On the Resiliency of Static Forwarding Tables

Marco Chiesa*, Ilya Nikolaevskiy†, Slobodan Mitrović‡, Andrei Gurtov§¶,
Aleksander Mądry||, Michael Schapira**, Scott Shenker††‡‡
*Université Catholique de Louvain, Belgium †Department of Computer Science, Aalto University, Finland ‡École Polytechnique Fédérale de Lausanne, Switzerland §Linköping University, Sweden ¶ITMO University, Russia ||Massachusetts Institute of Technology, US **Hebrew University of Jerusalem, Israel ††University of California, Berkeley, US ‡‡International Computer Science Institute, US

***Abstract*—Fast Reroute (FRR) and other forms of immediate failover have long been used to recover from certain classes of failures without invoking the network control plane. While the set of such techniques is growing, the level of resiliency to failures that this approach can provide is not adequately understood. In this paper, we embarked upon a systematic algorithmic study of the resiliency of forwarding tables in a variety of models (i.e., deterministic/probabilistic routing, with packet-header-rewriting, with packet-duplication). Our results show that the resiliency of a routing scheme depends on the "connectivity" k of a network, i.e., the minimum number of link deletions that partition a network. We complement our theoretical result with extensive simulations. We show that resiliency to 4 simultaneous link failures, with limited path stretch, can be achieved without any packet modification/duplication or randomization. Furthermore, our routing schemes provide resiliency against $k-1$ failures, with limited path stretch, by storing $\log(k)$ bits in the packet header, with limited packet duplication, or with randomized forwarding technique.**

I. Introduction

Routing on the Internet (both within an organizational network and between such networks) typically involves computing a set of *destination-based* routing tables (i.e., tables that map the destination IP address of a packet to an outgoing link). Whenever a link or node fails, routing tables are recomputed by invoking the routing protocol to run again (or having it run periodically, independent of failures). This produces well-formed routing tables, but results in relatively long outages after failures as the protocol is recomputing routes.

As critical applications began to rely on the Internet, such outages became unacceptable. As a result, "fast failover" techniques have been employed to facilitate immediate recovery from failures.[1] The most well-known of these is Fast Reroute in MPLS where, upon a link failure, packets are sent along a precomputed alternate path without waiting for the global recomputation of routes [1]. This, and other similar forms of fast failover thus enable rapid response to failures but are limited to the set of precomputed alternate paths.

The fundamental question is, then, how resilient can forwarding tables be? That is, how many link failures can failover routing tolerate before connectivity is interrupted (i.e., packets are trapped in a forwarding loop, or hit a dead end) without invoking the control plane? The answer to this question depends on (1) the number of failures the forwarding scheme should withstand (e.g., resiliency to multiple simultaneous link failures is crucial in overlay networks over optical backbone networks [2], [3]), (2) the structural properties of the network graph (e.g., in terms of connectivity), and (3) the limitations imposed on the routing scheme (e.g., with/without packet marking).

The goal of this paper is to shed light on the theoretical guarantees that are achievable by *any* failover forwarding table and leverage these insights to devise better, more resilient, immediate failover schemes. We only consider link failures, not vertex failures (which are not always detectable by neighboring routers, so such fast resilient routing techniques may not apply).

We distinguish between *static* routing tables and *dynamic* routing tables. While dynamically and adaptively changing the forwarding tables at a router in response to link failures can achieve high resiliency (see, e.g., link reversal [6] and [7], [8]), current routing protocols and infrastructure do not support such stateful failover routing. Our focus is hence on static, OpenFlow-like [9], failover routing, where a router/switch matches packet headers to forwarding rules.

First, we observe that designing static routing tables that are robust to multiple failures is a relatively simple task when the forwarding decisions can rely on both the source and destination of the packet. Unfortunately, the number of forwarding entries grows quadratically with the size of the network. We seek "scalable" static failover schemes that rely on limited, locally-available information, specifically: the destination address, the packet's incoming link, and the set of non-failed links incident to the router. We note that per-incoming-link destination-based forwarding tables are a necessity as destination-based routing alone is unable to achieve robustness against even a single link failure [10], (and, moreover, entails computationally hard challenges [3], [10]–[12]).

We investigate four models of "scalable" static failover routing: basic routing, routing with packet-header rewriting, routing with packet-duplication, and probabilistic routing. We present, for each of these four models, new immediate failover schemes with provably improved resiliency over past approaches. Our results are summarized in Table I. We experimentally compare these schemes. Our findings show that a high level of resiliency is achievable even with no/little

[1]By "immediately", we mean that there is no control plane delay, but the fast failover can only happen after (a) the failure is detected and (b) the router can update its routing table to use the backup route. These delays depend on the technology used for failure detection and table management, so the resulting delays can vary substantially, but typically are much less than the time it takes for the control plane to reconverge.

TABLE I: Summary of the resiliency of routing tables for arbitrary topologies.

	Per-destination	**Per-incoming port**	**Packet header rewriting**	**Packet duplication**	**Probabilistic forwarding**	**Resiliency**
Static Routing Tables	X					$\times$ impossible for any $k \geq 2$
	X	X				✓for any $k \leq 5$
	X	X	X			✓with $> k$ bits [4], [5], k-bits [2], $\lg k$-bits, and 3-bits
	X	X		X		✓at most $2k - 1$ packets
	X	X			X	✓with limited stretch[2]
Dynamic Routing Tables	X	X	X			✓1-bit in the packet header [6]–[8]

rewriting of packet headers.

Basic (BSC) failover routing (2nd row): each packet is matched to an outgoing port only on the destination address, the incoming link, and the set of non-failed links. Past work [13], [14] (1) achieved guaranteed robustness against *only* a single link/node failure [15]–[20], (2) achieved robustness against $b\frac{k}{2}$ 1c link failures for k-connected networks [2], and (3) proved that resiliency to any set of link failures [16] cannot be achieved.

We show that resiliency to multiple link failures can be accomplished even through basic failover routing. We show how to generate static forwarding tables that are resilient to any k 1 failures for a broad variety of k-connected networks, including full-meshes, generalized hypercubes (proposed as datacenter topologies in [21]), and Clos networks (today's datacenter topologies). Motivated by these results, we make the following general conjecture:

Conjecture: For any k-connected graph, basic failover routing can be resilient to any k 1 failures.

We take a first step in this direction by proving that for any network that is k-connected for k 5, the above statement holds. We present several negative results, e.g., for natural forms of basic failover routing.

Failover routing with packet-header rewriting (HDR, 3rd row): a node has an ability to rewrite any bit in the packet header. Clearly, if it is possible to store an arbitrary amount of information in the packet header, perfect resiliency can be achieved by collecting information about every failed link that a packet encounters [4], [5]. Such approaches are not feasibly deployable in modern-day networks as the header of a packet may be too large for today's routing tables. More recent results show that for any k-connected network, k bits are sufficient to compute forwarding tables that are robust to (k 1) link failures [2]. Our focus is thus on failover routing schemes that involve only minimal rewriting of bits in the packet header, or even no rewriting whatsoever. We show that the ability to modify at most *three* bits suffices to provide the same level of resiliency.

[2]Our probabilistic method significantly outperforms a random walk in both the number of random bits it uses and the number of links it traverses to deliver a packet to the destination.

Failover routing with packet duplication (DPL, 4th row): a node has an ability to duplicate a packet (without rewriting its header) and send the copies through deterministically chosen outgoing links. We show how to compute, for any k-connected network, resilient routing tables that do not create more than $2f$ duplicates of a packet, where f is the number of failed links hit by a packet. (So, in particular, if there is no link failure, no packet duplication occurs.)

Randomized failover routing (RND, 5th row): as above, but the outgoing edge is chosen in a probabilistic manner. Observe that, in principle, in this model, even selecting an (active) outgoing edge uniformly at random achieves perfect resiliency. However, the expected delivery time of a packet, even if there was *no* link failures, would be very large – as large as (mn) in some network topologies. Instead, we present a randomized protocol that guarantees the expected delivery time to be significantly improved and gracefully growing with the number of actual link failures.

Comparing the four schemes: We experimentally evaluate the four proposed schemes both in terms of resiliency and in terms of path lengths (stretch). Our main conclusions are that (1) our positive results for the basic failover technique (which does not involve bit rewriting in the packet header) come with an average stretch of only 10%, and (2) for any k-connected network, the ability to rewrite only $\log(k)$ bits is sufficient to be resilient against k 1 link failures with only a small stretch compared to the technique that uses k bits. Hence, a high level of resiliency is achievable with little/no bit rewriting and without the overheads associated with packet duplication.

A. Organization

In Section II, we introduce our routing model and formally state the STATIC-ROUTING-RESILIENCY problem. Section III provides related work overview. In Section IV, we summarize routing techniques that will be leveraged throughout the whole paper. Section V is devoted to our main resiliency results for basic routing. Then, in Section VI and Section VII, we show that robustness to (k 1) link failures, where k is the connectivity of a graph, can be achieved through the rewriting of packet header, or if the packet can be duplicated, respectively. Section VIII is devoted to designing an algorithm that, for any k-connected graph, computes probabilistic routing functions that are robust to k 1 link failures and have bounded expected delivery time. We present the experimental evaluation of our failover schemes in Section IX. Certain impossibility results are presented in Section X. Finally, we conclude by Section XI.

