On the Resiliency of Randomized Routing Against Multiple Edge Failures *

Marco Chiesa[1], Andrei Gurtov[2,5], Aleksander Mądry[3], Slobodan Mitrović[4], Ilya Nikolaevskiy[5], Michael Schapira[6], and Scott Shenker[7,8]

1 **Université catholique de Louvain, Belgium**
2 **Helsinki Institute for Information Technology, Finland**
3 **Massachusetts Institute of Technology, US**
4 **École Polytechnique Fédérale de Lausanne, Switzerland**
5 **Aalto University, Department of Computer Science, Finland**
6 **Hebrew University of Jerusalem, Israel**
7 **University of California, Berkeley, US**
8 **International Computer Science Institute, US**

Abstract

We study the STATIC-ROUTING-RESILIENCY *problem, motivated by routing on the Internet: Given a graph $G = (V, E)$, a unique destination vertex d, and an integer constant $c > 0$, does there exist a static and destination-based routing scheme such that the correct delivery of packets from any source s to the destination d is guaranteed so long as (1) no more than c edges fail and (2) there exists a physical path from s to d? We embark upon a study of this problem by relating the edge-connectivity of a graph, i.e., the minimum number of edges whose deletion partitions G, to its resiliency. Following the success of randomized routing algorithms in dealing with a variety of problems (e.g., Valiant load balancing in the network design problem), we embark upon a study of randomized routing algorithms for the* STATIC-ROUTING-RESILIENCY *problem. For any k-connected graph, we show a surprisingly simple randomized algorithm that has expected number of hops $O(|V|k)$ if at most $k-1$ edges fail, which reduces to $O(|V|)$ if only a fraction α of the links fail (where $\alpha < 1$ is a constant). Furthermore, our algorithm is deterministic if the routing does not encounter any failed link.*

1998 ACM Subject Classification C.2.2 Network Protocols

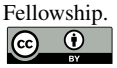
Keywords and phrases Randomized, Routing, Resilience, Connectivity, Arborescenses

1 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. The most well-known of these is Fast Reroute in MPLS where, upon a link failure, packets are sent

* The authors would like to thank to anonymous reviewers. This research is (in part) supported by European Union's Horizon 2020 research and innovation programme under the ENDEAVOUR project (grant agreement 644960), by Swiss National Science Foundation (grant number P1ELP2_161820), the NSF award 1553428, and a Sloan Research Fellowship.



along a precomputed alternate path without waiting for the global recomputation of routes [22]. This, and other similar forms of fast failover thus enable rapid response to failures but are limited to the set of precomputed alternate paths. Most of existing approaches protect only from a single failure, however in many scenarios (e.g. overlay networks [10], highly-connected large datacenter networks [14]) multiple failures at the same time may be a common occurrence.

The goal of this paper is to perform a theoretical study of failover routing. The fundamental question is, how resilient can failover routing be? That is, how many link failures can failover routing schemes tolerate before connectivity is interrupted (i.e., packets are trapped in a forwarding loop, or hit a dead end)? The answer to this question depends on both the structural properties of the graph, and the limitations imposed on the routing scheme.

Clearly, if it is possible to store arbitrary amount of information in the packet header, perfect resiliency can be achieved by collecting information about every failed link that is hit by a packet [18, 25]. 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. Our focus is thus on failover routing schemes that do not involve any change in the packet headers. Another traditional approach to achieving high resiliency is implementing stateful routing, i.e., storing information at a node every time a packet is seen being received from a different incoming link (see, e.g., link reversal [13] and other approaches [19, 20]). As current routing protocols do not allow network operators to implement such stateful failover routing, our goal is to design protocols that correspond to a stateless, or *static*, failover routing.

Specifically, we consider a particularly simple and practical form of static failover routing: for each incoming link, a router maintains a destination-based routing table that maps the destination address of a packet and the set of non-failed ("active") links, to an output link. The router can locally detect which outgoing links are down and forwards packets accordingly. One should note that maintaining such per-incoming-link destination-based routing tables is necessary; not only is destination-based routing unable to achieve robustness against even a single link failure [17], but it is even computationally hard to devise failover routing schemes that maximize the number of nodes that are protected [1, 5, 17, 23]. We only consider link failures, not router failures (which are not always detectable by neighboring routers, and so such fast failover techniques may not apply).

A failover routing algorithm is responsible for computing, for each node (vertex) of a network (graph), a *routing function* that *matches* an incoming packet to an outgoing edge. A set of such routing functions for each vertex guarantees *reachability* between a pair of vertices, u and v, for which there exists a connecting path in the graph, if any packet directed to node v originated at node u is correctly routed from u to v.

We are interested in routing functions that rely solely on information that is locally available at a node (e.g., the set of non-failed edges, the incoming link along which the packet arrived, and any information stored in the header of the packet).

While it is known that every k-connected network cannot be partitioned by deleting at most $k-1$ links, it is not known whether any static "deterministic" routing (i.e., the outgoing port of each packet is always uniquely determined at a vertex v by its incoming link and the failed edges incident at v) achieves such resiliency.

On the other hand, routing based on random walks, i.e., choosing the outgoing link at random, achieves the best possible resilience as they will eventually deliver a packet to the destination as long as the network is connected. But, it comes with a huge cost. Namely, a random walk might traverse the whole network even when there is no single failed link. In fact, the expected delivery time of a packet would be as large as $\Theta(|V|^3)$ in some network topologies [6]. Furthermore, a random walk almost never reaches the destination following the shortest path. So, although extremely robust, when it comes to the time needed to deliver a packet to the destination, the behaviour of random walks is undesirable.

