Local Fast Rerouting with Low Congestion: A Randomized Approach

Gregor Bankhamer, Robert Elsässer and Stefan Schmid

***Abstract*—Most modern communication networks include fast rerouting mechanisms, implemented entirely in the data plane, to quickly recover connectivity after link failures. By relying on *local* failure information only, these data plane mechanisms provide very fast reaction times, but at the same time introduce an algorithmic challenge in case of multiple link failures: failover routes need to be robust to additional but locally unknown failures downstream.**

This paper presents local fast rerouting algorithms which not only provide a high degree of resilience against multiple link failures, but also ensure a low congestion on the resulting failover paths. We consider a randomized approach and focus on networks which are highly connected before the failures occur. Our main contributions are three simple algorithms which come with provable guarantees and provide interesting resilience-load tradeoffs, significantly outperforming *any* deterministic fast rerouting algorithm with high probability.

***Index Terms*—Failover routing, randomized routing, resilience**

I. Introduction

EMERGING applications, e.g., in the context of industrial, tactile or 5G networks, come with stringent latency and dependability requirements. To meet such requirements, Fast Re-Route (FRR) mechanisms have been specified for many networks [1]–[4]: *local* failover mechanisms in the data plane which avoid the time-consuming advertisement and collection of failure information and re-computation of routes in the control plane [5], [6]. Rather, these mechanisms rely on a *pre-defined* logic, often implemented in terms of conditional failover rules [3]. For example, wide-area networks often use IP Fast Reroute [1] or MPLS [2] Fast Reroute to deal with failures on the data plane, the Border Gateway Protocol (BGP) uses on BGP-PIC [7] for quickly rerouting flows, many data centers use Equal Cost MultiPath (ECMP) [8] which provides automatic failover to another shortest path, and Software-Defined Networks (SDNs) provide FRR functionality in terms of OpenFlow fast-failover groups [3], among many others [4].

Manuscript received August 26, 2020; revised April 13, 2021, and December 23, 2021; accepted April 11, 2022; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor X. XXXXXXX. Date of publication June XX, XXXX; date of current version May 10, 2022.

Gregor Bankhamer and Robert Elsässer are affiliated with the Department of Computer Sciences of the University of Salzburg, Austria.

Stefan Schmid is with the Faculty of Computer Science of the University of Vienna, Austria and with TU Berlin, Germany.

We would like to thank our anonymous reviewers for their helpful remarks that helped us to improve our paper. This research was supported in part by the Vienna Science and Technology Fund (WWTF) under grant number ICT19-045, 2020-2024 (project WHATIF) and by the European Union's Horizon 2020 research and innovation programme under Grant Agreement no. 824115 (HiDALGO).

However, while FRR mechanisms are attractive and widely used to deal with single failures, they introduce an *algorithmic challenge* in the presence of *multiple* link failures, as they are common in large networks such as datacenter and Internet networks [9]–[11]: rerouting decisions need to be made based on *incomplete* information about the failure scenario, and in particular, about failures *downstream*. The problem becomes particularly challenging if the rerouted flows should not only preserve connectivity under failures but also a low *load*, an important criteria in practice: congested routes threaten dependability and indeed, congestion is a main concern of any traffic engineering algorithm.

Recently, a series of negative results have been obtained on what can be achieved using *deterministic* fast rerouting algorithms (e.g., [12], [13]). In particular, it has been shown that even on networks which are still highly-connected after failures, the congestion resulting from *any* deterministic local fast failover algorithm is bound to be high in the worst case, i.e., *polynomial* in the number of link failures [14], [15].

This paper initiates the study of *randomized* algorithms to provide high resiliency and low congestion at the same time. In particular, we show that using a randomized approach, the congestion can be reduced from polynomial to polylogarithmic, with high probability, hence breaking deterministic congestion lower bounds.

A. Model in a Nutshell

In a nutshell, we consider the fundamental problem of congestion-minimal fast rerouting on a complete undirected network $G = (V, E)$, where each pair of the n nodes (e.g., switches, routers, or hosts) is directly connected (i.e., the network forms a clique). Such complete networks are typically studied in the related work and can be seen as an approximation of highly-connected networks as they arise, e.g., in the context of datacenters.

The network links (henceforth called *edges*) of G are subject to multiple concurrent failures, determined by an adversary and the goal is to pre-define local failover rules for the different nodes V such that traffic is rerouted to the destination while balancing the network *load*. A failover rule is essentially a *match-action* forwarding rule which not only *matches* certain header fields of the arriving packet (e.g., the IP destination address), but which can also be conditioned on the link failures incident to a given node $v \in V$, thereby specifying for which packets the rule is triggered; the *action* part then defines to which link the packet needs to be forwarded accordingly. These rules are static, i.e., the routing table is not allowed to

be updated during the whole routing procedure. To asses the performance of our protocols, we revisit the challenging (and practically relevant [16], [17]) in-cast scenario where $n-1$ sources inject one indefinite flow each to a single destination d which is known to the adversary [14], [15], [18], [19]. In our empirical analyses we additionally consider the so called gravity model [20], and evaluate adaptions of our algorithms in the Clos fat-tree topology against state-of-the-art approaches.

B. The Deterministic Case Lower Bound

The authors of [14] showed that deterministic failover algorithms are bound to result in a high load even in case of an initially completely connected network which is still highly connected after the failures [14]. The proof has been generalized further by Pignolet et al. in [15]. More specifically, the authors showed that: (1) when only relying on destination-based failover rules (i.e., rules which can only match the IP destination of a packet), an adversary can always induce a load of $\Omega(\varphi)$ at some edge by cleverly failing φ edges; (2) when failover rules can also depend on the source address, an edge load of $\Omega(\sqrt{\varphi})$ can be achieved, when failing φ many edges.

When considering the node load only, this bound can be extended and accounts for further information that may be used by the routing rules. Particularly, if we require that some packet starting from node v takes the same path under the same set of underlying edge failures (i.e., the packets' paths are oblivious and may not change depending on the other traffic moving around the network), a node load of $\Omega(\sqrt{\varphi})$ can be generated by the adversary. Note that this extension allows for including the hop counter inside the routing rule without weakening the result of the lower bound.

C. Our Results

The main contribution of this paper are three randomized fast rerouting algorithms which not only provide a high resilience to multiple link failures but also an exponentially lower load than any possible deterministic algorithm.