II. Model

We represent the network as an undirected multigraph $G = (V; E)$, where each router in the network is modeled by a vertex in V and each link between two routers is modeled by an undirected edge in the multiset E. We denote an (undirected) edge between x and y by $fx; yg$. Each vertex v routes packets according to a *forwarding function* that matches an incoming packet to a sequence of forwarding actions. Packet *matching* is performed according to the set of active (non-failed) edges incident at v, the incoming edge, and any information stored in the packet header (e.g., destination label, extra bits), which are all information that are *locally* available at a vertex. Since our focus is on per-destination forwarding functions, we assume that there exists a unique destination $d\ 2\ V$ to which every other vertex wishes to send packets and, therefore, that the destination label is not included is the header of a packet. Forwarding *actions* consist of routing packets through an outgoing edge, rewriting some bits in the packet header, and creating duplicates of a packet.

In this paper we consider four different types of forwarding functions. We first explore a particularly simple forwarding function, which we call *basic* routing (BSC). In basic routing (Section V) a packet is forwarded to a specific outgoing edge based only on the incoming port and the set of active outgoing edges. The other two forwarding functions, which are generalization of BSC are the following ones: *probabilistic* routing, in which a vertex forwards a packet through an outgoing edge with a certain probability, *header-rewriting* routing, in which a vertex rewrites the header of a packet, and *duplication* routing, in which a vertex creates copies of a packet. Basic routing is a special case of each of these forwarding functions.

We present the formal definitions of the probabilistic, header-rewriting and duplication routing models in Section VI and Section VII, respectively.

The Static-Routing-Resiliency (Srr) problem. Given a graph G, a forwarding function f is *c-resilient* if, for each vertex $v\ 2\ V$, a packet originated at v and routed according to f reaches its destination d as long as at most c edges fail and there still exists a path between v and d. The input of the Srr problem is a graph G, a destination $d\ 2\ V$, and an integer $c > 0$, and the goal is to compute a set of resilient forwarding functions that is c-resilient. In this paper we investigate the relationship between the resiliency that can be achieved by static routing tables and the connectivity of a graph. We say a graph is *k-connected* if there exist k edge-disjoint paths between any pair of vertices in the graph. We now intuitively introduce our main routing techniques.

III. Related Work

There is a huge body of literature on related topics, and here we give only a high-level overview. We make several distinctions among the studies satisfying these requirements; the first is whether the routing algorithm can rewrite packet headers (inserting/modifying additional state). This category includes [2], [4], [22]–[34] and the general thrust of these results (with some variation) is that adding one or a few additional bits (or tunnels) can achieve resiliency to one or two link failures, whereas one can achieve resiliency to $k \quad 1$ link failures with k bits. When one allows an unlimited list of failed node/links in the packet header, [4] and [5] deliver packets as long as the network remains connected. The next category involves solutions that do not modify the packet header, and here we can further distinguish between solutions that modify the forwarding tables based on packet arrivals, and those that have static tables. The dynamic approaches can deliver packets whenever the network remains connected [7], [8]. Among the static approaches, some depend only on the destination address, and some also depend on the incoming port. The former are guaranteed to deliver packets under any arbitrary non-disconnecting set of failures only if the routing tables are not deterministic, otherwise, for deterministic static routing tables, not only the problem of protecting against one single failure may not admit a solution, but it is even hard to compute routing tables that maximize the number of vertices that are protected [3], [10]–[12]. The latter (i.e., per-incoming port static deterministic routing tables) exploit the incoming port of a packet to infer what links have failed. Our work belongs to this category. We distinguish between approaches that deterministically chose the outgoing port and those that do it in a probabilistic manner. Among the former ones, previous works are limited in several aspects: the proposed heuristics have no provable failover guarantees [13], [14]; failover mechanisms have limited guaranteed resilience against *only* one single link/node failure [15]–[20] or resiliency to $b\frac{k}{2} \quad 1c$ link failures for k-connected graphs [2]; routing focus limited to shortest-path-IP [13], [31], [32]; impossibility of achieving resiliency to any arbitrary number of link failures [16]. For specific topologies, works [35], [36] achieve resiliency to $k \quad 1$ link failures but no general methodology is described. In contrast, we show how to compute routing tables that are robust to $k \quad 1$ link failures for arbitrary k-connected graphs, with $k \quad 5$ and we show that resiliency to k link failures cannot be guaranteed for any k-connected graph with static routing tables. Most of previous work involved congestion minimization problems. Valiant [37] proposed a probabilistic routing scheme with the goal to balance the load of the underlying network. Since then, that scheme is called *Valiant Load-Balancing* (VLB), whose one of the main ingredients is randomization. VLB was extensively used in designing networks. Zhang-Shen et al. [38] employed VLB to design fault-tolerant networks with guaranteed no congestion under few router or link failures. Greenberg et al. [39] adopt VLB to reduce volatility of traffic and failure pattern of their data centers. In [40], Shepherd et al. extend VLB in order to build cost-effective networks robust to changes in demand patterns. We use probabilistic routing to reroute along multiple link failures with limited path stretch. Another line of work [41] proves that in any network with k link failures any shortest path can be constructed by concatenating at most $k + 1$ shortest paths of the original

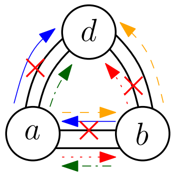

Fig. 1: A 4-connected graph with 4 arc-disjoint arborescences.

network. Furthermore, the authors provided a way to leverage that result to restore shortest paths in MPLS routing whenever multiple failures occur in the network.

IV. General Routing Techniques

As in [2], we leverage a well-known result from graph theory [42], which allows us to decompose any k-connected graph in a set of k directed spanning trees (rooted at the same vertex) such that no pair of spanning trees shares an edge in the same direction. As an example, consider Fig. 1, in which each pair of vertices is connected by two edges (ignore the red crosses) and four arc-disjoint arborescences `Blue`; `Orange`; `Red`, and `Green` are depicted by colored arrows of different styles (normal, dashed, dotted, and dash-dotted respectively). Efficient fast algorithms to compute such arborescences can be found in [43]. We now show the main techniques that we use to route along these arborescences.

Arborescence-based routing. Throughout the paper, unless specified otherwise, we let $T = fT_1; : : : ; T_k g$ denote a set of k arc-disjoint spanning arborescences of G rooted at a common destination vertex. All our routing techniques are based on a decomposition of G into T. We say that a packet is routed in *canonical* mode along an arborescence T if a packet is routed through the unique directed path of T towards the destination. If packet hits a failed edge at vertex v along T, it is processed by v (e.g., duplication, header-rewriting) according to the capabilities of a specific forwarding function and it is rerouted along a different arborescence. We call such routing technique *arborescence-based* routing. One crucial decision that must be taken is the next arborescence to be used after a packet hits a failed edge. In this paper, we propose two natural choices that represent the building blocks of all our forwarding functions. When a packet is routed along T_i and it hits a failed arc $(v; u)$, we consider the following two possible actions:

Reroute along the next available arborescence, e.g., reroute along $T_{next} = T_{(i+1) \mod k}$. Observe that, if the outgoing arc belonging to T_{next} failed, we forward along the next arborescence, i.e. $T_{(i+2) \mod k}$, and so on.

Bounce on the reversed arborescence, i.e., we reroute along the arborescence T_{next} that contains arc $(u; v)$.

We say that a forwarding function is a *circular-arborescence* routing if each vertex can arbitrarily choose the first arborescence to route a packet and, for each T_i 2 T, we use canonical routing until a packet hits a failed edge, in which case we reroute along the next available arborescence. We will show an example in Section V-A.

To grasp how bouncing enters in our picture for obtaining k 1 resiliency, consider the following case. Assume that in the network there are k=2 failed links, such that every single out of k arborescences contains one of the links. As a reminder, arborescences that we construct might share links, but not arcs. So, this example might suggest that there are scenarios in which already k=2 failed links make all the arborescences not very useful, and that no algorithm can cope with that. But, there is a twist. Let $k = 2$, and T_1 and T_2 be the two arborescences and let they share the same failed edge a. Furthermore, let a be the only failed edge T_1 and T_2 contain. If a packet hits a while routed along T_1 or T_2, then after bouncing on a the packet will reach d without any further interruption! So, we have just found a way to resolve a case in which every arborescence contains one failed link. And, that is not an isolated scenario.

Motivated by this observation, we introduce the concept of *good arborescence*. We say that an arborescence T is good if bouncing on any failed arc of T the packet reaches d without any further interruption. We studied the properties of good arborescences in our recent work [44]. In particular, the following lemma is shown.

Lemma 1. *[Lemma 4 of [44]] If G contains $f < k$ failed edges, then T contains at least k f good arborescences.*

In the next sections, we show how it is possible to achieve different degrees of resiliency by utilizing our general routing techniques and different forwarding functions (i.e., basic, probabilistic, packet-header-rewriting, and packet-duplication).

V. Basic Routing

In this section we show how to achieve (k 1)-resiliency for any arbitrary k-connected graph, with k 5, using basic forwarding functions (`BSC`), which map an incoming edge and the set of active edges incident at v to an outgoing edge. This striking result demonstrates that resiliency to multiple failures can surprisingly be achieved even without invoking the control plane and without adding any additional information in the header of a packet. We also show that for several k-connected graphs (e.g., Clos networks, full-mesh), with arbitrary positive k, there exist sets of (k 1)-resilient forwarding functions.