1.1 Our Results

In this paper, we show how randomness can be used to achieve $k-1$ resilient routing in k-connected networks while significantly outperforming random walks in terms of number of traversed nodes. Namely, we introduce **Randomized failover routing (`RND`)** in which outgoing edge is chosen for packets in a probabilistic manner based on the destination label, the incoming edge, and the set of non-failed edges. The randomized protocol that we present provides bound on the expected delivery time that gracefully grows with the number of actual link failures.

Our randomized routing functions provide delivery in case of any $k-1$ link failures for any k-connected graph. We achieve that by leveraging the standard decomposition of k-connected graphs into k arc-disjoint spanning arborescences $\mathcal{T}$ [9]. We also provide a bound on the expected number of hops that our algorithm performs, which is $O(Hk)$ for any $k-1$ failures and $O(H)$ for αk failures, where H is the length of the longest branch of any arborescence of $\mathcal{T}$ and $\alpha < 1$ is a constant. Furthermore, our routing functions are deterministic as long as the routing does not encounter any failure. Hence, packets belonging to the same logical connection are routed along the same path, minimizing reordering complexity at the receiver side.

Motivated by the fact that one can protect against $k-1$ failures in k-connected graphs using randomness, we make the following general conjecture, whose proof eludes us despite much effort.

- **Conjecture:** For any k-connected graph, one can find deterministic failover routing functions that are robust to any $k-1$ failures.

1.2 Organization

The rest of the paper is organized as follows. Section 2 provides background on existing works. In Section 3 we introduce our routing model and formally state the STATIC-ROUTING-RESILIENCY problem. The summary of our routing techniques that are leveraged throughout the whole paper are presented in Section 4. In Section 5 we focus on studying the relation between arborescences our input graph decomposes into and failed links. Section 6 builds on Section 5 and is devoted to designing an algorithm that, for any k-connected graph, computes randomized routing functions that are robust to $k-1$ edge failures and have bounded expected delivery time.

2 Related Work

Past work [2, 28] (1) designed such routing functions with guaranteed robustness against *only* a single link/node failure [11, 12, 21, 27, 29, 31], (2) achieved robustness against $\lfloor \frac{k}{2} - 1 \rfloor$ edge failures for k-connected graphs [10], and (3) proved that it is impossible to be robust against any set of edge failures that does not partition the network [12].

Thanks to its flexibility and oblivious behavior, another line of study was motivated by randomization. Namely, some of the previous work developed randomized routing schemes, usually to directly or indirectly achieve low congestion and/or balance the network load. In particular, Busch et al. [7] use randomization to adjust packet priorities, which in turn allows them to control deflection of packets.

Valiant [26] proposed a randomized routing algorithm 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. [30] employed VLB to design fault-tolerant networks with guaranteed no congestion under few router or link failures. Greenberg et al. [15] adopt VLB to reduce volatility of traffic and failure pattern of their data centers. In [24], Shepherd et al. extend VLB in order to build cost-effective networks robust to changes in demand patterns.

Beraldi [3] presents a search protocol for mobile networks that is based on modified random walks, i.e. based on biased random walks with look-ahead. Motivated by the success of ant-colonies in their search for food, Günes et al.[16] studied ant algorithms, which in their heart rely on randomization, as an approach to designing on-demand ad-hoc routing algorithms.

Chiesa et al. [8] studied resilience under link failures in k-connected networks. They devise static routing schemes that are resilient under $k-1$ failures in the following regimes: (1) if the routers are allowed to use three bits in the packet header for read/write operation, or (2) if the network supports broadcasting. A building block of those schemes is the result that every k-connected graph contains k arc-disjoint arborescences rooted at the same vertex [9].

3 Model

We represent our network as an undirected multigraph $G = (V(G), E(G))$, where each router in the network is modeled by a vertex in $V(G)$ and each link between two routers is modeled by an undirected edge in the multiset $E(G)$. When it is clear from the context, we simply write V and E instead of $V(G)$ and $E(G)$. We denote an (undirected) edge between x and y by $\{x, y\}$. A graph is *k-edge-connected* if there exist k edge-disjoint paths between any pair of vertices of G.

Each vertex v routes packets according to a *routing 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 all are *locally* available at a vertex.

Since our focus is on per-destination routing functions, we assume that there exists a unique destination $d \in V$ to which every other vertex wishes to send packets and, therefore, that the destination label is not included in the header of a packet. Forwarding *actions* consist in routing packets through an outgoing edge, rewriting some bits in the packet header, and creating duplicates of a packet.

In this paper we consider *randomized* routing functions, in which a vertex forwards a packet through an outgoing edge with a probability based only on the incoming port and the set of active outgoing edges. We present the formal definitions of the randomized routing model in Section 6.

The STATIC-ROUTING-RESILIENCY (SRR) problem. Given a graph G, a routing function f is *k-resilient* if, for each vertex $v \in V$, a packet originated at v and routed according to f reaches its destination d as long as at most k edges fail and there still exists a path between v and d. The input of the SRR problem is a graph G, a destination $d \in V(G)$, and an integer $k > 0$, and the goal is to compute a set of resilient routing functions that is k-resilient.

4 General Routing Techniques and Randomized Algorithm

Definition and notation. We denote a directed arc from x to y by (x, y) and by $\vec{G}$ the directed copy of G, i.e. a directed graph such that $V(\vec{G}) = V$ and $\{x, y\} \in E$ if and only if $(x, y), (y, x) \in E(\vec{G})$. A subgraph T of $\vec{G}$ is an *r-rooted arborescence* of $\vec{G}$ if (i) $r \in V$, (ii) $V(T) \subseteq V$, (iii) r is the only vertex without outgoing arcs and (iv), for each $v \in V(T) \setminus \{r\}$, there exists a single directed path from v to r that only traverses vertices in $V(T)$. If $V(T) = V$, we say that T is a r-rooted *spanning* arborescence of $\vec{G}$. When it is clear from the context, we use the word "arborescence" to refer to a d-rooted spanning arborescence, where d is the destination vertex. We say that two arborescences T_1 and T_2 are *arc-disjoint* if $(x, y) \in E(T_1) \implies (x, y) \notin E(T_2)$. A set of l arborescences $\{T_1, \ldots, T_l\}$ is *arc-disjoint* if the arborescences are pairwise arc-disjoint. We say that two arc-disjoint arborescences T_1 and T_2 do not *share* an edge $\{x, y\} \in E$ if $(x, y) \in E(T_1) \implies (y, x) \notin E(T_2)$.

