future research on low stretch failover algorithms.

In order to guarantee reproducibility and facilitate other researchers to build upon our algorithms, we will make source code and simulation results publicly available at [34].

Acknowledgements. We would like to thank the anonymous reviewers and our shepherd Elias P. Duarte Jr.

REFERENCES

[1] Y. Wang, H. Wang, A. Mahimkar, R. Alimi, Y. Zhang, L. Qiu, and Y. R. Yang, "R3: Resilient routing reconfiguration," *ACM SIGCOMM CCR*, vol. 40, no. 4, pp. 291–302, Aug. 2010.

[2] 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.

[3] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C.-N. Chuah, and C. Diot, "Characterization of failures in an ip backbone," in *Proc. IEEE INFOCOM*, 2004.

[4] S. Schmid and J. Srba, "Polynomial-time what-if analysis for prefix-manipulating mpls networks," in *Proc. IEE INFOCOM*, 2018.

[5] A. Shaikh, C. Isett, A. Greenberg, M. Roughan, and J. Gottlieb, "A case study of ospf behavior in a large enterprise network," in *Proc. ACM IMW*, 2002.

[6] D. Xu, Y. Xiong, C. Qiao, and G. Li, "Failure protection in layered networks with shared risk link groups," *IEEE network*, vol. 18, no. 3, pp. 26–41, 2004.

[7] J. Liu, A. Panda, A. Singla, B. Godfrey, M. Schapira, and S. Shenker, "Ensuring connectivity via data plane mechanisms," in *Proc. 10th USENIX NSDI*, 2013, pp. 113–126.

[8] 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.

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

[10] A. Kamisiński, "Evolution of ip fast-reroute strategies," in *Proc. International Workshop on Resilient Networks Design and Modeling (RNDM)*, 2018.

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

[12] Switch Specification 1.3.1, "OpenFlow," in *https://www.opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.3.1.pdf*, 2012, (last accessed in April 2019).

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

[14] 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.

[15] M. Chiesa, A. Gurtov, A. Madry, S. Mitrovic, I. Nikolaevkiy, A. Panda, M. Schapira, and S. Shenker, "Exploring the limits of static failover routing (v4)," *arXiv:1409.0034 [cs.NI]*, 2016.

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

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

[18] 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.

[19] E. Blanton and M. Allman, "On making tcp more robust to packet reordering," *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 1, pp. 20–30, 2002.

[20] 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.

[21] 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.

[22] 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.

[23] K.-T. Foerster, Y.-A. Pignolet, S. Schmid, and G. Tredan, "CASA: congestion and stretch aware static fast rerouting," in *Proc. IEEE INFOCOM*, 2019.

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

[25] ——, "IP fast rerouting for multi-link failures," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 3014–3025, 2016.

[26] M. Kaufmann and K. Mehlhorn, "A linear-time algorithm for the homotopic routing problem in grid graphs," *SIAM J. on Computing*, vol. 23, no. 2, pp. 227–246, 1994.

[27] M. Borokhovich, L. Schiff, and S. Schmid, "Provable data plane connectivity with local fast failover: Introducing openflow graph algorithms," in *Proc. ACM SIGCOMM HotSDN*, 2014.

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

[29] A. Bhalgat, R. Hariharan, T. Kavitha, and D. Panigrahi, "Fast edge splitting and edmonds' arborescence construction for unweighted graphs," in *Proc. SODA*, 2008.

[30] H. N. Gabow, "Efficient splitting off algorithms for graphs," in *Proc. ACM STOC*, 1994.

[31] J. Tapolcai, L. Rónyai, B. Vass, and L. Gyimothi, "List of shared risk link groups representing regional failures with limited size," in *Proc. IEEE INFOCOM*, 2017.

[32] J.-C. Bermond and P. Fraigniaud, "Broadcasting and np-completeness," *Graph Theory Notes of New York*, no. XXII, pp. 8–14, 1992.

[33] L. Lovász, "Covering and coloring of hypergraphs," in *Proc. 4th Southeastern Conf. on Combinatorics, Graph Theory, and Computing, Utilitas Mathematica*, 1973.

[34] https://gitlab.cs.univie.ac.at/ct-papers/2019-dsn.

[35] NetworkX 2.2, "Random regular graph generation," https://networkx.github.io/documentation/stable/reference/generated/networkx.generators.random_graphs.random_regular_graph.html#networkx.generators.random_graphs.random_regular_graph, 2018, (last accessed in April 2019).

[36] B. Bollobás, *Random Graphs*, ser. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 2001, no. 73.

[37] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson, "Measuring ISP topologies with rocketfuel," *IEEE/ACM Trans. Netw.*, vol. 12, no. 1, pp. 2–16, 2004.