A. Circular Routing

We first show that circular-arborescence routing is not sufficient to achieve 3-resiliency. Consider the example in Fig. 1 with 3 vertices a, b, and c and 6 edges (depicted as black lines) $e^A_{a;b} = fa; bg$, $e^F_{a;b} = fa; bg$, $e^A_{a;d} = fa; dg$, $e^F_{a;d} = fa; dg$, $e^A_{b;d} = fb; dg$, and $e^F_{b;d} = fb; dg$, where A stands for "active" edge and F for "failed" edge (depicted with a red cross over them). Four arc-disjoint arborescences $T = f$`Blue`;`Orange`;`Red`;`Green`g are depicted by colored arrows. Let < `Blue`;`Orange`;`Red`;`Green` > be a circular ordering of the arborescences in T. We now describe how a packet p originated at a is forwarded throughout the graph using a circular-arborescence routing. Since $e^F_{a;d}$ is failed, p cannot be routed along the `Blue` arborescence. It is then rerouted through `Orange`, which also contains a failed edge $e^F_{a;b}$ incident at a. As a consequence, p is forwarded to b through the `Red` arborescence. At this point, p cannot be forwarded to d because $e^F_{b,d}$, which belongs to `Red`, failed. It is then rerouted through `Green`, which also contains a failed edge $e^F_{a,b}$ incident at b. Hence, p is rerouted again through `Blue`, which leads p to the initial state—a forwarding loop.

An intuitive explanation is the following one. Since an edge might be shared by two distinct arborescences, a packet may hit the same failed edge both when it is routed along the first arborescence and when it is routed along the second arborescence. As a consequence, even $\frac{k}{2}$ failed edges may suffice to let a packet be rerouted along the same initial vertex and initial arborescence, creating a forwarding loop. Our first positive result shows that a forwarding loop cannot arise in 2- and 3-connected graphs if circular-arborescence routing is adopted.

Theorem 2. *For any k-connected graph, with $k = 2, 3$, any circular-arborescence routing is $(k-1)$-resilient. In addition, the number of switches between trees is at most 4.*

Proof. Consider a 2-connected graph $G = (V, E)$, two arc-disjoint arborescences T_1 and T_2 of G, and an arbitrary failed edge $e = \{u, v\} \in E$. W.l.o.g, T_1 is the first arborescence that is used to route a packet p. When p hits e (w.l.o.g, at u), p cannot hit e in the opposite direction along T_2. In fact, this would mean that there exists a directed path from u to v that belongs to T_2 and that (v, u) is contained in T_2—a directed cycle.

A similar, but more involved argument, holds for the 3-connected case. There are 3 arc-disjoint arborescences T_1, T_2 and T_3 of G. W.l.o.g, T_1 is the first arborescence that is used to route a packet p. When p hits a first failed edge e_1 it is switched to be routed along T_2. Then it may not hit e_1 in other direction as it would mean that T_2 has a loop including edge e_1 which is impossible for a arborescence. Therefore packet will hit e_2 next or be delivered to d. After switching to T_3 it will have to hit e_1 in the opposite direction. Next it will have to hit e_2 in the opposite direction while being routed along T_1. Lastly, after switching to T_2 packet will not be able to hit e_2 as it will cause a loop in arborescence and it will not be able to hit e_1 as it consists of arcs from T_1 and T_3 but not T_2. □

B. 4-Connected Graphs

Let us look again at the graph in Fig. 1. It is not hard to see that a different circular ordering of the arborescences (i.e., $<$ `Blue`, `Green`, `Orange`, `Red` $>$) would be robust to any three failures. However, our first result shows that in general circular-arborescence routing is not sufficient to achieve $(k-1)$-resiliency, for any $k \geq 4$.

Theorem 3. *There exists a 4-connected graph and a set of k arc-disjoint arborescences on it, such that there does not exist any 3-resilient circular-arborescence forwarding function.*

Proof sketch. Consider the 4-connected graph in Fig. 1. There are total $4! = 24$ different circular routings on it. Taking symmetry into account - only 6 routings have to be considered (w.l.o.g `Blue` can be the first in the ordering). For each possible routing there is a set of failed edges which leads to forwarding loop. At the beginning of this section we provided an example of such situation then circular routing is $<$ `Blue`, `Orange`, `Red`, `Green` $>$. Similar failure sets exist for all 5 remaining orderings but are omitted here due to the space limitations. □

In the experimental section we show that a naive choice of the arc-disjoint arborescences will very likely cause a forwarding loop.

To overcome this, we first introduce the following key lemma, in which we show how to construct four arc-disjoint arborescences such that some of them do not share edges with each other. Then, we compute a circular-arborescence routing that is 3-resilient based on these arborescences.

Lemma 4. *For any $2k$-connected graph G, with $k \geq 1$, and any vertex $d \in V$, there exist $2k$ arc-disjoint arborescences $T_1, \ldots, T_{2k}$ rooted at d such that $T_1, \ldots, T_k$ do not share edges with each other and $T_{k+1}, \ldots, T_{2k}$ do not share edges with each other.*

We prove this lemma in the appendix. A similar lemma holds also for any $(2k+1)$-connected graph, where $k \geq 0$ (see [45] for proof).

Lemma 5. *For any $2k+1$-connected graph G, with $k \geq 1$, and any vertex $d \in V$, there exist $2k+1$ arc-disjoint arborescences $T_1, \ldots, T_{2k+1}$ rooted at d such that $T_1, \ldots, T_k$ do not share edges with each other and $T_{k+1}, \ldots, T_{2k}$ do not share edges with each other.*

The following theorem states that a circular ordering $< T_1, \ldots, T_4 >$ of the arborescences constructed as in Lemma 4 is a 3-resilient circular-arborescence routing. We will make use of the general case of Lemma 4 in Sect. VII.

Theorem 6. *For any 4-connected graph, there exists a circular-arborescence routing that is 3-resilient. In addition, the number of switches between trees is at most $2f$, where f is the number of failed edges.*

Proof. According to lemma 4 there exist arborescenses T_1, T_2, T_3, T_4 s.t. T_1 does not share edges with T_3, and T_2 does not share edges with T_4 (not that T_2 and T_3 are renamed after application of the lemma). We will now show that circular routing $< T_1, T_2, T_3, T_4 >$ is 3-resilient.

A packet p is routed along T_1 (see Fig. 2). It either reaches

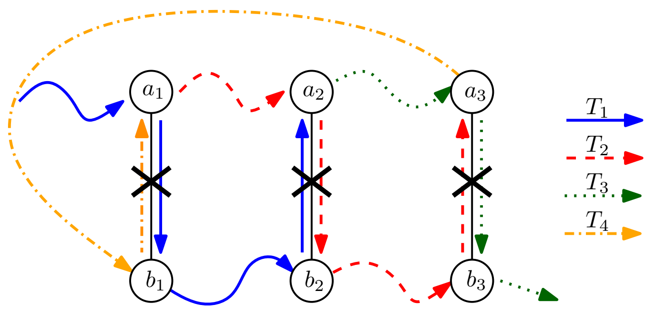

Fig. 2: Proof of Theorem 6. Curved lines represent paths in the graph.

the destination vertex d or it hits a failed edge $e_1 = fa_1; b_1g$ at a_1. In the latter case, it is rerouted along T_2. It either reaches d or it hits a failed edge $e_2 = fa_2; b_2g$ at a_2. In the latter case, observe that e_1 is a distinct edge from e_2, otherwise if $fa_1; b_1g = fb_2; a_2g$, we have a cycle in T_2. Hence, p is routed along T_3. It either reaches d or it hits a failed edge $e_3 = fa_3; b_3g$ at a_3. In the latter case, observe that e_3 is a distinct edge from both e_1 and e_2, otherwise if $fa_2; b_2g = fb_3; a_3g$, we have a cycle in T_3 and if $fa_1; b_1g = fb_3; a_3g$ then T_3 shares an edge with T_1—a contradiction. Hence, p is routed along T_4. It either reaches d or it hits a failed edge $e \;\; 2$ $ffb_1; a_1g; fb_2; a_2g; fb_3; a_3gg$. If $e \;\; = fb_3; a_3g$, T_4 contains a cycle — a contradiction. If $e \;\; = fb_2; a_2g$, T_4 shares an edge with T_2 — a contradiction. Hence, $e \;\; = fb_1; a_1g$ and p is rerouted along T_1. It either reaches d or it hits a failed edge $e^0 \; 2 \; ffa_1; b_1g; fb_2; a_2g; fb_3; a_3gg$. If $e^0 = fa_1; b_1g$, T_1 contains a cycle — a contradiction. If $e^0 = fb_3; a_3g$, T_1 shares an edge with T_3 — a contradiction. Hence, $e^0 = fb_2; a_2g$ and p is rerouted along T_2. It either reaches d or it hits a failed edge $e \; 2 \; ffa_2; a_2g; fb_3; a_3gg$. If $e = fa_2; b_2g$, T_2 contains a cycle — a contradiction. Hence, $e = fb_3; a_3g$ and p is rerouted along T_3. It either reaches d or it hits the failed edge $fa_3; b_3g$, which is not possible since T_3 does not contain a cycle. Hence p always reaches d and there are at most $2f$ switches because they occur only at encountering a failed arc and each failed edge results in 2 failed arcs. □

C. 5-*Connected Graphs*

We now leverage our second routing technique, i.e., bouncing a packet along the opposite arborescence when a packet hits a failed edge. The intuition behind bouncing a packet is the following one. When we bounce a packet along the opposite arborescence T, we know that at least one failed edge that belongs to T is not contained in the path from p to the destination vertex.

Let $T_1; : : : ; T_k$ be k arc-disjoint arborescences of G such that a circular-arborescence routing based on the first $k \quad 1$ arborescences is $(c \quad 1)$-resilient, with $c < k$. Let R be a set of forwarding functions such that: each vertex that originates a packet p, forwards it along T_k and, if a failed edge is hit along T_k, then p is routed according to the circular-arborescence based on the first $k \quad 1$ arborescences. Then, we have the following result.