We present three failover strategies. Assuming up to $\varphi = O(n)$ edge failures, the first algorithm ensures that a load of $O(\log n \log\log n)$ is not exceeded at most nodes, while the remaining $O(\mathrm{polylog}\, n)$[1] nodes reach a load of at most $O(\mathrm{polylog}\, n)$. As we consider randomized approaches, we require the above statement to hold *with high probability*[2]. The second approach we present reduces the edge failure resilience to $O(n/\log n)$, however it is purely *destination-based* and achieves a congestion of only $O(\log n \log\log n)$ at *any* node w.h.p. Finally, by assuming that the nodes do have access to $\mathrm{polylog}\, n$ bits of shared information, which are not known to the adversary, the node load can be reduced even further. That is, a maximum load of only $O(\sqrt{\log n})$ occurs at *any* node w.h.p. All three strategies ensure loop-freedom w.h.p. [21] and avoid packet reorderings (i.e., all packets of the same flow are forwarded along the same path).

[1]By $\mathrm{polylog}\, n$, we denote the family of functions in n which lie in $O(\log^p n)$ for any constant $p > 0$

[2]We use the well established notion of *with high probability*, or w.h.p. , to denote probability of at least $1 - n^{-\Omega(1)}$.

While our focus lies on complete networks, which constitute a major open problem in the literature today, we show how our first two protocols may be adapted to the widely used Clos datacenter topology [22], [23] (more precisely we consider the Clos topology with 3 layers sometimes also simply referred to as fat-tree topology). We then report on empirical insights obtained through simulations and compare our approaches to other state-of-the-art failover protocols [18], [24]. The extension of our protocols to general topologies remains an open problem. However, this may be possible with the help of network decompositions based on spanning arborescences: it is known that any k-connected graph can be spanned by k arc-disjoint arborescences [25] (such a decomposition can be computed efficiently [26]), which enables a loop-free resilient routing [27]. The idea is then to use our approach to balance flows across arborescences. Finally, note that our third protocol is mostly of theoretical interest: extending it to general topologies would require to compute a set of Hamilton cycles for each such topology, which is NP-complete [28].

D. Further Related Work

Link failures are the most common failures in communication networks [29]–[31] and it is well-known that ensuring connectivity via the control plane can be slow [12], [32], even if it is centralized [33] or based on link reversal [34], [35]; it may also introduce undesirable transient behavior, such as a high loop ratio [21]. Data plane based failover mechanisms which do not require table reconfigurations can be orders of magnitudes faster [12] but are algorithmically challenging as routing tables need to be precomputed without knowledge of failures. For a general overview on failover mechanisms in the data plane, we refer to the recent survey by Chiesa et al. [36], referencing over two hundred papers on the topic, several of them published at IEEE/ACM Transactions on Networking [15], [24], [37]–[47].

Except for one model, we are interested in fast failover mechanisms in the data plane which do not require the modification of packet headers: while the modification of packet headers can simplify ensuring connectivity (e.g., by carrying failure information in the header) [11], [48], packet header rewriting typically comes with overheads and may even be infeasible [12], [49], [50]. Furthermore, while several interesting heuristics have been proposed in the literature [33], we are concerned with mechanisms which come with formal (probabilistic) performance guarantees. In particular, we are interested in scenarios in which *multiple* links can fail simultaneously; that is, in addition to ensuring traditional properties such as a perfect protection ratio [51] (i.e., ensuring resilience against any single failure), we aim to preserve connectivity and low load even under a large number of failures.

Prior work already derived several fundamental results on the feasibility of preserving connectivity using deterministic local fast failover mechanisms, in the presence of multiple failures and on the routing level, in different settings. In particular, Feigenbaum et al. [12] proved that it is not possible to achieve a *perfect (static) resilience* in arbitrary networks using deterministic local fast rerouting and without header

rewriting: it is impossible to define failover rules such that connectivity is preserved on the routing level as long as the network is physically connected. These results were recently extended by Foerster et al., who derive more general negative and positive results, also considering planar graphs in more details [13]. In [24], Chiesa et al. conjecture [24], [52] that is at least always possible to deterministically achieve what they call *ideal resilience*: unlike perfect resilience which requires connectivity on the routing level as long as the underlying *arbitrary* network is connected, ideal resilience focuses on k-(edge-)connected networks and requires connectivity on the routing level as long as there are at most $k-1$ link failures. Today, it is still unknown whether this conjecture holds in general, however, at least it has been proved true for several special graph classes as well as for scenarios with at most $k/2$ failures [27]. Furthermore, Chiesa et al. [24], [52] showed that ideal resilience can be achieved using randomized algorithms, by routing along precomputed spanning arborescences and by switching to a random alternative arborescence when encountering a failure [26].

The papers discussed above primarily focus on preserving connectivity, and much less is known about the design of failover algorithms which also account for load. More specifically, while there exist several results on deterministic algorithms for the incast scenario considered in this paper [12], [13], [18], [53], we are the first to study randomized algorithms and we show that the resulting load can be significantly better than the deterministic lower bound. Motivated by our empirical results on the Clos fat-tree topology (cf. Section VI) described in this paper, in a follow-up conference submission we theoretically analyzed an adapted version of the *Intervals* protocol from Section IV, see [54].

Bibliographic note. A preliminary version (without all technical details) was presented at IEEE ICNP 2019 [19].

II. Model

We model the communication network as a complete undirected graph $G = (V, E)$ where the nodes V represent the switches or routers which need to be configured with static forwarding rules (in this paper sometimes also simply called routing rules) and where the links E can fail.

In particular, we consider three different models, i.e., types of (match-action) *rulesets*, which are of increasing power depending on the information on which the rules can depend (the match part) and the information which they can change in the packet header (the action part):

- **Destination address:** Rules can only match the destination address (e.g., the IP destination) of the packets. Packets cannot be modified.
- **Hop count:** In addition to the destination address, rules can match the hop count: how far the packets have travelled so far.
- **Hop count modification:** One of our algorithms (*Shared-Permutations* – see Section V) can additionally modify the hop count to arbitrary $O(\log n)$ bit values. Note that this rule is not needed for the first two algorithms we develop.