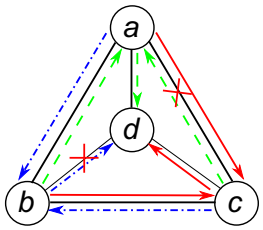


Figure 1 A 3-connected graph with 3 arc-disjoint arborescences colored red, blue, and green

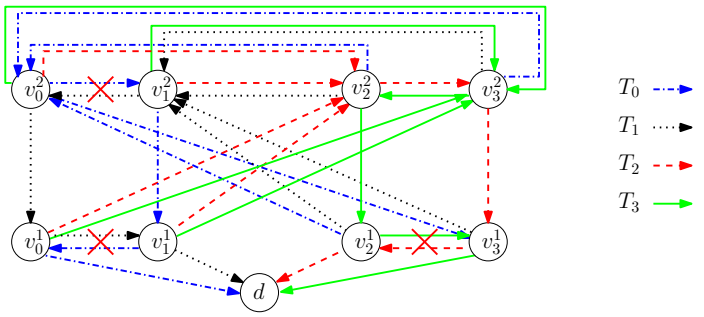


Figure 2 Graph used in the proof of Theorem 7 for $k = 2$.

For example, consider Fig. 1, in which each pair of nodes is connected by an edge (ignore the red crosses) and three arc-disjoint (d-rooted spanning) arborescences `Red`, `Green`, and `Blue` are depicted by colored arrows.

Arborescence-based routing. Throughout the paper, unless specified otherwise, we let $\mathcal{T} = \{T_1, \ldots, T_k\}$ denote a set of k d-rooted arc-disjoint spanning arborescences of $\vec{G}$. All our routing techniques are based on a decomposition of $\vec{G}$ into $\mathcal{T}$. The existence of k arc-disjoint arborescences in any k-connected graph was proven in [9], while fast algorithms to compute such arborescences can be found in [4]. 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 the 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 routing 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 routing 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 some available arborescence**, e.g., reroute along T', where T' is chosen randomly from distribution that we define in the sequel. Observe that, if the outgoing arc belonging to T' failed, we randomly pick another arborescence T'', and so on.

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

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_i and T_j be the two arborescences and let they share the same failed edge a. Furthermore, let a be the only failed edge T_i and T_j contain. If a packet hits a while routed along T_i or T_j, 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, as we discuss in the sequel.

From a different point of view, bouncing is a way of recycling arborescences that contain one failed link. This observation is crucial to obtain an efficient and a simple randomized $(k-1)$-resilient routing scheme, which we are now ready to present. The algorithm is parametrized by q that we define later.

Algorithm 1 Definition of RAND-BOUNCING-ALGO.

RAND-BOUNCING-ALGO: Given $\mathcal{T} = \{T_1, \ldots, T_k\}$

1. $T :=$ an arborescence from $\mathcal{T}$ 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 q, replace T by an arborescence from $\mathcal{T}$ sampled u.a.r.
      (b) Otherwise, bounce the failed edge and update T correspondingly

In the following sections we first study the connection between arborescences of $\mathcal{T}$ and failed links, and show how a part of their intricate interaction can be represented in a simple and an elegant way via, so-called, *meta-graph*. Afterwards, we show the RAND-BOUNCING-ALGO is $(k-1)$-resilient and we analyze its efficiency.

5 Meta-graph, Good Arcs, and Good Arborescences

The goal of this section is to provide an understanding of the structural relation between the arborescences of $\mathcal{T}$ when the underlying k-connected network has at most $k-1$ failed edges. The perspective that we are building here drives the construction of our randomized algorithm.

We start by introducing the notion of a *meta-graph*. To that end, we fix an arbitrary set of failed edges F. Throughout the section, we assume $|F| < k$, and define $f := |F|$. Then, we define a meta-graph $H_F = (V_F, E_F)$ as follows:

- $V_F = \{1, \ldots, k\}$, where vertex i is a representative of arborescence T_i.
- For each failed edge $e \in E$ belonging to at least one arborescences of $\mathcal{T}$ we define the corresponding edge e_F in H_F in the following way:
  - $e_F := \{i, j\}$, if e belongs to two different arborescences T_i and T_j;
  - $e_F := \{i, i\}$, i.e. e_F is a self-loop, if e belongs to a single arborescence T_i only.

Note that in our construction H_F might contain parallel edges. Intuitively, the meta-graph represents a relation between arborescences of $\mathcal{T}$ for a fixed set of failed edges. We provide the following lemma as the first step towards understanding the structure of H_F.