Lemma 7. *The set of forwarding functions R is c-resilient.*

Proof. First we route a packet p along T_k. If p hits a failed edge $fx; yg$ at x, we switch to circular-arborescence routing based on arborescences $T_1; : : : ; T_{k \quad 1}$ starting from the arborescence that contains arc $(y; x)$. Suppose, by contradiction, that routing is not c-resilient, i.e., a forwarding loop arises with no more than c link failures. Let $e_i = (a_i; b_i)$, with $i = 1; : : : ; r < 2c \quad 2$, be the i'th failed arc hit by a packet p. Let T_i be the arborescence that contains arc $(b_1; a_1)$. Two cases are possible: (i) the forwarding loop hits edge $fa_1; b_1g$ or (ii) not.

In case (i), consider the scenario in which only edges $fa_2; b_2g; : : : ; fa_r; b_rg$ failed (less than c edges in total). If a packet p is originated by a_1 and it is initially routed along T_i, then it will hit the same failed arcs as a packet in the assumed forwarding loop and it will eventually hit $(b_1; a_1)$. Since this arc is not failed, p will enter a forwarding loop, which is a contradiction since the circular-arborescence routing on $T_1; :::; T_{k \quad 1}$ is $(c \quad 1)$-resilient.

Analogously, in case (ii), consider the scenario in which only edges $fa_2; b_2g; : : : ; fa_r; b_rg$ failed. Since the forwarding loop does not hit $(a_1; b_1)$, we have a contradiction since we assumed that the circular-arborescence routing is $(c \quad 1)$-resilient. Hence, our routing scheme is c-resilient. □

The 4-resiliency for any 5-connected graph now easily follows from Lemma 7 and Theorem 6:

Theorem 8. *For any* 5*-connected graph G there exist a set of* 4*-resilient forwarding functions. In addition, the number of switches between trees is at most* $2f$*, where f is the number of failed edges.*

Proof. According to lemma 5 there exist arborescenses $T_1; T_2; T_3; T_4; T_5$ s.t. T_1 does not share edges with T_3, and T_2 does not share edges with T_4 (not that T_2 and T_3 are renamed after application of the lemma).

Note that trees T_1–T_4 are the same as in the proof of Theorem 6. Therefore circular-arborescence on them is 3-resilient. Now from lemma 7 it follows that there is a 4-resilient routing scheme consisting of routing on T_5 until first failure is reached, then bouncing and performing circular routing afterwards. □

Since every planar graph with no parallel edges is at most 5-connected [46], the following corollary easily follows.

Corollary 9. *For any k-connected planar graph with no parallel edges there exist a set of* $(k \quad 1)$*-resilient forwarding functions.*

The following theorem shows that a certain level of resiliency can be achieved by simply using circular-arborescence routing.

Theorem 10. *For any k-connected graph there exist a set of* $b\frac{k}{2}c$*-resilient forwarding functions.*

Proof. It easily follows from Lemma 7 and the fact that every circular-arborescence routing is $(b\frac{k}{2}c \quad 1)$-resilient. □

Constrained topologies For several graph topologies that are common in Internet routing or datacenter networks, we show that $(k \quad 1)$-resilient forwarding functions can be computed in polynomial time. The list of graphs that admit $(k \quad 1)$-resilient forwarding functions encompasses cliques, complete bipartite graphs, generalized hypercubes, Clos networks, and grids [21], [46], [47]. We refer the reader to [45] for further details.

Avoiding loops for arbitrary number of failures. Observe that a packet may easily enter a forwarding loop when it is routed with circular-arborescence routing. In fact, if a packet has to be routed more than once through the same arborescence in order to reach the destination, a switch cannot distinguish whether that packet was already routed on that arborescence. This means that, if there are more failures than those that

Algorithm 1 Definition of HDR-Log-K-Bits.

```
HDR-Log-K-Bits: Given $T = \{T_1, \dots, T_k\}$ and $d$
1. Let $T_i$ be the first tree that is used to route a packet.
2. Set $currcirc := i$.
3. Repeat until the packet is delivered to $d$
   a. Route along $T_i$ until $d$ is reached or the routing hits a failed edge.
   b. If the routing hits a failed edge $a$ and $a$ is shared with arborescence $T_j$.
      (i) If $currcirc \neq i$, let $currcirc := (currcirc + 1) \mod k + 1$ and $i := currcirc$.
      (ii) Otherwise, let $i := j$.
```

Algorithm 2 Definition of HDR-3-Bits.

```
HDR-3-Bits: Given $T = \{T_1, \dots, T_k\}$ and $d$
1. Set $i := 1$.
2. Repeat until the packet is delivered to $d$
   1. Route along $T_i$ until $d$ is reached or the routing hits a failed edge.
   2. If the routing hits a failed edge $a$ and $a$ is shared with arborescence $T_j$, $i \neq j$.
      (a) Bounce and route along DFS traversal in $T_j$.
      (b) If the routing hits a failed edge in $T_j$, route back to the edge $a$.
   3. Set $i := (i + 1) \mod k + 1$
```

can be tolerated, a forwarding loop may be created. This is the case for the routing algorithms that we described so far, where a packet can be routed more than once through the same arborescence. To avoid forwarding loops for any arbitrary number of failures, without guaranteeing to deliver a packet to the destination with more than $k - 1$ failures, it is necessary to add only two extra bits in the packet header, so that a switch can detect whether a packet is being routed more than once through the same arborescence. The idea is to leverage the two extra bits to count how many times a packet has been routed through a specific arborescence, e.g., T_1. Since our routing techniques do not route more than two times on each arborescence, once our two-bits counter reaches a value of 3, a packet can be dropped since it implies that more than $k - 1$ links failed in the network.

VI. Packet header rewriting

We devote this section to algorithms that rewrite a very limited number of bits in the packet header in order to achieve $(k - 1)$-resiliency, and present two such algorithms, approached in mutually different ways. The first algorithm uses $\lceil \log k \rceil$ and the second one uses only 3 bits in the packet header. As depicted in Table I, concerning the number of bits allocated in the packet header, both algorithms substantially improve upon the previous work. Our experiments, that we present in Section IX, suggest that the algorithm that uses only $\lceil \log k \rceil$ bits is of a high practical relevance.

Algorithm 1 (HDR-Log-K-Bits) requires only $\lceil \log k \rceil$ bits in the packet header as it simply needs to store variable $currcirc$. Variable i is not stored in the packet header but is inferred from the incoming arc on which the packet is received. The correctness of Algorithm HDR-Log-K-Bits is based upon Lemma 1, i.e., there exists at least one good arborescence even if $k - 1$ links fail. Intuitively, HDR-Log-K-Bits attempt to route a packet p through each arborescence in a circular order. The current arborescence traversed by a packet in the circular order is stored in $currcirc$. Whenever a failed link is hit, a bounce operation is performed. Two cases are possible: a packet is rerouted on a good arborescence or not. In the former case, by definition of good arborescences, the packet is correctly delivered to the destination without hitting any further failed link. In the latter case, the packet will hit a failed link while being routed through the bounced arborescence, In that case, HDR-Log-K-Bits checks the value stored in $currcirc$ and it reroutes p on the next arborescence in the circular order. As such, HDR-Log-K-Bits guarantees that packet is eventually routed and bounced on each arborescence, eventually on the good one.

Algorithm HDR-3-Bits (Alg. 2) requires only 3 bits in the packet header to construct a set of $(k - 1)$-resilient forwarding functions. This algorithm extends Alg. 1 and consists of routing a packet in three phases. HDR-3-Bits attempts to route a packet p along each arborescence in a circular order (Phase 1). Whenever p hits a failed link l while being routed along an arborescence T, HDR-3-Bits reroutes p on the bounced arborescence that shares link l with T (Phase 2). In this phase, HDR-3-Bits routes p through a "traversal visit" of the bounced arborescence. A *traversal visit* of an arborescence is a cycle that traverses each vertex in the graph and does not traverse a link in the same direction twice. A traversal visit can be computed with a simple Depth-First-Search algorithm. If p does not reach the destination vertex, then it must hit a failed link l' on the bounced arborescence. In that case, since p does not contain an identifier of the last arborescence used in the circular order, HDR-3-Bits performs a "back-tracking" operation that consists in rerouting p backwards on the bounced arborescence using a reversed traversal visit until link l is hit again (Phase 3). When this happens, HDR-3-Bits determines that p was being routed on the T since the bounced arborescences shares l with T. It therefore reroutes p on the successor of T in the circular order of the arborescences (Phase 1 again). Two bits in the packet header are used to keep track of the routing phase. Each time a packet is received from a link, the third bit is used to distinguish on which of the two arborescences that packet is currently being routed. Due to the space limitations we refer readers to [48] for details and proofs.

VII. Packet duplication

In this section we show that, for any k-connected graph G, it is always possible to compute duplication forwarding functions (DPL) that are $(k - 1)$-resilient. DPL maps an incoming edge and the set of active edges incident at v to a subset of the outgoing edges at v. A packet is duplicated at v and one copy is sent to each of the edges in that set.

Algorithm 3 Definition of DPL-ALGO.

```
1) $p$ is first routed along $T_1$.
2) $p$ is routed along the same arborescence towards the destination, unless a failed edge is hit.
3) if $p$ hits a failed edge $(x; y)$ along $T_i$, then:
    a) if $i < s$: one copy of $p$ is created; the original packet is forwarded along $T_{i+1}$; the copy is forwarded along $T_l$, where $T_l$ is the arborescence that contains arc $(y; x)$.
    b) if $i = s$: $s \quad 1$ copies of $p$ are created; the original packet is forwarded along $T_{s+1}$; the $j$'th copy, with $1 \quad j \quad s \quad 1$, is routed along $T_{s+j+1}$.
    c) if $i > s$: $p$ is destroyed.
```

A naive approach would flood the whole network with copies of the same packets, i.e., each vertex creates a copy of a packet for each outgoing edge and forwards it through that edge. There are two drawbacks to this approach. First, marking packets is necessary to avoid forwarding loops. Second, at least a copy of the packet will be routed through each edge, wasting routing resources. In the following, we present an algorithm that creates a very limited number of copies of a packet and guarantees robustness against any $k \quad 1$ edge failures.