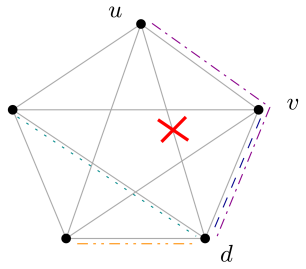

Fig. 1. All-to-one routing in the complete graph K_5. Each node sends one flow towards destination d, each corresponding to one of the colored lines in non-solid style. Because the link (u, d) is failed the flow of u needs to take a detour. This causes the edge (v, d) to accumulate a load of 2.

In addition to header information, the matching part of each above rule may also depend on the *local link failures*, the link failures incident to the given node v, but not on other remote failures which are not known at the configuration time. While the first model is the standard model used by routers and supported generally, the latter two models require software-defined switches or routers, e.g., based on OpenFlow [3].

We consider an adversarial model and assume that the link failures are chosen by an adversary. More specifically, we assume that the failover rules are generated by a randomized algorithm (or rely again on a hash function which matches the hop count), and that the adversary is *oblivious*: it knows the failover protocol including the used probability distributions, but not the generated random values nor the resulting loads. (Later in this paper we briefly discuss even stronger adversaries.)

In order to assess the performance of our protocols, we consider the *all-to-one* traffic pattern, in which each node $v \in V \setminus \{d\}$ sends a single *flow* towards some common destination d. For each node v, such a flow is defined as an indefinite sequence of packets with source v and destination d. The goal is to minimize the *load* of any link (or node) in the network, which is defined as the number of flows crossing this link (or node). In case such a flow hits a link multiple times the load of this edge is increased by 1 for each such hit. An example of the outcome of all-to-one routing in the K_5 graph is given in Fig. 1. While, w.h.p., our protocols avoid packets to visit a node more than once, it may happen that some flows travel in a (temporary) forwarding loops for a small amount of hops. We ensure that packets of a flow are always forwarded along the same path, hence avoiding packet reorderings.

III. Beating Deterministic Approaches with Three Permutations

This section presents our first failover algorithm. While it is simple as forwarding is only based on the destination and hopcount header fields, it ensures w.h.p. very low loads even under a large number of link failures.

From the point of view of some fixed node v the first protocol, we call it *3-Permutations*, works as follows. For destination d, the node v stores three permutations $\pi^{(1)}_{v,d}$, $\pi^{(2)}_{v,d}$ and $\pi^{(3)}_{v,d}$ of all nodes $u \in V \setminus \{d\}$. Each node chooses these three permutations uniformly at random. Upon receiving a

Input: A packet with destination d and hop count $h(p)$
1: **if** (v, d) is intact **then** forward p to d and **return**
2: **else** set i to $\arg\max_{j\in\{1,2,3\}}\{h(p) \geq (j-1)C_1\}$ and send p to first directly reachable node in $\pi^{(i)}_{v,d}$
3: $h(p) = h(p) + 1$

Fig. 2. *3-Permutations* protocol. Point-of-view of some node v

packet p intended for d, the node v first tries to forward it directly via the link (v, d). In case this link failed, v inspects the current hop counter of p, denoted by $h(p)$. Depending on $h(p)$, the node v then chooses one of the three permutations $\pi^{(i)}_{v,d}$ and forwards the packet to the first *reachable* partner w in this permutation. We call a node w *reachable* from v, if the direct link (v, w) is not failed. The criteria for selecting which permutation to use is simple. In case $h(p) < C_1$ for a value $C_1 = \Theta(\log n)$, permutation $\pi^{(1)}_{v,d}$ is consulted. For $C_1 \leq h(p) < 2C_1$ the permutation $\pi^{(2)}_{v,d}$ is used and in any remaining case $\pi^{(3)}_{v,d}$ is utilized. In any case the packets hop counter is increased by 1 before handing it to the next node. A concise description is given in Fig. 2.

Our main contribution is related to the way how these permutations are selected. Instead of opting for a deterministic protocol, we assume that each node v chooses the permutations $\pi^{(i)}_{v,d}$ out of all possible permutations of nodes $u \in V \setminus \{d\}$ uniformly and at random. As the adversary is *oblivious*, these permutations are not known to it and it needs to essentially blindly select edges for manipulation. Note however that this approach comes with a challenge. This random creation of failover routes may introduce temporary cycles into the packets routing paths. However, most of the packets p reach the destination d solely relaying on the failover entries given by the first permutation, $\pi^{(1)}_{v,d}$. And, only in case p ends up trapped in a cycle, further permutations are used to allow it to escape said cycle. We show that w.h.p., at most $O(\log^2 n \log\log n)$ load is accumulated at any node, even if the adversary is allowed to destroy a linear amount of edges.

Theorem 1. *Assume that the adversary fails at most $\alpha \cdot n$ edges where $\alpha < 1$ is a non-negative constant*[3]*. Then, if all nodes perform all-to-one routing to any destination d and follow the* 3-Permutations *protocol, a maximum of*

$$O(\log n \cdot \log\log n)$$

flows passes at all but $O(\log^2 n)$ nodes. Furthermore, all remaining nodes, except for d, receive a load of at most $O(\log^2 n \cdot \log\log n)$ and every packet travels $O(\log n)$ hops. These statements hold w.h.p.

In order for the nodes to follow this protocol they require the exact value of C_1. This value upper-bounds the number of hops needed by any packet to reach the destination d, unless it is trapped in a cycle due to the permutation $\pi^{(1)}_{v,d}$. We show in Lemma 7 that C_1 can be bounded from above by $16\log_{(1/\alpha)} n$. If α is not known to the nodes, then C_1 can be set to some value in $\omega(\log n)$. This slightly changes the result of Theorem 1 where up to $O(\log^2 n)$ many nodes receive a load of $O(C_1 \cdot \log n \log\log n)$.