▶ **Lemma 1.** *The set of connected components of H_F contains at least $k - f$ trees.*

Proof. We give a proof by contradiction. To that end, assume that the set of connected components of H_F, denoted by $\mathcal{C}$, contains at most $k - f - 1$ trees. Now, if $C \in \mathcal{C}$ is a tree, we have $|E(C)| = |V(C)| - 1$, and $|E(C)| \geq |V(C)|$ otherwise. We also have

$$\begin{aligned} \sum_{C \in \mathcal{C}} |E(C)| &= \sum_{C \in \mathcal{C} \text{ is not a tree}} |E(C)| + \sum_{C \in \mathcal{C} \text{ is a tree}} |E(C)| \\ &\geq \sum_{C \in \mathcal{C} \text{ is not a tree}} |V(C)| + \sum_{C \in \mathcal{C} \text{ is a tree}} (|V(C)| - 1). \end{aligned} \tag{1}$$

Next, following our assumption that $\mathcal{C}$ contains at most $k - f - 1$ trees, from (1) we obtain

$$\sum_{C \in \mathcal{C}} |E(C)| \geq \sum_{C \in \mathcal{C}} |V(C)| - (k - f - 1). \tag{2}$$

Furthermore, as by the construction we have $\sum_{C \in \mathcal{C}} |V(C)| = |V_F| = k$, (2) implies

$$\sum_{C \in \mathcal{C}} |E(C)| \geq |V_F| - (k - f - 1) = f + 1. \tag{3}$$

On the other hand, from the construction of H_F we have

$$\sum_{C \in \mathcal{C}} |E(C)| = f,$$

which leads to a contradiction with (3). ◀

Lemma 1 implies that the fewer failed edges there are, the larger fraction of connected components of the meta-graph H_F are trees. Note that an isolated vertex is a tree as well.

In the sequel, we show that each tree-component of H_F contains at least one vertex corresponding to an arborescence from which any bounce on a failed edge leads to the destination d without hitting any new failed edge. To that end, we introduce the notion of good arcs and good arborescences. We say that an arc (u, v) is a *good arc* of an arborescence T if on the (unique) v-d path in T there is no failed edge. Let $a = (i, j)$, for $i \neq j$, be an arc of $\vec{H_F}$, $\{u, v\}$ be the edge that corresponds to a, and w.l.o.g. assume (u, v) is an arc of T_j. Then, we say a is a *well-bouncing* arc if (u, v) is a good arc of T_j. Intuitively, a well-bouncing arc (i, j) of $\vec{H_F}$ means that by bouncing from T_i to T_j on the failed edge $\{v, u\}$ the packet will reach d via routing along T_j without any further interruption. Finally, we say that an arborescence T_i is a *good arborescence* if every outgoing arc of vertex $i \in V_F$ is well-bouncing.

▶ **Lemma 2.** *Let T be a tree-component of H_F s.t. $|V(T)| > 1$. Then, $\vec{T}$ contains at least $|V(T)|$ well-bouncing arcs.*

Proof. Let T_i be an arborescence of $\mathcal{T}$ such that $i \in V(T)$. Then, by the construction of H_F we have that T_i contains a failed link. Next, a failed link closest to the root of T_i is a good arc of T_i. Therefore, for every $i \in V(T)$, we have that T_i contains an arc which is both good and failed. Furthermore, by the construction of H_F and the definition of well-bouncing arcs, we have that for every good, failed link of T_i there is the corresponding well-bouncing arc of $\vec{T}$. Also, observe that the construction of H_F implies that a well-bouncing arc corresponds to exactly one good-arc.

Now, putting all the observations together, we have that each T_i, for every $i \in V(T)$, has a good failed link which further corresponds to a well-bouncing arc of $\vec{T}$. As all the arborescences are arc-disjoint, and there are $|V(T)|$ many of them represented by the vertices of T, we have that $\vec{T}$ contains at least $|V(T)|$ well-bouncing arcs. ◀

Now, building on Lemma 2, we prove the following.

▶ **Lemma 3.** *Let T be a tree-component of H_F. Then, there is an arborescence T_i such that $i \in V(T)$ and T_i is good.*

Proof. Consider two cases: $|V(T)| = 1$, and $|V(T)| > 1$. In the case $|V(T)| = 1$, T is an isolated vertex which implies that it has no outgoing arcs. Therefore, T represents a good arborescence.

If $|V(T)| > 1$, then from Lemma 2 we have that $\vec{T}$ contains at most $2(|V(T)| - 1) - |V(T)| < |V(T)|$ arcs which are not well-bouncing. This implies that there is at least one vertex in T from which every outgoing arc is well-bouncing. ◀

Let us understand what this implies. Consider an arborescence T_i, and a routing of a packet along it. In addition, assume that the routing hits a failed edge e, such that e is shared with some other arborescence T_j. Now, if e corresponds to a well-bouncing arc of $\vec{H_F}$, then by bouncing on e and routing solely along T_j, the packet will reach d without any further interruption. Lemma 3 claims that for each tree-component T of H_F there always exists an arborescence T_i, with $i \in V(T)$, which is good, i.e. every failed edge of T_i corresponds to a well-bouncing arc of $\vec{H_F}$.

We can now state the main lemma of this section.

▶ **Lemma 4.** *If G contains at most $k-1$ failed edges, then $\mathcal{T}$ contains at least one good arborescence.*

Proof. We prove that there exists an arborescence T_i such that if a packet bounces on any failed edge of T_i it will reach d without any further interruption. Let F be the set of failed edges, at most $k-1$ of them. Then, by Lemma 1 we have that H_F contains at least $k - f \geq 1$ tree-components. Let T be one such component.

By Lemma 3, we have that there exists at least an arborescence T_i such that every outgoing arc from i is well-bouncing. Therefore, bouncing on any failed arc of T_i the packet will reach d without any further interruption. ◀

6 Randomized Routing via Good Arborescences

In this section, we show that a set of routing functions for G obtained by RAND-BOUNCING-ALGO is $(k-1)$-resilient. Note that our routing function (RND) 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.

The section is structured as follows. As a prelude, we show a simple, yet inefficient, randomized routing algorithm, called RAND-ALGO, that although is $(k-1)$-resilient, fails to achieve low expected number of hops in case of $k-1$ failed edges. We then apply our results from Section 5 to show that RAND-BOUNCING-ALGO is both $(k-1)$-resilient and requires up to an order fewer number of hops, compared to RAND-ALGO, to reach the destination.