The general idea is to carefully combine the benefits of both circular-arborescence and bounce routing (as for HDR routing in Sect. VI). Circular-arborescence routing allows us to visit each arborescence, while bouncing a packet allows us to discover well-bouncing arcs (see Sect. VI for the definition of well-bouncing arcs). Bouncing packets comes at the risk of easily introducing forwarding loops as packets may be bounced between just two arborescences. Hence, we leverage our construction of arborescences from Lemma 4, which helps us to eventually hit $k \quad 1$ distinct failed edges, and we forbid any bouncing that may create a forwarding loop. For simplicity, we assume that $k = 2s$ is even.

Let G be a $2s$-connected graph and $T_1; : : : ; T_{2s}$ be $2s$ arc-disjoint arborescences such that $T_1; : : : ; T_s$ ($T_{s+1}; : : : ; T_{2s}$) do not share edges with each other (as in Lemma 4). We define the DPL-ALGO in Alg. 3 and in the following show that it provides a set of $(2s \quad 1)$-resilient forwarding functions.

We start by observing that each failed edge hit along the first s arborescences cannot be a well-bouncing arc, otherwise this would mean that at least a copy of a packet will reach d.

Lemma 11. *Let T_i be a good arborescence. If* DPL-ALGO *fails to deliver a packet to d, then $i > s$.*

Proof. If the statement would not be true, then the algorithm would route the packet to the destination using Step 3a. □

Theorem 12. *For any $2s$-connected graph and $s \quad 1$,* DPL-ALGO *computes $(2s \quad 1)$-resilient forwarding functions. In addition, the number of copies of a packet created by the algorithm is f, if $f < s$, and $2s \quad 1$ otherwise, where f is the number of failed edges.*

Proof. Suppose that DPL-ALGO fails to deliver a packet to d. Step 3a guarantees that the algorithm will route the packet along each T_i, with $1 \quad i \quad s$, and, since it fails, each T_i, $1 \quad i \quad s$, must contains an arc that belongs to a failed edge. Step 3b guarantees that the algorithm will route the packet along each T_i, with $k < i \quad 2s$, and, since it fails, each T_i, $s < i \quad 2s$, must contains an arc that belongs to a failed edge. Therefore, each T_i contains an arc that belongs to a failed edge.

This trivially implies that each T_i has a good failed arc. By Lemma 11 and since in our construction $T_{s+1}; : : : ; T_{2s}$ do not share failed edges, we have that failed edges that the algorithm approaches in $T_1; : : : ; T_s$ (s many of them) are disjoint from all the good failed arcs of $T_{s+1}; : : : ; T_{2s}$ (s many of them) otherwise at least a copy of a packet would reach d. Therefore there are at least $2s$ failed edges which is a contradiction.

Number of created copies can be counted trivially: if $f < s$ then step 3b will never be executed and one copy of packet will be created for each failure encountered at step 3a. Copies are created only while failure is encountered at T_i for $i \quad s$. As copies are all sent on T_j for $j > s$ there will be no more than $2s \quad 1$ copies in total.

□

Routing from any arbitrary initial arborescence. In contrast to the BSC and HDR forwarding techniques, in DPL-ALGO each vertex is forced to start routing packets on the same initial arborescence T_1. This approach has one main drawback: even in the absence of link failures, routing is constrained along an arborescence, which may not even be a shortest path arborescence. This leads to unacceptable high path lengths. We solve this issue by adding a counter with $d\log(s)e$ bits in the packet header. We initially set the counter to 0 and we increase it every time a packet hits a failed edge. If the counter is smaller than s, we apply step 3a from Alg. 3, otherwise, if the counter is equal to s, we apply step 3b. We do not use the counter when we route along any arborescence T_i, with $i > s$. Evaluating this version of the algorithm in Section IX, we show that it achieves high resiliency with very limited stretch. We call this algorithm DPL-LOG-K-BITS.

VIII. RANDOMIZED ROUTING

In this section, we devise a set of routing functions for G that is $(k \quad 1)$-resilient but requires a source of random bits. We extend our routing function definition, which we call randomized routing (RND), as follows: a routing function maps an incoming edge and the set of active edges incident at v to a set of pairs $(e; q)$, where e is an outgoing edge and q is the probability of forwarding a packet through e. A packet is forwarded through a unique outgoing edge.

Algorithm 4 constructs a set of $(k \quad 1)$-resilient randomized routing functions, which we call BOUNCED-RAND-ALGO.

Algorithm 4 Definition of BOUNCED-RAND-ALGO.

```
BOUNCED-RAND-ALGO: Given $T = fT_1; : : : ; T_k g$
1. $T$ := an arborescence from $T$ sampled uniformly at random (u.a.r.)
2. While $d$ is not reached
   1. Route along $T$ (canonical mode)
   2. If a failed edge is hit then
      (a) With probability 1=2, replace $T$ by an arborescence from $T$ sampled u.a.r.
      (b) Otherwise, bounce the failed edge and update $T$ correspondingly
```

A sketch of correctness. Assume that we, magically, know whether the arborescence we are routing along is a good one or not. Then, on a failed edge we could bounce if the arborescence is good, or switch to the next arborescence otherwise. And, we would not even need any randomness. However, we do not really know whether an arborescence is good or not since we do not know which edges will fail. To alleviate this lack of information we use a random guess. So, each time we hit a failed edge we take a guess that the arborescence is good, where the parameter q estimates our likelihood. Notice that BOUNCED-RAND-ALGO implements exactly this approach. As an example, consider Fig. 1. If a packet originated at a is first routed through `Orange` and the corresponding outgoing edge $e^F_{a;b}$ is failed, then the packet is forwarded with probability q to an arborescence from T sampled u.a.r. and with probability $1 \quad q$ through `Green`, which shares the outgoing failed edge $e^F_{a;b}$ with `Red`.
By leveraging Lemma 1, we can show that BOUNCED-RAND-ALGO leads to $(k \quad 1)$-resilient routing with the number of switches gracefully growing with the number of failed links.

Theorem 13 (Theorem 11 of [44]). *Given a k-connected graph G, destination d and a decomposition of G into k arc-disjoint arborescences T rooted at d, there exists a $(k \quad 1)$-resilient algorithm that delivers a packet to d after $O\left(\frac{k}{k \quad f} H\right)$ hops in expectation, where H is the length of a longest path of any arborescence of T and f the number of failed edges. The algorithm uses randomization only when encounetrs a failed edge. In particular, if $f = 0$, the algorithm is deterministic.*

A. Bouncing is Efficient

It might be tempting to implement a variation of BOUNCED-RAND-ALGO that on each failed edge switches to another arborescences chosen uar, i.e. to ignore step 2(2)b in Algorithm 4. Let RAND-ALGO denote such a variant. The following theorem shows that BOUNCED-RAND-ALGO significantly outperforms RAND-ALGO. See [44] for a proof.

Theorem 14. *For any $k > 0$, there exists a $2k$ edge-connected graph on $O(N)$ vertices and $O(k^2 + kN)$ edges, a set of $2k$ arc-disjoint spanning trees, and a set of $k \quad 1$ failed edges, such that the expected number of tree switches with* RAND-ALGO *is (k^2). Furthermore, the routing makes $(k^2 N)$ hops in expectation.*

IX. EXPERIMENTS

We experimentally evaluate the four proposed schemes both in terms of resiliency and in terms of path lengths (stretch). Our main conclusions are that (1) our positive results for basic failover technique (which does not involve marking packets) come with an average stretch of only 10%, and (2) for any k-connected network, we show that the ability to rewrite only $d\log ke$ bits is sufficient to be resilient against $k \quad 1$ link failures with only small stretch compared to the technique that uses k bits. Hence, a high level of resiliency is achievable with little/no packet rewriting of bits in the packet header and without the overheads associated with packet duplication.

First, we assess the effectiveness of the BSC-ALGO, which is based on a circular-arborescence forwarding function. Recall that BSC-ALGO is based on a special construction of a set of arc-disjoint arborescences. We show that an arbitrary set of arbitrary arc-disjoint arborescences would very likely be prone to forwarding loops.

Arbitrary arc-disjoint arborescences are not 3-resilient. We experimentally quantify the amount of routers that are no longer able to send packets to a destination vertex when circular-arborescence is used on an arbitrary set of arc-disjoint arborescences, Fig. 3. We generate 1000 different 4-connected random networks with sizes ranging from 10 to 40 vertices. For each network with N routers, we consider $320 \quad N$ random 3-link failures scenarios. We then count the number of routers that are no longer able to reach the destination router (i.e., are trapped in a forwarding loop) in at least one failure scenario. As shown in Fig. 3, roughly 65% of the routers lost connectivity to the destination vertex in at least one failure scenario. We point out that we are only providing a lower bound as an exploration of all possible 3-link failures in large networks is computationally unfeasible. In contrast, our construction of arc-disjoint arborescences described in Lemma 4 guarantees that no pair of vertices is disconnected in any 4-connected network for any 3-link failures.

Path stretch in the absence of failures. In [2] it was shown that arc-disjoint arborescences have limited stretch with respect to shortest paths in the absence of link failures. The authors also observe that, if packet header marking is allowed by the forwarding function, a single extra bit can be used to switch to failover routing only when a packet hits a failed link. Otherwise, a packet is forwarded according to any arbitrary scheme defined by a network operator (e.g., shortest paths). We omit the results for the path stretch in the absence of failures as they are similar to the ones already obtained in [2].