The reason why we employ exactly three permutations per destination is as follows. Either a packet ends up in a forwarding loop when forwarded via $\pi^{(1)}_{v,d}$ or it will reach the destination within C_1 hops. The same is true for packets being forwarded via $\pi^{(2)}_{v,d}$ and $\pi^{(3)}_{v,d}$. For each such permutation, the probability that the packet ends in a forwarding loop is $\text{polylog}\, n/n$. Hence, only if the packet ends up in a forwarding loop for all three permutations, it will not reach the destination. The probability for this is $\text{polylog}\, n/n^3$ as the permutations are generated randomly. Because packets following our protocol take a different path depending on their destination and source node (roughly n^2 possibilities), we need to multiply this probability of a bad event by n^2. Hence, the probability that *any* packet does not reach its desired destination is $\text{polylog}\, /n$, which is a low probability event. When it comes to memory complexity, each node may store 3 permutations of n nodes for every destination d. A naive approach would therefore require routing tables of size $O(n^2 \log n)$ bits to be prepared for routing to any arbitrary destination d. This can be overcome as follows. First, each node v only computes 3 random permutations $\pi^{(i)}_v$, $i = 1, 2, 3$ on *all* nodes. The permutation $\pi^{(i)}_{v,d}$ for each $d \in V$ is simply $\pi^{(i)}_v$, and thus we apply $\pi^{(i)}_v$ to obtain our failover strategy regardless of d. Note that if the edge (v, d) is not failed, then any packet with target d that reaches v is sent directly from v to d. If, however, (v, d) is failed, then such a packet is sent to the first node w in $\pi^{(i)}_v$ for which (v, w) is not failed. Secondly, note that the nodes only consult their permutations up until the first reachable node (see Line 2 of Fig. 2). Even if all αn failed edges are incident to the same node v, then at least one of the first $3\log_{(1/\alpha)} n$ nodes in each of the permutations $\pi^{(i)}_{v,d}$ is directly reachable from v w.h.p. This follows from the fact that the adversary does not know the random bits generated at some node. Therefore, nodes may truncate the permutations, storing only the first $3\log_{(1/\alpha)} n$ entries of each permutation. In the low probability event that none of the first $3\log_{(1/\alpha)} n$ is directly reachable from v (due to the failed edges), another $3\log_{(1/\alpha)} n$ nodes are selected uniformly at random – without replacement. Employing these improvements yields an improved total memory complexity of $O(\log^2 n/\log(1/\alpha)) = O(\log^2 n)$ per node.

In the following, we consider some arbitrary but fixed node d as destination. As we establish probabilistic guarantees of at least $1 - n^{-(1+\Omega(1))}$ for the statements in Theorem 1 w.r.t. this fixed node, applying the union bound then implies that the results indeed hold for arbitrary destinations d w.h.p.

Regarding the tightness of our result, assume all $\alpha \cdot n$ failed edges are so called *destination edges*, i.e., edges incident to the destination. Then, the flow starting at the other end v of such a (failed) edge is first sent to a node selected uniformly at random from the set $V \setminus \{d\}$. The resulting distribution of the load can be seen as the outcome of throwing αn balls into $n - 1$ bins [55], and the maximum load immediately reaches $\Omega(\log n/\log\log n)$ at some node w.h.p.

[3]More specifically, α can be an arbitrary constant with $0 < \alpha < (n-1)/n$. Note that this upper-bound quickly tends towards 1 for large n.

A. Notation and Conventions

As we consider a fixed destination d we omit it from the indices of our previously defined notation. Additionally, for $i \in \{1,2,3\}$ and some integer $1 \leq j \leq n-1$, we denote by $\pi_v^{(i)}(j)$ the j-th node in v's i-th permutation.

Definition 1 (Inner/Destination Edges and Good/Bad Nodes)**.** We call each edge (v,d) with $v \in V$ a *destination edge* as it is incident to the destination d. All remaining edges are called *inner edges*. Furthermore, we call a node $v \in V \setminus \{d\}$ *good* if the destination edge (v,d) is not failed. Otherwise, we call v a *bad* node. By V_G and V_B we denote the sets of good and bad nodes, respectively.

The intuition behind calling such a node *good* is that it may directly forward packets to d when following the protocol in Fig. 2. Additionally, we define $\mathcal{F}$ to be the set of failed edges and further partition this set into $\mathcal{F}^{(in)}$ and $\mathcal{F}^{(d)}$. The former contains all failed inner edges, the latter the failed destination edges. We let ε and γ be constants such that $|\mathcal{F}^{(d)}| \leq \varepsilon \cdot n$ and $|\mathcal{F}^{(in)}| \leq \gamma \cdot n$, where $\varepsilon + \gamma \leq \alpha < 1$.

If we state that we apply Chernoff bounds for a random variable X, we mean the multiplicative variant $P[X \geq (1+\delta)\mu] \leq \exp\left(\min\{\delta, \delta^2\}\mu/3\right)$, where $\mu = E[X]$ and $\delta > 0$. For lower tails we use $P[X \leq (1-\delta)\mu] \leq \exp\left(\delta^2\mu/2\right)$ for $0 < \delta < 1$ (see e.g. [56]). Similar, if we say we apply *union bounds*, we mean Boole's inequality [56]. For a set of probabilistic events, this bound states that the probability of at least one event in this set occurring is no greater than the sum of the probabilities of the individual events. Besides w.h.p., we introduce the following abbreviations: Instead of *with probability*, *uniformly at random* and *random variable*, we use w.p., u.a.r. and r.v., respectively. We denote the binomial distribution with m trials and success probability p by $B(m,p)$. Finally, by $\log n$ we denote $\log_2 n$.

B. Analysis

We first establish some structural properties that describe the paths that the packets take according to our failover strategy. For $i \in \{1,2,3\}$, we define the directed subgraphs $G'^{(i)} = (V \setminus \{d\}, E'^{(i)})$ with edge sets $E'^{(i)} = \{(v, \pi_v^{(i)}(1)) \mid v \in V_B\}$. Under assumption that $\mathcal{F}^{(in)} = \emptyset$, this graph depicts the possible paths a packet traverses to either the good nodes V_G or to some possibly existing cycle. It is easy to see that any graph $G'^{(i)}$ hosts multiple trees, each rooted in some $w \in V_G$ as the nodes in V_G are the only ones with out-degree 0. The other components in $G'^{(i)}$ do not contain a node of V_G. Instead, they contain a cycle, in which each node on the cycle[4] is the root of a subtree (see component on the right of Fig. 3). The first important result of our analysis is that the size of these structures is at most $O(\log n \cdot \log\log n)$ w.h.p.

In the next step, we account for the failures in $\mathcal{F}^{(in)}$. Here we use the fact the permutations are chosen completely at random. We show that only $O(\log n)$ of all edges in the graphs $G'^{(i)}$ are failed, which reinforces the intuition that failing inner edges has little effect compared to the failure set $\mathcal{F}^{(d)}$. This approach allows us to construct the graphs $G''^{(i)}$, which now account for inner edge failures and correctly depict the paths that the packets traverse.