6.1 A Simple (Inefficient) Randomized Routing

Consider the following naive randomized algorithm RAND-ALGO for routing along arborescences. A packet is routed along the same arborescence until it either reaches its destination or hits a failed edge. In the latter case, it is rerouted along another arborescences chosen uniformly at random. We show that there exists a k-connected graph and a set of failed edges such that the expected number of tree switches that RAND-ALGO makes is $\Omega(k^2)$. This further implies that the expected number of hops is $\Omega(Hk^2)$ in the worst case, where H is the length of a longest path in any arborescence and assuming that longest path in all the arborescences are up to a constant factor the same.

To prove the promised bound, we start by defining a $2k$ edge connected graph $G = (V, E)$ and its set of $2k$ arc disjoint spanning trees $T_0, \ldots, T_{2k-1}$ as follows.

- Set V consists of a destination vertex d and $4k$ additional vertices arranged into two equal-sized layers $L_1 = \{v_0^1, \ldots, v_{2k-1}^1\}$ and $L_2 = \{v_0^2, \ldots, v_{2k-1}^2\}$.
- Set E is defined by the following four subgraphs: (1) L_2 is a clique of size $2k$; (2) (L_1, L_2) is a complete bipartite graph; (3) for each $k = 0, \ldots, k-1$, there is an edge (v_{2i}^1, v_{2i+1}^1) and (4) vertex d is connected to each vertex of L_1. There is no other edge included in G.

Next, we construct $2k$ arc-disjoint spanning trees $T_0, \ldots, T_{2k-1}$ (see Fig. 2 for an example with $k = 2$). We use $[t]_0$ to denote set $\{0, 1, 2, \ldots, t-1\}$.

- For each $i \in [k]_0$, add the following arcs:
  - (v_{2i+1}^2, v_{2i}^2), (v_{2i}^2, v_{2i}^1), (v_{2i}^1, v_{2i+1}^1), and (v_{2i+1}^1, d) into T_{2i+1};
  - arcs (v_{2i}^2, v_{2i+1}^2), (v_{2i+1}^2, v_{2i+1}^1), (v_{2i+1}^1, v_{2i}^1), and (v_{2i}^1, d) into T_{2i}.
- For each $i \in [k]_0$, and for each $j \in [2k]_0 \setminus \{2i, 2i+1\}$, add the following arcs:
  - (v_j^2, v_{2i}^2), and (v_j^1, v_{2i}^2) into T_{2i};
  - (v_j^2, v_{2i+1}^2), and (v_j^1, v_{2i+1}^2) into T_{2i+1}.

Finally, consider a scenario in which edges (v_0^2, v_1^2), (v_2^2, v_3^2), $\ldots$, (v_{2k-4}^2, v_{2k-3}^2) and (v_0^1, v_1^1), (v_2^1, v_3^1), $\ldots$, (v_{2k-4}^1, v_{2k-3}^1), (v_{2k-2}^1, v_{2k-1}^1) failed.

We say that a packet is routed *downwards* (*upwards*) if it is routed from a vertex in L_2 (L_1) to a vertex in L_1 (L_2). Let E_d be the expected number, minimized over all the vertices, of tree switches of a packet that is routed downwards, E_u be the expected number of tree switches of a packet that is routed along T_i and is currently located at v_i^2, for some $i \in [2k-2]_0$, and E_2 be the expected number of tree switches of a packet that is originated by a vertex in L_2. Then, we can show.

▶ **Lemma 5.** *It holds* $E_u \geq \frac{3}{2k-1} E_d + \frac{2k-4}{2k-1} E_u + 1$.

Proof. Let p be routed along T_h and located at v_h^2, for some $h \in [2k-2]_0$. W.l.o.g, let $h = 0$. By the construction of T_0, from v_0^1 packet p should be forwarded to v_1^2 but (v_0^2, v_1^2) has failed. So, from v_0^2, p is forwarded downwards along T_1, T_{2k-2} or T_{2k-1} with probability $\frac{3}{2k-1}$ and routed along any other tree T_j to a vertex v_j^2 in L_2 with probability at least $\frac{2k-4}{2k-1}$. Hence, the lemma follows. ◀

▶ **Lemma 6.** *We have* $E_d \in \Omega(k^2)$.

Proof. By the construction, a packet routed downwards traverses arc (v_i^2, v_i^1) of T_i. W.l.o.g, let p be routed along (v_{2i}^2, v_{2i}^1) of T_{2i}. As (v_{2i}^1, v_{2i+1}^1), which belongs to T_{2i}, has failed, p is rerouted along T_j for some $j \in [2k]_0 \setminus \{2i\}$. Among them, only T_{2i+1} has a path from v_{2i}^1 to d that does not contain any failed link. T_{2i+1} is chosen with probability $\frac{1}{2k-1}$.
If any other tree T_j is chosen except T_{2k-2} and T_{2k-1}, which happens with probability $\frac{2k-4}{2k-1}$, then p is rerouted through T_j from v_{2i}^1 to a vertex v_j^2 in L_2, and hence

$$E_d \geq \frac{2k-4}{2k-1} E_u + 1. \tag{4}$$

Putting together with (4) and Lemma 5 we obtain $E_d \in \Omega(k^2)$. ◀

We finally observe that any packet originated at a vertex of L_2 is routed downwards at least once before reaching the destination vertex, i.e., $E_2 \geq E_d = \Omega(k^2)$, which proves the following theorem.