Little/no bit rewriting in packet header is sufficient for high resiliency and low stretches. We use as a point of reference for our evaluation the algorithm presented in [2], which uses k bits in the packet header. We define the stretch of a routing function R as the ratio between the number of links traversed by algorithm and the number of links traversed by the algorithm in [2]. We generated 1000 different 4-connected random networks with 100 routers. For each network we look at 3200 random link failures scenarios. In Fig. 4, Fig. 5, and Fig. 6, we show the cumulative distribution function of the path stretch from each source vertex to a specific destination

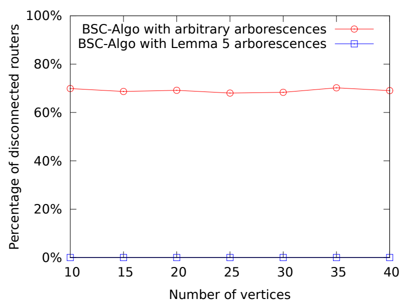

Fig. 3: 4-connected, 3 link failures.

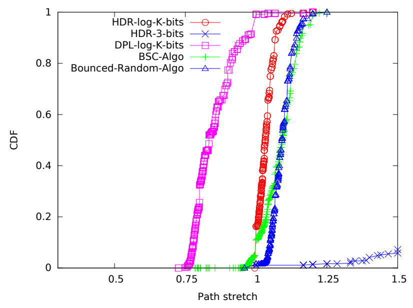

Fig. 4: 4-connected, 1 link failure.

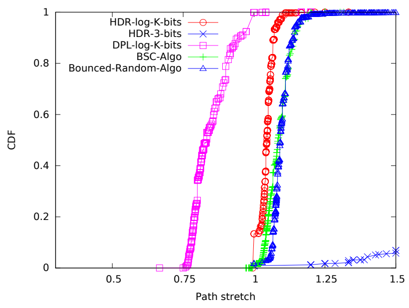

Fig. 5: 4-connected, 2 link failures.

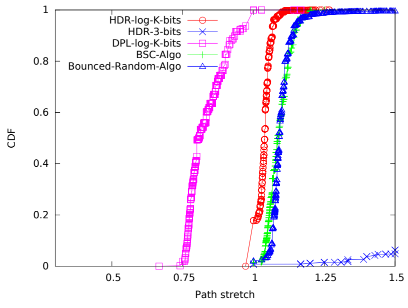

Fig. 6: 4-connected, 3 link failures.

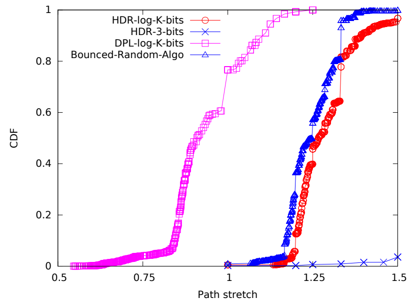

Fig. 7: 8-connected, 4 link failures.

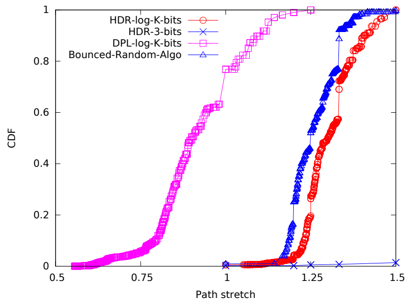

Fig. 8: 8-connected, 7 link failures.

using our four 3-resilient algorithms, i.e., BSC-ALGO, which routes packets based on a circular-arborescence forwarding function, HDR-LOG-K-BITS, which rewrites $\log(k)$ bits in the packet header, HDR-3-BITS, which rewrites only 3 bits in the packet header, and DPL-LOG-K-BITS, which rewrites $\log(k)$ bits in the packet header and possibly creates duplicates of a packet, for 1, 2, and 3 link failures. We stress the fact that in all the depicted graphs, we only compute the stretch for those packets that actually hit at least one failed link. We first observe that no rewriting of bits in the packet header (i.e., BSC-ALGO) leads to surprisingly limited average stretch, i.e., 90% of the packets have stretch smaller than 1.2. While BOUNCED-RAND-ALGO performs similarly to the deterministic scheme with no packet-header rewriting, HDR-LOG-K-BITS reduces the average stretch to roughly 1.1. Not surprisingly, DPL-LOG-K-BITS can reduce stretch much further than our comparison algorithm as it can explore different paths in the network at the same time[3]. Finally, we observe that the path stretch in HDR-3-BITS is unacceptable when compared to the other approaches. The main reason is that packets are not directly routed through the destination vertex along an arborescence (see Sect. VII). We finally compare in Fig. 7 and Fig. 8 the performance of the three other (k 1)-resilient algorithms on 8-connected networks. We observe a similar trend to the one observed for 4-connected networks.

X. IMPOSSIBILITY RESULTS FOR BASIC ROUTING

We now show that simplified forms of failover forwarding functions are not sufficiently powerful. It is well-known that without matching the incoming-edge it is not even possible to

[3]The stretch of DPL-LOG-K-BITS is computed on the first copy of a packet that reaches the destination.

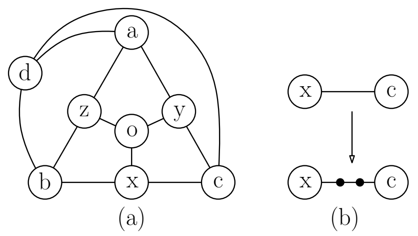

Fig. 9: (a) No circular routing functions can guarantee 2-resiliency. (b) Edge transformation.

construct 1-resilient static forwarding functions [10]. To overcome this, [14] suggests to route packets based on a circular ordering of the edges incident at each vertex. Namely, a set of forwarding functions is *link-circular* if each vertex v routes packets based on an ordered circular sequence $< e_1; : : : ; e_l >$ of its incident edges as follows. If a packet p is received from an edge e_i, then v forwards it along e_{i+1}. If the outgoing edge e_{i+1} failed, v forwards p through e_{i+2}, and so on. We show that this simplified forwarding functions cannot provably guarantee (k 1)-resiliency even for 3-connected graphs.

Theorem 15. *There is a* 3*-connected graph* G *for which no* 2*-resilient link-circular forwarding function exists.*

Proof. Consider the 3-connected graph shown in Fig. 9(a), where d is the destination. Suppose, by contradiction, that there exists a 2-resilient set of circular routing functions. Since the graph is symmetric, w.l.o.g, assume that o routes clockwise, i.e., a packet received from x is sent to z, from z to y, and from y to x. Also, w.l.o.g, o sends its originated packet p to y when none of its incident edges fail.

We first claim that vertices y, a, and z route counterclockwise. Suppose, by contradiction, that (i) y routes clockwise, or (ii) a routes clockwise, or (iii) z routes clockwise. For each case, consider the following failure scenarios. In case (i), suppose both edges $(a; d)$ and $(z; b)$ fail. In case (ii), suppose both edges $(y; c)$ and $(z; b)$ fail. In case (iii), suppose both edges $(y; c)$ and $(a; d)$ fail. In each case packet p is routed along $(y; a; z; o; y)$ and a forwarding loop arises—a contradiction.

Observe now that, in the absence of failures, if c sends a packet p to x, if x routes clockwise it forwards it directly to b, otherwise, if x routes counterclockwise, p is forwarded through o, z, a, y, o, x, and, also in this case, to b. Consider the scenario where both edges $(c; d)$ and $(b; d)$ failed. A packet p received by y from o is routed from c to x and, because of the previous observation, to b. After that, it is routed through $(z; o; y)$ and a forwarding loop arises—a contradiction. □

We now exploit the previous theorem to state another impossibility result, which shows that the connectivity between two vertices, i.e., the maximum amount of disjoint paths between the two vertices, does not match the resiliency guarantee for these two vertices. In other words, even if a vertex v is k-connected to the destination (but *not* the entire graph), it is not possible to guarantee that a packet originated at v will reach d when k 1 edges fail. Clearly, if we want to protect against k 1 failures a single vertex that is k-connected to d, we can safely route along its k edge-disjoint paths one after the other until the packet reaches its destination. However, if there are more vertices to be protected, it may be not possible to protect all of them. We say that a forwarding function is *strong-connectivity-resilient* if each packet that is originated by a vertex v that is k-connected to the destination d, can be routed towards the destination as long as less than k edges fail. By leveraging Theorem 15 and using a simple graph transformation we can show that strong-connectivity-resilient routing functions are not always achievable.

Theorem 16. *There are a graph G^0 and a destination d for which no set of strong-connectivity-resilient forwarding functions exists.*

Proof. Consider the 3-connected graph G shown in Fig. 9(a), where d is the destination. We construct a new graph G^0 from G by applying edge transformations as shown on Fig. 9(b). Each edge in the original graph is replaced by a path consisting of three edges. We call the new added vertices *intermediate* (depicted as small black circles) and the old ones *original*. Each original vertex of G retains its 3-connectivity to d.

It is easy to see that intermediate vertices must forward a packet received through one edge to the other one, if it did not fail. Otherwise, if an intermediate vertex v bounces back to a vertex u a packet, then if all edges incident at u fail, except $(v; u)$, a forwarding loop arises. This implies that we only need to compute routing functions at original vertices.

We now prove that the routing functions at the 8 original vertices, except d, must be link-circular. Once we prove this, the statement of the theorem easily follows from Theorem 15, where we proved that no link-circular routing functions can guarantee 2-resiliency on G.