[4]Such a cycle may consists of a single node $v \in V_B$ if $E'^{(i)}$ contains the edge (v,v). This happens in case of $\pi_v^{(i)}(1) = v$.

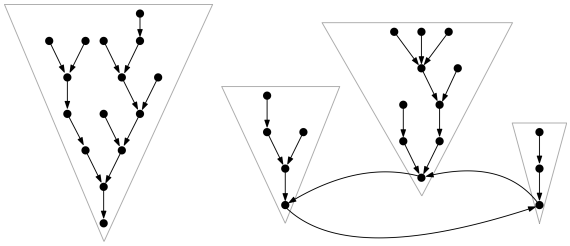

Fig. 3. The structures contained in the subgraph G'. On the left, a tree rooted in some $v \in V_G$ is presented. On the right, we have a cycle and each node of the cycle is again the root root of a tree.

Finally, we put everything together and use the graphs $G''^{(i)}$ to show the result of Theorem 1. At this point we also argue that 3 permutations per node do indeed suffice for *any* packet to be routed to d w.h.p.

a) Measuring Forests: As mentioned we start by analyzing the graphs $G'^{(i)}$. We first consider some fixed $i \in \{1,2,3\}$ and omit the superscript (i). That is, we consider the graph G' together with the edge set $E' = \{(v, \pi_v(1)) | v \in V_B\}$ and $V' = V \setminus \{d\}$. As already discussed, only the existence of some cycles between nodes in V_B prevents G' from being a forest. A rough perspective on G' is given in Fig. 3.

We start by establishing some structural properties of the graph G'.

Lemma 1. *The graph G' does not contain paths or cycles of length larger than $4\log_{1/\varepsilon} n$ w.p. $1 - n^{-3}$. Additionally, the number of cycles in G' is $O(\log n)$ w.p. $1 - n^{-3}$.*

Proof: Let $v \in V_B$ be an arbitrary node with $(v,d) \in \mathcal{F}^{(d)}$ and consider the edges $(u, \pi_u(1))$ of all $(1-\varepsilon)n$ nodes $u \in V_B$. Starting at v we uncover the outgoing edges one after the other and follow the resulting path until either a node $w \in V_G$ is reached or a cycle is created. This way, a path of at least $i+1$ distinct nodes is traversed w.p. at most

$$\frac{\varepsilon n - 1}{n-1} \cdot \frac{\varepsilon n - 2}{n-1} \cdot \ldots \cdot \frac{\varepsilon n - i}{n-1} < \varepsilon^i \left(1 + o(1)\right) < \varepsilon^{i+1}.$$

Here each quotient corresponds to the probability of continuing the path for one further step without hitting a node $u \in V_B$ that we already visited. Therefore, our fixed node v is at the beginning of a path of length at least $4\log_{1/\varepsilon} n$ with probability at most n^{-4}.
To bound the probability that *none* of the nodes in V_B are the origin of such a long path we apply union bounds. As $|V_B|$ such nodes exist, and each of these nodes has probability n^{-4} to form such a path, we conclude that this probability is at most $|V_B| \cdot n^{-4} < n \cdot n^{-4} = n^{-3}$. Note that this also upper bounds the length of the cycles, as one may see a cycle as a path that is concluded with a previously visited node.

To determine the number of cycles contained in G', consider again some node $v \in V_B$ and the path that is created by the process described above. Assuming that we are at the last node before the path terminates, there are two possible outcomes:

Selecting one of the w.h.p. at most $4\log_{1/\varepsilon} n$ already visited nodes and creating a cycle, or one of the $(1-\varepsilon)n$ nodes $v \in V_G$. Therefore, a cycle is created by v w.p. at most

$$\frac{4\log_{1/\varepsilon} n}{(1-\varepsilon)n + 4\log_{1/\varepsilon} n}.$$

Now we sequentialize this process for all $v \in V_B$ one after the other. Clearly there exist dependencies as it is possible that $v \in V_B$ is already contained in a path that was already uncovered before. Similarly, the path starting in v might hit the path of another node, which was uncovered earlier. Note however that this only decreases the probability for v to create a new cycle. Therefore w.p. $p = O(\log n/n)$ some fixed node $v \in V_B$ creates a new cycle regardless of the other nodes. Chernoff bounds immediately yield the desired result. ∎

Consider again the graph $G' = (V', E')$ with $V' = V \setminus \{d\}$ and some fixed node $w \in V_G$. Remember that such a node is the root of a tree in G', induced by the edges $(v, \pi_v(1))$ for $v \in V_B$. Let now L_i denote the set of nodes at level i of w's tree, where $L_0 = \{w\}$. We now construct $L_1, L_2, ...$ step by step as follows. In the i-th step construct $L_i = \{v \mid v \in (V_B \setminus \bigcup_{j=0}^{i-1} L_j) \wedge \pi_v(1) \in L_{i-1}\}$. One can see this as constructing the tree $w \in V_G$ layer-by-layer, by adding nodes with outgoing edges connected to nodes in the set L_i in the i-th step. We define the r.v. $X_i = |L_i|$ and when fixing $w \in V_G$ we call $\{X_i\}$ the *layer sequence*, or in short *sequence*, corresponding to v. The step-wise construction described above directly yields the following lemma.

Lemma 2. *Fix some root node $w \in V_G$ in G' together with its layer sequence $\{X_i\}$. Then, it holds for level i that $X_i \sim B(m,p)$ with*

$$m = |V_B| - \sum_{j=1}^{i-1} X_j \text{ and } p = \frac{X_i}{|V'| - \sum_{j=0}^{i-2} X_j}$$

Proof: Let $w \in V_G$ and consider the process of uncovering edges just as described in the paragraph before this lemma. Clearly it holds that $X_1 \sim B(|V_B|, 1/V')$ as the partner of every $v \in V_B$ is chosen u.a.r. due to the nature of the permutations π_v.