▶ **Theorem 7.** *For any $k > 0$, there exists a $2k$ edge-connected graph, a set of $2k$ arc-disjoint spanning trees, and a set of $2k-1$ failed edges, such that the expected number of tree switches with* RAND-ALGO *is* $\Omega(k^2)$. [1]

6.2 Correctness of Randomized-Bouncing Routing

In this section we prove that RAND-BOUNCING-ALGO eventually delivers a packet to d, i.e. it avoids loops, and in the next section we analyze its efficiency.
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 this likelihood. Notice that RAND-BOUNCING-ALGO implements exactly this approach. As an example, consider Fig. 1. If a packet originated at a is first routed through `Red` and the corresponding outgoing edge $\{a, c\}$ is

[1] We note that there exists a more involved example for which RAND-ALGO makes $\Omega(nk^2)$ hops, in expectation, to deliver a packet to d.

failed, then the packet is forwarded with probability q to `Blue` or `Green` chosen u.a.r., and with probability $1-q$ it is bounced to `Green`, which shares the outgoing failed edge $\{a, c\}$ with `Red`. By the following lemma we show that this approach leads to $(k-1)$-resilient routing.

▶ **Lemma 8.** RAND-BOUNCING-ALGO *produces a set of* $(k-1)$*-resilient routing functions.*

Proof. By Lemma 4 we have that there exists at least one arborescence T_i of $\mathcal{T}$ such that bouncing on any failed edge of T_i the packet will reach d without any further interruption. Now, as on a failed edge algorithm RAND-BOUNCING-ALGO will switch to T_i with positive probability, and on a failed edge of T_i the algorithm will bounce with positive probability, we have that the algorithm will eventually reach d. ◀

6.3 Number of Switches of RAND-BOUNCING-ALGO

In this subsection we analyze the expected number of times I the packet is rerouted from one arborescence to another one in RAND-BOUNCING-ALGO. As we are interested in providing an upper bound on I, we make the following assumptions. First, we assume that bouncing from an arborescence which is not good the routing always bounces to an arborescence which is not good as well. Second, we assume that only by bouncing from a good arborescence the routing will reach d without switching to any other arborescence. Third, we assume that there are exactly $k-f$ good arborescences, which is the lower bound provided by Lemma 1 and Lemma 3. Clearly, these assumptions can only lead to an increased number of iterations compared to the real case. Finally, for the sake of brevity we define $t := \frac{f}{k}$.

Now, we are ready to start with the analysis. As the first step we define a random variable, where in the definitions T is the arborescence variable from algorithm RAND-BOUNCING-ALGO,

$$X \quad := \quad \text{number of times a failed edge is hit before reaching } d \text{ if routing on } T$$

Let T_{init} be the first arborescence that we consider in RAND-BOUNCING-ALGO. Then, $\mathbb{E}[I]$ is upper-bounded by

$$\mathbb{E}[I] \leq \Pr[T_{init} \text{ is not good}]\,\mathbb{E}[X|T_{init} \text{ is not good}] + \Pr[T_{init} \text{ is good}]\,\mathbb{E}[X|T_{init} \text{ is good}], \quad (5)$$

where from our assumptions we have

$$\Pr[T_{init} \text{ is not good}] = t, \text{ and } \Pr[T_{init} \text{ is good}] = 1-t.$$

To simplify calculations, let X_P and Y_P be *pessimistic* upper bound on conditional expected values. That is, let X_P be the same as $\mathbb{E}[X|T_{init} \text{ is not good}]$ and Y_P as $\mathbb{E}[X|T_{init} \text{ is good}]$ under assumption that: the packet always hits a failed edge unless it bounces on a good arborescence; and, whenever packet bounces on a non-good arborescence it switches to a non-good one.

Now, let us express X_P and Y_P as functions in X_P, Y_P, q, and t, while following our assumptions. If T is not a good arborescence, then a routing along T will hit a failed edge. If it hits a failed edge, with probability $1-q$ the routing will bounce and switch to a non good arborescence. With probability qt the routing scheme will set T to be a non good arborescence, and with probability $q(1-t)$ it will set T to be a good arborescence. Formally, we have

$$X_P = 1 + qtX_P + q(1-t)Y_P + (1-q)X_P. \quad (6)$$

Applying an analogous reasoning about Y_P, we obtain

$$Y_P = 1 + qtX_P + q(1-t)Y_P. \quad (7)$$

Observe that the equations describing X_P and Y_P differ only in the term $(1-q)X_P$. This comes from the fact that bouncing on a good arborescences the packet will reach d without hitting any other failed edge.

By some simple calculations that we present in Appendix A, we obtain

$$\mathbb{E}[I] \leq U(q) := \frac{t}{(1-q)q(1-t)} + \frac{1}{1-q}. \tag{8}$$

Now we can prove the following lemma.

▶ **Lemma 9.** *We have that*

$$\mathbb{E}[I] \leq 2 + 4\frac{t}{1-t} = 2 + 4\frac{f}{k-f}.$$

Proof. From (8) we have $\mathbb{E}[I] \leq U(q)$. Setting $q = 1/2$ we obtain

$$U(1/2) \leq 2 + 4\frac{t}{1-t},$$

and by plugging $t = f/k$ the lemma follows. ◀

Note that if we know f in advance, or have some guarantee in terms of an upper bound on f, we can derive parameter q that improves the running time of RAND-BOUNCING-ALGO, as provided by the following lemma.

▶ **Lemma 10.** *$U(q)$ is minimized for $q = q^* := 1 - (1+\sqrt{t})^{-1}$, and equal to*

$$U(q^*) = \frac{1+\sqrt{t}}{1-\sqrt{t}}. \tag{9}$$

Proof. Consider $U(q)'$, which is

$$U(q)' = \frac{t(1-q)^2 - q^2}{(1-q)^2 q^2 (t-1)}.$$

In order to find the value of q that minimizes $U(q)$, denote it by q^*, we find the roots of $U(q)' = 0$ with respect to q. There is only one positive solution of equation $U(q)' = 0$, which is also the minimizer q^*, and is equal to

$$q^* = 1 - \frac{1}{1+\sqrt{t}},$$

as desired.