From now on, we will consider only failures between two intermediate vertices, thus a routing table at each original vertex consists of just four entries: Where to send a packet received from each of its three neighbors n_1, n_2, and n_3 and where to send a locally originated packet. We can discard the last entry as it does not influence if a routing table is circular. Hence, we simplify our routing table notation as follows. Let $f^v(n) = n^0$ be a routing table at vertex v such that a packet received from a neighbor n is forwarded to a neighbor n^0.

We make the following observations. First, for each original vertex v, we have that $f^v(n) \not= n$, with n 2 $fn_1; n_2; n_3g$ i.e. no vertex bounces a packet back to the edge where it received it, exactly as in the case of intermediate vertices. Second, all entries in the routing table are distinct. Otherwise, suppose by contradiction that, w.l.o.g., $f^v(n_1) = f^v(n_2) = n_3$ and $f^v(n_3) = n_1$. If both n_1 and n_3 have a dead-end ahead because of two edge failures, then a forwarding loop among n_3, v, and n_1 arises. Hence, the routing function at each vertex must be link-circular. Since a link-circular routing function at intermediate vertices consists in forwarding a packet to the other edge, it easily follows that the same link-circular routing functions at original vertices are 2-resilient for G. □

We show that there exists a limit on the resiliency that can be attained in a k-connected graph, in which each vertex is k-connected to the destination. It was proved in [16] that resiliency against any failures that do not disconnect a sender from d cannot be guaranteed. We claim a stronger bound.

Theorem 17. *There is a 2-connected graph for which no set of 2-resilient forwarding functions exists.*

Proof. Consider the graph G^0 constructed in Theorem 16 (see Fig. 9). This graph is obviously 2-connected. Any 2-resilient forwarding functions on it are also strong-connectivity-resilient forwarding functions (because 2-resilient forwarding functions forward correctly in case of any 2 failures starting from any vertex, including original 8 vertices). However, according to Theorem 16 no such routing function exists. □

XI. Conclusions

We presented the Static-Routing-Resiliency problem and explored the power of static fast failover routing in a variety of models: deterministic routing, routing with packet-duplication, routing with packet-header-rewriting, and randomized routing. Our results suggest that even under severe restrictions on forwarding (no/little rewriting of bits in the packet header) a high-level of resiliency is achievable with negligible stretch.

XII. Acknowledgements

This research is (in part) supported by European Union's Horizon 2020 research and innovation programme under the ENDEAVOUR project (grant agreement 644960) and by Swiss National Science Foundation (grant number P1ELP2_161820).

References

[1] P. Pan, G. Swallow, and A. Atlas, "Fast Reroute Extensions to RSVP-TE for LSP Tunnels," *RFC 4090, IETF, 2005*, 2005.

[2] T. Elhourani, A. Gopalan, and S. Ramasubramanian, "IP Fast Rerouting for Multi-Link Failures," in *Proc. IEEE INFOCOM*, 2014.

[3] A. Atlas and A. Zinin, "Basic Specification for IP Fast Reroute: Loop-Free Alternates," *RFC 5286, IETF*, 2008.

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

[5] B. Stephens, A. L. Cox, and S. Rixner, "Plinko: Building Provably Resilient Forwarding Tables," in *Proc. of HotNets*. ACM, 2013.

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

[7] J. Liu, A. Panda, A. Singla, B. Godfrey, M. Schapira, and S. Shenker, "Ensuring Connectivity via Data Plane Mechanisms," in *Proc. of NSDI*, 2013, pp. 113–126.

[8] J. Liu, B. Yan, S. Shenker, and M. Schapira, "Data-driven Network Connectivity," in *Proc. of HotNets*. ACM, 2011, p. 8.

[9] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *CCR*, 2008.

[10] K.-W. Kwong, L. Gao, R. Guérin, and Z.-L. Zhang, "On the Feasibility and Efficacy of Protection Routing in IP Networks," *ToN*, 2011.

[11] M. Borokhovich and S. Schmid, "How (Not) to Shoot in Your Foot with SDN Local Fast Failover," in *OPODIS*, 2013, pp. 68–82.

[12] G. Schollmeier, J. Charzinski, A. Kirstadter, C. Reichert, K. J. Schrodi, Y. Glickman, and C. Winkler, "Improving the Resilience in IP Networks," in *Proc. HPSR*, 2003.

[13] A. Atlas and A. Zinin, "U-turn Alternates for IP/LDP Fast-Reroute," *IETF Internet draft version 03*, February 2006.

[14] B. Yang, J. Liu, S. Shenker, J. Li, and K. Zheng, "Keep Forwarding: Towards K-link Failure Resilient Routing, INFOCOM," in *Proc. IEEE INFOCOM*, 2014, pp. 1617–1625.

[15] G. Enyedi, G. Rétvári, and T. Cinkler, "A Novel Loop-free IP Fast Reroute Algorithm," in *Proc. EUNICE*, 2007.

[16] J. Feigenbaum, P. B. Godfrey, A. Panda, M. Schapira, S. Shenker, and A. Singla, "On the resilience of routing tables," in *PODC*, July 2012.

[17] S. Nelakuditi, S. Lee, Y. Yu, Z.-L. Zhang, and C.-N. Chuah, "Fast Local Rerouting for Handling Transient Link Failures," *ToN*, 2007.

[18] J. Wang and S. Nelakuditi, "IP Fast Reroute with Failure Inferencing," in *Workshop on Internet Network Management*, 2007.

[19] B. Zhang, J. Wu, and J. Bi, "RPFP: IP fast reroute with providing complete protection and without using tunnels," in *IWQoS*, 2013.

[20] Z. Zhong, S. Nelakuditi, Y. Yu, S. Lee, J. Wang, and C. nee Chuah, "Failure Inferencing based Fast Rerouting for Handling Transient Link and Node Failures," in *Proc. IEEE INFOCOM*, 2005.

[21] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers," in *Proc. of SIGCOMM*, 2009.

[22] B. Stephens and A. L. Cox, "Deadlock-Free Local Fast Failover for Arbitrary Data Center Networks," in *Proc. IEEE INFOCOM*, 2016.

[23] B. Stephens, A. L. Cox, and S. Rixner, "Scalable Multi-Failure Fast Failover via Forwarding Table Compression," in *SOSR*, 2016.

[24] A. Gopalan and S. Ramasubramanian, "Multipath Routing and Dual Link Failure Recovery in IP Networks Using Three Link-Independent Trees," in *Proc. ANTS*, 2011, pp. 1–6.

[25] S. Kini, S. Ramasubramanian, A. Kvalbein, and A. Hansen, "Fast recovery from dual-link or single-node failures in ip networks using tunneling," *IEEE/ACM Trans. Networking*, vol. 18, no. 6, pp. 1988–1999, Dec 2010.

[26] A. Kvalbein, A. F. Hansen, T. Čičic, S. Gjessing, and O. Lysne, "Multiple Routing Configurations for Fast IP Network Recovery," *IEEE/ACM Trans. Networking*, vol. 17, no. 2, pp. 473–486, April 2009.

[27] S. Lor, R. Landa, and M. Rio, "Packet re-cycling: eliminating packet losses due to network failures," in *HotNets IX*. ACM, 2010, p. 2.

[28] M. Motiwala, M. Elmore, N. Feamster, and S. Vempala, "Path Splicing," in *Proc. of SIGCOMM*, 2008, pp. 27–38.

[29] G. T. Nguyen, R. Agarwal, J. Liu, M. Caesar, P. B. Godfrey, and S. Shenker, "Slick Packets," *SIGMETRICS*, 2011.

[30] S. Sae Lor, R. Landa, R. Ali, and M. Rio, "Handling Transient Link Failures Using Alternate Next Hop Counters," in *NETWORKING*, 2010.

[31] M. Shand, S. Bryant, and S. Previdi, "IP Fast Reroute using Not-Via Addresses," *IETF Internet draft version 05*, March 2010.

[32] K. Xi and H. J. Chao, "IP Fast Reroute for Double-link Failure Recovery," in *Proc. IEEE GLOBECOM*, 2009.

[33] M. Xu, Q. Li, L. Pan, and Q. Li, "MPCT: Minimum Protection Cost Tree for IP Fast Reroute Using Tunnel," in *Proc. IWQoS*. IEEE Press, 2011, pp. 16:1–16:3.

[34] G. Robertson and S. Nelakuditi, "Handling Multiple Failures in IP Networks through Localized On-Demand Link State Routing," *IEEE Transactions on Network and Service Management*, vol. 9, no. 3, 2012.

[35] V. Liu, D. Halperin, A. Krishnamurthy, and T. E. Anderson, "F10: A Fault-Tolerant Engineered Network," in *Proc. of NSDI*, 2013.

[36] C. Filsfils, P. Francois, B. Decraene, J. Uttaro, N. Leymann, and M. Horneffer, "Loop-Free Alternate (LFA) Applicability in Service Provider (SP) Networks," *RFC 6571, IETF*, 2012.

[37] L. G. Valiant, "A scheme for fast parallel communication," *SIAM journal on computing*, vol. 11, no. 2, pp. 350–361, 1982.

[38] R. Zhang-Shen and N. McKeown, "Designing a fault-tolerant network using valiant load-balancing," in *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*. IEEE, 2008.

[39] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Vl2: a scalable and flexible data center network," in *ACM SIGCOMM computer communication review*, vol. 39, no. 4. ACM, 2009, pp. 51–62.

[40] F. Shepherd and P. Winzer, "Selective randomized load balancing and mesh networks with changing demands," *Journal of Optical Networking*, vol. 5, no. 5, pp. 320–339, 2006.