Assume now we are in the $(i-1)$-th step and already uncovered all edges between the sets $L_0, L_1, ..., L_{i-2}$. To construct L_{i-1} we need to consider edges $(v, \pi_v(1))$, where $\pi_v(1) \in L_{i-2}$ and $v \in V_B \setminus \bigcup_{j=0}^{i-2} L_j$. At this point $m := |V_B| - X_1 - X_2 - ... - X_{i-1}$ nodes still have uncovered outgoing edges. We know that these edges do not connect to nodes in the sets $L_0, ..., L_{i-2}$. Now, as the edge partners are chosen u.a.r the probability that such an edge connects to nodes in L_{i-1} is exactly $p := X_{i-1}/(|V'| - X_0 - X_1 - ... - X_{i-2})$. Note that there exist no dependencies between the selections of these m nodes. Hence, X_i follows $B(m,p)$. ∎

This means that we can describe the tree rooted in $w \in V_G$ by a sequence of binomial distributions, whose expected value depends on the previous layers. The following statement gives us a bound on $m \cdot p$ w.h.p., showing that the set of nodes at level i indeed decreases exponentially fast. The proof is given in the supplementary material and mostly relies on the statement of Lemma 2 in conjunction with Chernoff bound applications.

Lemma 3. *Let $w \in V_G$ be a root node in G' together with its corresponding layer sequence $\{X_i\}$. Then, there exists a constant $0 < \beta < 1$ such that for $i < \log^2 n$ it holds that $E[X_{i+1}] \leq X_i \cdot \varepsilon(1 + o(1)) \leq X_i \cdot \beta$. Additionally, there exists a constant $C > 0$ such that, w.p. $1 - n^{-3}$ it holds for all $i \leq \log^2 n$ that $X_i < C \log n$.*

According to Lemma 1 packets travel at most $O(\log n)$ hops until reaching the destination. This implies the following.

Corollary 1. *Consider* any *$w \in V_G$ with its corresponding layer sequence $\{X_i\}$ in G'. Then, for $i > C' \log n$ it holds that $X_i = 0$ w.p. at least $1 - n^{-3}$.*

Clearly for some fixed node $w \in V_G$ with sequence $\{X_i\}$, our main interest lies in the value $X = \sum_{i \geq 0} X_i$. In the following we say that $X_{i-1} < a$ *increases* into the interval $[a, b)$, iff $X_i \in [a, b)$. Analogously we say $X_{i-1} \geq b$ *decreases* into the same interval iff $X_i \in [a, b)$.

Lemma 4. *Consider again a root $w \in V_G$ and the corresponding layer sequence $\{X_i\}$. Then, for $j < \log(\log n / \log\log n)$ and a constant $\hat{\beta}$ with $\beta < \hat{\beta} < 1$ the following holds: at most $O(\hat{\beta}^{-j})$ members of $\{X_i\}$ increase into the interval*

$$\left[C \log n \cdot \hat{\beta}^j, C \log n \cdot \hat{\beta}^{j-1}\right)$$

w.p. at least $1 - n^{-3}$. Note that β and C are the constants defined in Lemma 3.

Proof: In the following we consider the elements of the sequence $\{X_i\}$ one after the other, starting with X_0. By Lemma 2 and Lemma 3 we know that the i-th value X_i follows a binomial distribution with mean less than $X_{i-1} \cdot \beta$. Using Chernoff bounds together with the fact that $\beta < 1$ is a constant, we obtain for any $t \geq 0$

$$Pr[X_i \geq t \mid X_{i-1} \leq t] \leq \exp(-\Omega(t)). \tag{1}$$

Note that for $t = C \log n \cdot \hat{\beta}^j$, (1) bounds the probability that X_{i-1} increases into $[C \log n \cdot \hat{\beta}^j, C \log n \cdot \hat{\beta}^{j-1})$. Now, from Corollary 1 it follows that at most $C' \log n$ elements may increase into the interval mentioned before. Therefore we can majorize the total number of increases into the interval by $B(C' \log n, \exp(-c \cdot t))$, where c is the constant hidden in $\Omega(t)$ in (1). Now, using the well-known upper bound $\binom{s}{t} \leq \left(\frac{e \cdot s}{t}\right)^t$ on the binomial coefficient (see e.g. Proposition B.2 of [57]) we get

$$Pr\left[B\left(C' \log n, \exp\left(-cC \cdot \log n \cdot \hat{\beta}^j\right)\right) = \frac{5}{cC} \cdot \hat{\beta}^{-j}\right] < \tag{2}$$

$$\left(O(1) \log n \cdot \hat{\beta}^j\right)^{O\left(\hat{\beta}^{-j}\right)} \left(\frac{1}{e}\right)^{5 \log n} < \frac{1}{n^{3.4}}$$

for n large enough and $j < \log(\log n / \log\log n)$. ∎

We are finally ready to state that no tree contained in G' consists of more than $O(\log n \cdot \log\log n)$ nodes.

Lemma 5. *Let $w \in V_G$ be a root and $\{X_i\}$ the corresponding layer sequence. Then it holds w.p. at least $1 - \text{polylog}\, n/n^3$ that $\sum_i X_i < O(\log n \cdot \log\log n)$*

Proof: We start by fixing some interval $J := [C \log n \cdot \hat{\beta}^j, C \log n \cdot \hat{\beta}^{j-1})$ for $j < \log(\log n / \log\log n)$ just as in Lemma 4. In the following we consider the so-called *cost* caused by this interval, i.e, $\sum_{\{i:\ X_i \in J\}} X_i$. Three possible events may cause some X_i to contribute to this sum:

1) X_{i-1} *increased* into the interval
2) X_{i-1} *decreased* into the interval
3) $X_{i-1} \in J$ and $X_i \in J$

Now, the first point is addressed by Lemma 4. We know that w.h.p. at most $O(\hat{\beta}^{-j})$ such values exist in total. As for the second point, assume that this event occurred for some X_{i-1}, i.e., assume X_{i-1} decreased into J. Clearly, for $X_{i'-1}$ with $i' > i$ to decrease into J again, there must be a $k \in i, \ldots, i'-2$ such that X_k increases into an interval with index $j' > j$. Using again Lemma 4 and applying the union bound over all intervals with $j < \log(\log n / \log\log n)$ we know that at most $O\left(\hat{\beta}^{-(j-1)} + \hat{\beta}^{-(j-2)} + ... + \hat{\beta}^{-1}\right) = O\left(\hat{\beta}^{-j}\right)$ such members exist.