[38] D. Zhuo, M. Ghobadi, R. Mahajan, K.-T. Foerster, A. Krishnamurthy, and T. E. Anderson, "Understanding and mitigating packet corruption in data center networks," in *Proc. ACM SIGCOMM*, 2017.

[39] H. Abelson and A. A. DiSessa, *Turtle geometry*. MIT press, 1986.

[40] F. Kuhn, R. Wattenhofer, Y. Zhang, and A. Zollinger, "Geometric ad-hoc routing: of theory and practice," in *Proc. ACM PODC*, 2003.

[41] D. Madory, "Renesys blog: Large outage in pakistan," https://dyn.com/blog/large-outage-in-pakistan/, (last accessed in April 2019).

[42] R. Singel, "Fiber optic cable cuts isolate millions from internet, future cuts likely," https://www.wired.com/2008/01/fiber-optic-cab/, 2008, (last accessed in April 2019).

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

[44] C. Wilson, "'dual' fiber cut causes sprint outage," https://web.archive.org/web/20080906210432/http://telephonyonline.com/access/news/Sprint_service_outage_011006/, 2006, (last accessed in April 2019).

[45] G. Iannaccone, C.-n. Chuah, R. Mortier, S. Bhattacharyya, and C. Diot, "Analysis of link failures in an ip backbone," in *Proc. ACM SIGCOMM Workshop on Internet Measurment*, 2002.

[46] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C. N. Chuah, Y. Ganjali, and C. Diot, "Characterization of failures in an operational ip backbone network," *IEEE/ACM Transactions on Networking*, vol. 16, no. 4, pp. 749–762, Aug 2008.

[47] A. J. González and B. E. Helvik, "Analysis of failures characteristics in the uninett ip backbone network," in *2011 IEEE Workshops of International Conference on Advanced Information Networking and Applications*, 2011.

[48] J. Feigenbaum, B. Godfrey, A. Panda, M. Schapira, S. Shenker, and A. Singla, "Brief announcement: On the resilience of routing tables," in *Proc. ACM PODC*, 2012.

[49] J. Meza, T. Xu, K. Veeraraghavan, and O. Mutlu, "A large scale study of data center network reliability," in *Proc. ACM IMC*, 2018.

[50] A. Vahdat, D. Clark, and J. Rexford, "A purpose-built global network: Google's move to sdn," *Commun. ACM*, vol. 59, no. 3, pp. 46–54, Feb. 2016.

[51] G. Enyedi, G. Rétvári, and T. Cinkler, "A novel loop-free ip fast reroute algorithm," in *Proc. EUNICE*, 2007.

[52] S. Nelakuditi, S. Lee, Y. Yu, Z.-L. Zhang, and C.-N. Chuah, "Fast local rerouting for handling transient link failures," *IEEE/ACM Transactions on Networking (ToN)*, vol. 15, no. 2, pp. 359–372, 2007.

[53] J. Wang and S. Nelakuditi, "Ip fast reroute with failure inferencing," in *Proc. ACM SIGCOMM Workshop on Internet Network Management*, 2007.

[54] B. Zhang, J. Wu, and J. Bi, "Rpfp: Ip fast reroute with providing complete protection and without using tunnels," in *Proc. IEEE IWQoS*, 2013.

[55] 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.

[56] 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.

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

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

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

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

[61] M. Canini, P. Kuznetsov, D. Levin, and S. Schmid, "A Distributed and Robust SDN Control Plane for Transactional Network Updates," in *Proc. IEEE INFOCOM*, 2015.

[62] 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.

[63] 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.

[64] P. Hande, M. Chiang, R. Calderbank, and S. Rangan, "Network pricing and rate allocation with content-provider participation," in *Proc. IEEE INFOCOM*, 2010.

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

[66] J. S. Jensen, T. B. Krogh, J. S. Madsen, S. Schmid, J. Srba, and M. T. Thorgersen, "P-rex: Fast verification of mpls networks with multiple link failures," in *Proc. ACM CoNEXT*, 2018.

[67] 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.

[68] T. Hasunuma, "On edge-disjoint spanning trees with small depths," *Inf. Process. Lett.*, vol. 75, no. 1-2, pp. 71–74, 2000.

[69] S. L. Johnsson and C. Ho, "Optimum broadcasting and personalized communication in hypercubes," *IEEE Trans. Computers*, vol. 38, no. 9, pp. 1249–1268, 1989.

[70] J. G. Peters, C. Rapine, and D. Trystram, "Small depth arc-disjoint spanning trees in two-dimensional torodial meshes," Technical Report SFU-CMPT-TR-2002-10, School of Computing Science, Simon Fraser University, Tech. Rep., 2002.

[71] 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.

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