[41] A. Bremler-Barr, Y. Afek, H. Kaplan, E. Cohen, and M. Merritt, "Restoration by path concatenation: Fast recovery of mpls paths," in *Proceedings of the Twentieth Annual ACM Symposium on Principles of Distributed Computing*, ser. PODC '01. New York, NY, USA: ACM, 2001, pp. 43–52. [Online]. Available: http://doi.acm.org/10.1145/383962.383980

[42] J. Edmonds, "Edge-disjoint branchings," *Combinatorial Algorithms*, pp. 91–96, 1972.

[43] A. Bhalgat, R. Hariharan, T. Kavitha, and D. Panigrahi, "Fast Edge Splitting and Edmonds' Arborescence Construction for Unweighted Graphs," in *Proc. SODA*, 2008.

[44] M. Chiesa, A. Gurtov, A. Mądry, S. Mitrović, I. Nikolaevskiy, M. Schapira, and S. Shenker, "On the resiliency of randomized routing against multiple edge failures," *Accepted to ICALP*, 2016.

[45] M. Chiesa, I. Nikolaevskiy, A. Panda, A. Gurtov, M. Schapira, and S. Shenker, "Exploring the limits of static failover routing," *CoRR*, vol. abs/1409.0034, 2014. [Online]. Available: http://arxiv.org/abs/1409.0034

[46] R. Diestel, *Graph Theory*. Springer, February 2000.

[47] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *SIGCOMM*, 2008.

[48] M. Chiesa, I. Nikolaevskiy, S. Mitrović, A. Gurtov, A. Mądry, M. Schapira, and S. Shenker, "The quest for resilient (static) forwarding tables," *Accepted to INFOCOM*, 2016.

[49] W. Mader, " A reduction method for edge-connectivity in graphs," *Ann. Discrete Math.*, vol. 3, pp. 145–164, 1978.

Appendix

LEMMA 4. *For any $2k$-connected graph G, with k 1, and any vertex d 2 V, there exist $2k$ arc-disjoint arborescences $T_1;:::;T_{2k}$ rooted at d such that $T_1;:::;T_k$ do not share edges with each other and $T_{k+1};:::;T_{2k}$ do not share edges with each other.*

Proof. It is well known (see [49]) that any undirected graph $G = (V;E)$ is k-edge-connected if and only if G can be constructed from the initial graph of two nodes connected by k parallel edges by the following four operations, which keep the graph k-connected:

(i) add an edge,

(ii) pinch $d\frac{k}{2}e$ edges with a new node z^0,

(iii) pinch $b\frac{k}{2}c$ edges with a new node z^0 and add an edge connecting z^0 with an existing node,

(iv) pinch $b\frac{k}{2}c$ edges with a new node z^0, pinch then again in the resulting graph $b\frac{k}{2}c$ edges with another new node

z so that not all of these $b\frac{k}{2}c$ edges are incident to z^0, and finally connect z and z^0 by a new edge.

In addition, the initial graph can be such that it contains at least an arbitrary chosen vertex of G.

It is trivial to construct $2k$ arborescenses with desired properties for initial graph and it is possible to construct such arborescenses for the graph obtained at each step using arborescenses for graph from previous step.

Let G be a k-connected graph, and G^0 a graph obtained by applying operation i–iv. If we are given a list $(T_1; : : : ; T_k)$ of arborescences of G, then we can construct a list $(T_1^0; : : : ; T_k^0)$ of arborescences for G^0.

Operation i: The addition of an edge does not introduce any new vertex in G, so we set $T_i^0 := T_i$.

Operation ii and operation iii: Let z^0 denote the vertex added to G in order to obtain G^0. Initially, we let $T_i^0 := T_i$, and then modify each T_i^0, so that $(T_1^0; : : : ; T_k^0)$ is a list of ADBED arborescences of G^0, in two phases. In the first phase we alter each T_i^0 that contains a pinched arc, and in the second phase we modify the remaining ones.

The first phase. For each edge $e = fx; yg\ 2\ (E(G)\ n\ E(G^0))$, i.e. for each pinched edge, if arc $(x; y)$ belongs to T_i, let $e_1 = fx; z^0g$ and $e_2 = fy; z^0g$ be the two edges that are split off from G^0 in order to obtain e. We then add arcs $(x; z^0)$ and $(z^0; y)$ to T_i^0 and remove $(x; y)$.
If after the changes any T_i^0 is not an arborescence, we remove outgoing edges at z^0 until T_i^0 is an arborescence. This can be done by simply breaking cycles at z^0 and removing multiple paths from z^0 to d at z^0.

Now, we show some properties of the currently obtained $T_1^0; : : : ; T_k^0$.

First, observe that z^0 has at most one outgoing arc in each of the arborescences as we remove all the cycles, and parallel paths from z^0 to d.

Second, by the construction of $T_1^0; : : : ; T_k^0$ and the properties of $T_1; : : : ; T_k$ we have that each edge incident to z^0 is shared by at most one arborescence in $fT_1^0; : : : ; T^0_{b\frac{k}{2}c}g$ and at most one arborescence in $fT^0_{b\frac{k}{2}c+1}; : : : ; T^0_{2b\frac{k}{2}c}g$.

Third, observe that there are at most $k{=}2$ incoming arcs at z^0 belonging to $T_1^0; : : : ; T^0_{b\frac{k}{2}c}$ $(T^0_{b\frac{k}{2}c+1}; : : : ; T^0_{2b\frac{k}{2}c})$. If it would not be the case, then it would mean that at least an edge in $E(G^0)\ n\ E(G)$ is shared by two arborescences among $T_1^0; : : : ; T^0_{b\frac{k}{2}c}$ $(T^0_{b\frac{k}{2}c+1}; : : : ; T^0_{2b\frac{k}{2}c})$ in G. However, that would contradict, along with out construction of $T_1^0; : : : ; T_k^0$ would contradict that $(T_1; : : : ; T_k)$ is ADBED of G.

The second phase. For each arborescence T_i^0 that has no outgoing arcs at z^0, we do as follows. W.l.o.g., assume that $i\quad bk{=}2c$. We add into T an arbitrary outgoing arc at z^0 such that the symmetric incoming arc is not contained in any tree in $fT_1^0; : : : ; T^0_{b\frac{k}{2}c}g$.

Next, our goal is to argue that there always exists an edge $fx; yg\ 2\ N(z^0)$ that is not shared by any arborescence in $fT_1^0; : : : ; T^0_{b\frac{k}{2}c}g$.
Observe that z^0 has at least k incident edges. On the other hand, as we have noted, there exist at most $bk{=}2c$ incoming arcs at z^0 belonging to $T_1^0; : : : ; T^0_{b\frac{k}{2}c}$ and, at most $b\frac{k}{2}c\quad 1$ outgoing arcs that belong to $T_1^0; : : : ; T^0_{b\frac{k}{2}c}$. Hence, there exist at least $k\quad (bk{=}2c + bk{=}2c\quad 1)\quad 1$ edges that are not shared by any of $T_1^0; : : : ; T^0_{b\frac{k}{2}c}$. This means that there exists an arc $(x; z^0)$ that is not shared by any arborescence among $T_1^0; : : : ; T^0_{b\frac{k}{2}c}$.

Operation iv: In this case, we have two additional vertices z^0 and z. After we pinch at least $b\frac{k}{2}c$ edges to z^0, we do the same modifications applied for operations ii and iii. Since the degree of z^0 may be $k\quad 1$ at most one arborescence T_h^0 will not have an outgoing arc at z^0. After we pinch at least $b\frac{k}{2}c$ edges to z, we do the same modifications applied for operations ii and iii. Since the degree of z may be $k\quad 1$ at most one arborescence T_j^0 will not have an outgoing arc at z.

After that, we add an edge between z and z^0. W.l.o.g., let assume that $1\quad j\quad b\frac{k}{2}c$. If $j \not= h$ and $j\quad b\frac{k}{2}c < h$, we can safely add arc $(z; z^0)$ into T_j^0 and arc $(z^0; z)$ into T_h^0.

If $j = h$, we cannot add both arcs $(z; z^0)$ and $(z^0; z)$ into T_j^0, because it induces a cycle. We therefore consider an arbitrary arborescence T_f^0, where $1\quad f \not= j\quad b\frac{k}{2}c$. We add either $(z; z^0)$ or $(z^0; z)$ into T_f^0 in such a way that T_f^0 is a directed acyclic graph. W.l.o.g, let $(z; z^0)$ be the arc added into T_f^0. We then remove the outgoing arc $(z; x)$ of T_f^0 from T_f^0 and add it into T_j^0. We also add $(z^0; z)$ into T_j^0.

If $h\quad b\frac{k}{2}c$ we cannot add both arcs $(z; z^0)$ and $(z^0; z)$ into T_j^0 and T_h^0 as these arborscenses should not share an edge. Similarly to the case before we can choose some $f > b\frac{k}{2}c$ and add arc $(z; z^0)$ to T_f^0. Now we can move arc $(z; x)$ to T_j^0 from T_f^0. Then also add arc $(z^0; z)$ into T_j^0. Lastly, we add arc $(z; z^0)$ to T_h^0.

We showed that after applying operations i–iv it is possible to reconstruct arborescenses with desired properties. Therefore after applying all operations we will construct arborescenses for given graph G.

□

Marco Chiesa is a postdoctoral researcher at the Institute of Information and Communication Technologies, Electronics, and Applied Mathematics (ICTEAM) at the University of Louvain, Belgium. His research interests include Internet routing optimization, privacy, and security. He holds a Ph.D. in Computer Science from Roma Tre University (awarded in 2014).

Ilya Nikolaevskiy is a doctoral student at the Aalto university, Finland. He received his M.Sc degree from Petrozavadsk state university, Russia in 2011. His research interests include routing optimization, algorithm design and cryptography.