Finally, substituting q^* into (8) and simplifying the expression we obtain (9). ◀

Observe that

$$U(q^*) \leq \frac{4}{1-\frac{f}{k}}.$$

Therefore, if $f = \alpha k$, i.e., only a fraction of the edges fail, we obtain $U(q^*) \leq \frac{4}{1-\alpha}$. This means that the expected number of arborescence switches does not depend on the number of failed edges but on the ratio between this number and the connectivity of the graph. Otherwise, if $f = k-1$, we have that the expected number of arborescence switches is bounded by $4k$, which is linear w.r.t. to the connectivity of the graph. Combining these conclusions with Lemma 8, we obtain the following.

▶ **Theorem 11.** *Given a k-connected graph G, destination d and a decomposition of G into k arc-disjoint arborescences $\mathcal{T}$ rooted at d, there exists a $(k-1)$-resilient algorithm that delivers a packet to d after $O\left(\frac{k}{k-f}H\right)$ hops in expectation, where H is the length of a longest path of any arborescence of $\mathcal{T}$ 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.*

6.4 An Extension : Rerouting in a Non-uniform Manner

In this section we briefly study non-uniform choice of arborescence used for rerouting in algorithm RAND-BOUNCING-ALGO. To motivate that discussion, consider a scenario in which a packet hits a failed edge u, v while routed along arborescence T. Wlog, assume $T = T_k$. Furthermore, assume that path v-d along every other arborescence does not contain any failed link. Therefore, switching from T_k to any other arborescence the packet will reach d without any further interruption. If the packet is rerouted at step 2.2.(a) of algorithm RAND-BOUNCING-ALGO but not bounced, then the rerouting tree is chosen uniformly at random. It further means that the expected number of edges the packet will traverse before reaching d from v is

$$E_U = \sum_{i=1}^{k-1} \frac{dist_{T_i}(v)}{k-1} = \frac{\sum_{i=1}^{k-1} dist_{T_i}(v)}{k-1},$$

where $dist_{T_i}(a)$ is the number of the edges on the unique path from a to d along arborescence T_i.[2] However, the distances from v to d along different arborescences might significantly differ. This naturally suggests us to consider a non-uniform distribution of arborescences chosen at step 2.2.(a) of RAND-BOUNCING-ALGO, as we do in the rest of this section.

For each vertex $v \neq d$ and each arborescence T_i define probability p_v^i as

$$p_v^i := \frac{\frac{1}{dist_{T_i}(v)}}{\sum_{j=1}^{k-1} \frac{1}{dist_{T_j}(v)}}.$$

The expected number of the edges the packet will traverse if each arborescence is chosen with respect to the distribution given by p_v is

$$E_{NU} = \sum_{i=1}^{k-1} p_v^i dist_{T_i}(v) = \sum_{i=1}^{k-1} \frac{1}{\sum_{j=1}^{k-1} \frac{1}{dist_{T_j}(v)}} = \frac{k-1}{\sum_{i=1}^{k-1} \frac{1}{dist_{T_i}(v)}}.$$

Now we would like to show that indeed $E_U \overset{?}{\geq} E_{NU}$. But, it is the same as showing that

$$(k-1)^2 \overset{?}{\leq} \sum_{i=1}^{k-1} dist_{T_i}(v) \sum_{i=1}^{k-1} \frac{1}{dist_{T_i}(v)}.$$

However, the latter follows from Cauchy–Schwarz inequality as

$$(k-1)^2 = \left(\sum_{i=1}^{k-1} \sqrt{dist_{T_i}(v)} \sqrt{\frac{1}{dist_{T_i}(v)}} \right)^2 \leq \sum_{i=1}^{k-1} dist_{T_i}(v) \sum_{i=1}^{k-1} \frac{1}{dist_{T_i}(v)}.$$

Hence, $E_U \geq E_{NU}$, as advertised.

We note that this example is a potential scenario that might occur. However, and unfortunately, in case of failures we are unable to detect whether the described situation has occurred or not. Nevertheless, we believe that in practical applications the non-uniform choice of arborescences used for rerouting, as described above, would result in a more efficient routing than its uniform counterpart.

---

[2] As a remark, the best option in this scenario would be to reroute the packet along the arborescence T_i such that $i = \arg\min_i dist_{T_i}(v)$. Unfortunately, our model does not provide the information whether there is any failed edge on path v-d along T_i or not.

References

1 A. Atlas and A. Zinin. Basic Specification for IP Fast Reroute: Loop-Free Alternates. *IETF, RFC 5286*.

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

3 Roberto Beraldi. Biased random walks in uniform wireless networks. *Mobile Computing, IEEE Transactions on*, 8(4):500–513, 2009.

4 Anand Bhalgat, Ramesh Hariharan, Telikepalli Kavitha, and Debmalya Panigrahi. Fast Edge Splitting and Edmonds' Arborescence Construction for Unweighted Graphs. In *Proc. SODA*, pages 455–464, 2008.

5 Michael Borokhovich and Stefan Schmid. How (Not) to Shoot in Your Foot with SDN Local Fast Failover - A Load-Connectivity Tradeoff. In *OPODIS*, pages 68–82, 2013.

6 Graham Brightwell and Peter Winkler. Maximum hitting time for random walks on graphs. *Random Struct. Algorithms*, 1(3):263–276, October 1990.

7 Costas Busch, Maurice Herlihy, and Roger Wattenhofer. Randomized greedy hot-potato routing. In *SODA*, pages 458–466, 2000.

8 Marco Chiesa, Ilya Nikolaevskiy, Slobodan Mitrović, Aurojit Panda, Andrei Gurtov, Aleksander Mądry, Michael Schapira, and Scott Shenker. The quest for resilient (static) forwarding tables. In *International Conference on Computer Communications (INFOCOM), 2016 IEEE*. IEEE, 2016.