The third point is the most interesting. Assume $X_{i-1} \in [C \log n \cdot \hat{\beta}^j, C \log n \cdot \hat{\beta}^{j-1})$, and observe that if $X_i \leq \hat{\beta} X_{i-1}$ it follows that $X_i \notin J$. Now, according to Lemma 3 we know that $E[X_i] \leq X_{i-1}\beta$. Noting that $\hat{\beta} > \beta$ we bound $Pr[X_i \geq X_{i-1} \cdot \hat{\beta}]$. Applying Chernoff bounds with $\delta = \hat{\beta}/\beta - 1 = \Omega(1)$, we get

$$\begin{aligned} Pr[X_i \geq X_{i-1}\hat{\beta}] &< \exp(-\Omega(X_{i-1})) \\ &< \exp\left(-\Omega(\log n \cdot \hat{\beta}^j)\right). \end{aligned}$$

As the whole sequence $\{X_i\}$ has length at most $C' \log n$ according to Corollary 1, any of the above events can happen at most $C' \log n$ times. A similar approach as in (2) therefore immediately yields that at most $O(\hat{\beta}^{-j})$ many times X_i stays in the interval J w.p. at least $1 - n^{-3}$.

Summarizing, the cost caused by elements taking values in J is at most $O(\hat{\beta}^{-j} \cdot C \log n \cdot \hat{\beta}^{j-1}) = O(\log n)$ w.p. $1 - n^{-3}$. When applying the union bound we obtain that the total cost generated by all intervals with $j \leq \log(\log n / \log\log n)$ is $O(\log n \cdot \log\log n)$. Note that for $j > \log(\log n / \log\log n)$ it holds that $C \log n \cdot \hat{\beta}^{j-1} = O(\log\log n)$. Even if the whole sequence $\{X_i\}$ remains in these remaining intervals for all $C' \log n$ steps, a cost of at most $O(\log n \cdot \log\log n)$ can be accumulated. ∎

When applying the union bound, this gives us that *no* tree with root $w \in V_G$ of G' exceeds size $O(\log n \cdot \log\log n)$ w.h.p. However, remember that another type of component exists in G', namely cycles that may have additional nodes attached to them. Fix one of these components and let $C = \{v_1, ..., v_{|C|}\}$ be the set of nodes of the cycle. Furthermore define $A_i := \{v \mid v \in (V_B \setminus C) \wedge$ a path from v to v_i exists in $G' \} \cup \{v_i\}$. Then, this set induces a tree in G' unless the loop (v_i, v_i) is contained in E' which implies $|C| = 1$. In any case, one may see the component as being induced by the set $\bigcup_i A_i$ as a forest of trees, whose roots lie on the cycle C. To determine the total size of the set $\bigcup_{i=1}^{|C|} A_i$, we look at the growth of them layer-by-layer. However, this time we let all $|C|$ of these trees grow at the same time, again uncovering edges step-by-step. That is, $L_0 = C$ and construct any set L_i with $X_i = |L_i|$ just as we did when considering the roots $w \in V_G$, i.e., $L_i = \{v \mid v \in (V_B \setminus \bigcup_{j=0}^{i-1} L_j) \wedge \pi_v(1) \in L_{i-1}\}$. Observe that for this fixed cycle we can describe the sequence $\{X_i\}$ by the number of nodes in $V_B \setminus C$ being at distance i from the cycle. When replacing the set V_B by $V_B \setminus C$, the result of Lemma 2 also holds for this layer sequence. As Lemma 1 guarantees that $X_0 < O(\log n)$, the statement of Lemma 3 follows accordingly and allows us to repeat the whole approach.

Corollary 2. *The results of Lemma 3, Corollary 1, Lemma 4 and finally Lemma 5 also hold for the sequence $\{X_i\}$ of nodes in distance i to some fixed cycle in G'.*

b) Accounting for Inner Edge Failures: So far we established that none of the components in G' is of size more than $O(\log n \cdot \log\log n)$ w.h.p. Remember however that we still need to account for the failures in $\mathcal{F}^{(in)}$. We start by showing that only $O(\log n)$ of them lie on any path in G', even if the adversary fails $\Theta(n)$ inner edges.

Lemma 6. *The number of nodes $v \in V$ that have their first failover edge $(v, \pi_v(1))$ destroyed by the adversary is $O(\log n)$ w.p. $1 - n^{-3}$*

Proof: Consider some fixed node v and assume that the adversary destroyed $f_v < n - 1$ inner edges connected to v. Remember, the adversary cannot predict the permutation. Therefore a failed edge lies at the beginning of π_v w.p. at most $f_v/(n-1)$. Define r.v. X_v s.t. $X_v = 1$ iff $(v, \pi_v(1)) \in \mathcal{F}^{(in)}$, and 0 otherwise. Clearly the random variable $X = \sum_{v \in V} X_v$ is what we are looking for where

$$E[X] = \sum_{v \in V \setminus \{d\}} X_v = \frac{1}{n-1} \cdot \sum_{v \in V \setminus \{d\}} f_v \leq \frac{2\gamma n}{n-1} < 2.$$

In the third step, we use that $\sum_{v \in V \setminus \{d\}} f_v = 2 \cdot |\mathcal{F}^{(in)}| < 2\gamma n$. The factor 2 is introduced because the sum represents the number of incident failed edges, summed up over all nodes. This way, each of the $|\mathcal{F}^{(in)}| < \gamma n$ failed inner edges is counted twice. As all $X_v \in \{0, 1\}$ are independent from each other, we may apply Chernoff bounds to X. For $\delta = 9 \ln n / E[X] < \delta^2$ this yields that $Pr[X > E[X](1 + 9 \ln / E[X])] \leq \exp(-3 \ln n) = n^{-3}$. Hence $\Pr[X > 2 + 9 \ln] \leq n^{-3}$. ∎

In the following we show how to transfer $G'^{(i)}$ into $G''^{(i)}$ for any $i = 1, 2, 3$. The basic idea is to remove edges $(v, \pi_v^{(i)}(1))$ that lie in $\mathcal{F}^{(in)}$ from $G'^{(i)}$ and replace them with edges $(v, \pi_v^{(i)}(j))$, where $\pi_v^{(i)}(j)$ is the first reachable neighbor in the permutation of v. This way, the graph $G''^{(i)}$ represents the correct path of the packets with hop counter $(i-1)C_1 \leq h(p) < iC_1$ when following our protocol. The edge replacements throughout the construction of $G''^{(i)}$ causes subtrees of size $O(\log n \cdot \log\log n)$ to relocate (see Fig. 4). This may cause some components in $G''^{(i)}$ to be extended by these relocated subtrees, and also a new type of component may be created. That is, the roots of relocated subtrees may