9 Jack Edmonds. Edge-disjoint branchings. *Combinatorial Algorithms*, pages 91–96.

10 Theodore Elhourani, Abishek Gopalan, and Srinivasan Ramasubramanian. IP Fast Rerouting for Multi-Link Failures. In *Proc. IEEE INFOCOM*, pages 2148–2156, 2014.

11 Gábor Enyedi, Gábor Rétvári, and Tibor Cinkler. A Novel Loop-free IP Fast Reroute Algorithm. In *Proc. EUNICE*, pages 111–119. Springer-Verlag, 2007.

12 Joan Feigenbaum, P. Brighten Godfrey, Aurojit Panda, Michael Schapira, Scott Shenker, and Ankit Singla. On the resilience of routing tables. In *Brief announcement PODC*, July 2012.

13 Eli M. Gafni and Dimitri P. Bertsekas. Distributed algorithms for generating loop-free routes in networks with frequently changing topology. *IEEE Transactions on Communications*, 1981.

14 Phillipa Gill, Navendu Jain, and Nachiappan Nagappan. Understanding network failures in data centers: Measurement, analysis, and implications. *SIGCOMM Comput. Commun. Rev.*, 41(4):350–361, August 2011.

15 Albert Greenberg, James R Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: a scalable and flexible data center network. In *ACM SIGCOMM computer communication review*, volume 39, pages 51–62. ACM, 2009.

16 Mesut Günes, Martin Kähmer, and Imed Bouazizi. Ant-routing-algorithm (ara) for mobile multi-hop ad-hoc networks-new features and results. In *The second mediterranean workshop on ad-hoc networks*, 2003.

17 Kin-Wah Kwong, Lixin Gao, Roch Guérin, and Zhi-Li Zhang. On the Feasibility and Efficacy of Protection Routing in IP Networks. *IEEE/ACM Trans. Networking*, 19(5):1543–1556, October 2011.

18 K. Lakshminarayanan, M. Caesar, M. Rangan, T. Anderson, S. Shenker, and I. Stoica. Achieving convergence-free routing using failure-carrying packets. In *SIGCOMM*, 2007.

19 Junda Liu, Aurojit Panda, Ankit Singla, Brighten Godfrey, Michael Schapira, and Scott Shenker. Ensuring Connectivity via Data Plane Mechanisms. In *Proc. of NSDI*, pages 113–126, 2013.

20 Junda Liu, Baohua Yan, Scott Shenker, and Michael Schapira. Data-driven Network Connectivity. In *Proc. of HotNets*, pages 8:1–8:6, New York, NY, USA, 2011. ACM.

21 Srihari Nelakuditi, Sanghwan Lee, Yinzhe Yu, Zhi-Li Zhang, and Chen-Nee Chuah. Fast Local Rerouting for Handling Transient Link Failures. *IEEE/ACM Trans. Networking*, 15(2):359–372, April 2007.

22 P. Pan, G. Swallow, and A. Atlas. RFC 4090 Fast Reroute Extensions to RSVP-TE for LSP Tunnels. May 2005.

23 Gero Schollmeier, Joachim Charzinski, Andreas Kirstadter, Christoph Reichert, Karl J. Schrodi, Yuri Glickman, and Chris Winkler. Improving the Resilience in IP Networks. In *Proc. HPSR*, 2003.

24 FB Shepherd and PJ Winzer. Selective randomized load balancing and mesh networks with changing demands. *Journal of Optical Networking*, 5(5):320–339, 2006.

25 Brent Stephens, Alan L. Cox, and Scott Rixner. Plinko: Building Provably Resilient Forwarding Tables. In *Proc. of HotNets*, pages 26:1–26:7. ACM, 2013.

26 Leslie G. Valiant. A scheme for fast parallel communication. *SIAM journal on computing*, 11(2):350–361, 1982.

27 Junling Wang and Srihari Nelakuditi. IP Fast Reroute with Failure Inferencing. In *Proc. of SIGCOMM Workshop on Internet Network Management*, INM, pages 268–273, New York, NY, USA, 2007. ACM.

28 Baohua Yang, Junda Liu, Scott Shenker, Jun Li, and Kai Zheng. Keep Forwarding: Towards K-link Failure Resilient Routing. In *Proc. IEEE INFOCOM*, pages 1617–1625, 2014.

29 Baobao Zhang, Jianping Wu, and Jun Bi. RPFP: IP fast reroute with providing complete protection and without using tunnels. In *IWQoS*, pages 137–146, 2013.

30 Rui Zhang-Shen and Nick McKeown. Designing a fault-tolerant network using valiant load-balancing. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*. IEEE, 2008.

31 Zifei Zhong, Srihari Nelakuditi, Yinzhe Yu, Sanghwan Lee, Junling Wang, and Chen nee Chuah. Failure Inferencing based Fast Rerouting for Handling Transient Link and Node Failures. In *Proc. IEEE INFOCOM*, 2005.

A The Running Time of RAND-BOUNCING-ALGO

Calculations omitted from Section 6.3

Subtracting (6) from (7) we obtain

$$Y_P = qX_P. \tag{10}$$

Substituting (10) to (6) gives

$$X_P = \frac{1}{(1-q)q(1-t)}, \tag{11}$$

and therefore, from (10),

$$Y_P = \frac{1}{(1-q)(1-t)}. \tag{12}$$

Plugging (11) and (12) into (5), we obtain an upper bound on $\mathbb{E}[I]$

$$\begin{aligned} \mathbb{E}[I] &\leq t\frac{1}{(1-p)p(1-t)} + (1-t)\frac{1}{(1-p)(1-t)} \\ &= \frac{t}{(1-p)p(1-t)} + \frac{1}{1-p}. \end{aligned} \tag{13}$$

Let $U(q)$ denote the upper-bound provided by (13), i.e.

$$U(q) := \frac{t}{(1-q)q(1-t)} + \frac{1}{1-q}.$$