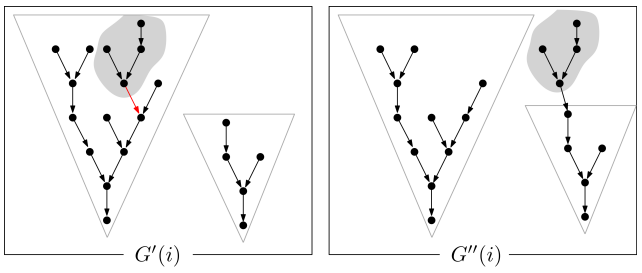

Fig. 4. For some node v the edge $(v, \pi_v^{(i)}(1))$ is failed (marked in red). In $G''(i)$ this edge is replaced by $(v, \pi_v^{(j)})$, causing the subtree rooted in v to relocate.

connect with nodes in other subtrees and cause the formation of a cycle.

Formally, we can show the following claim. As in the previous sections, we focus on a fixed i and will omit the superscript (i). The idea behind the proof is that, according Lemma 6, only $O(\log n)$ such subtrees are relocated. It is unlikely that more that $O(1)$ of these subtrees connect to the same structure. Full proof is given in the supplementary material.

Lemma 7. *Consider the graph G''. Then, none of the components contained in G'' has more than $O(\log n \cdot \log\log n)$ nodes. Furthermore, the number of contained cycles remains in $O(\log n)$ with each not exceeding length $O(\log n)$. Additionally, any packet that is not trapped in some cycle takes at most $C_1 = 16\log_{1/\varepsilon} n$ steps to reach d. All above statements hold w.p. $1 - O(n^{-2})$*

c) From Forests to Load: To determine the total load some node v receives we look at the graphs $G''^{(i)}$ one after another. When starting the *all-to-one* routing, each node initiates a flow and all of them follow paths according to the single outgoing edges in $G''^{(1)}$. Now, most of the flows reach the destination d after at most C_1 many hops. However, some might be trapped inside a cycle. In both cases consider the cost $T_1(v)$ that occurred at any node v until this point. Upon reaching a hop value of C_1 the flows currently trapped in a cycle start to traverse according to the permutation $\pi_v^{(2)}$. If we know where these cycle nodes are located in $G''^{(2)}$, we can again track the flows' paths and determine the loads caused by the next C_1 many hops, denoted $T_2(v)$. Finally, we repeat this approach one more time in the same manner and obtain the values $T_3(v)$. In the following we say that flows *exit* the system $G''^{(i)}$ if their respective hop counter reaches iC_1. Similarly we say flows *enter* the system $G''^{(i)}$, if they reach hop value $(i-1)C_1$ and $G''^{(i)}$ becomes relevant for their failover paths for the first time. Finally, we argue that $T_i(v) = 0$ w.h.p. for any v and $i \geq 4$, which is why we originally only required three permutations, and deduce that the total load v receives is $T(v) = T_1(v) + T_2(v) + T_3(v)$.

We start with the following statement. The proof, given in the supplementary material, exploits the structure of $G''^{(i)}$, as each node has out-degree either one or zero.

Lemma 8. *Let $i \geq 1$ and assume that every component in $G''^{(i)}$ is entered by $O(\log n \cdot \log\log n)$ total flows. Then, every node v that is not contained in a cycle receives a load of at most $T_i(v) = O(\log n \cdot \log\log n)$. Nodes contained in a cycle in $G''^{(i)}$ receive $T_i(v) = O(\log^2 n \cdot \log\log n)$ load until the flows exit $G''^{(i)}$ w.p. $1 - O(n^{-2})$*

Clearly, for $i = 1$ and $G''^{(1)}$ the assumption of above lemma is satisfied. This is because Lemma 7 guarantees that all structures are of size $O(\log n \cdot \log\log n)$ and each node starts sending one flow of packets. The next lemma deals with the inductive step and $i > 1$. The proof relies on the fact that the permutations $\pi_v^{(i+1)}$, $v \in V'$, were chosen independently from any previous permutation $\pi_v^{(j)}$, $j < (i+1)$. A detailed proof is given in the supplementary material.

Lemma 9. *Assume that the assumption of Lemma 8 holds for $G''^{(i)}$. Then, also in $G''^{(i+1)}$ no component is entered by more than $O(\log n \log\log n)$ flows w.p. $1 - O(n^{-2})$*

Above lemma concludes the inductive step, showing that Lemma 8 is applicable to all $i \geq 1$. We now state that it is indeed enough to only look at the graphs $G''^{(i)}$ where $1 \leq i \leq 3$ to determine the maximum load. The proof of the statement is very similar to the proof of Lemma 1 and given in the supplementary material. We track a fixed packet that starts at some node and argue that it is unlikely to be stuck in a cycle in $G''^{(1)}, G''^{(2)}$ and $G''^{(3)}$.

Lemma 10. *Fix an arbitrary packet p sent by a node $v \in V_B$. Then, p ends up in a cycle in $G''^{(1)}, G''^{(2)}$ and $G''^{(3)}$ w.p. at most $\mathrm{polylog}\, n/n^3$. Additionally, it holds that $T_i(v) = 0$ for all nodes $v \in V \setminus \{d\}$ and $i \geq 4$ w.p. $1 - \mathrm{polylog}\, /n^2$.*

Note that this lemma also implies that $O(\log n)$ hops suffice for any packet to reach the destination. We established in Lemma 9 that the assumption in Lemma 8 is indeed fulfilled for $G''^{(1)}, G''^{(2)}$ and $G''^{(3)}$. Hence, the total load any node receives that is not contained in a cycle in either $G''^{(2)}$ or $G''^{(1)}$ is $O(\log n \cdot \log\log n)$. And those that lie on a cycle have a load of $O(\log^2 n \cdot \log\log n)$, where according to Lemma 7 at most $O(\log^2 n)$ such nodes exist. Theorem 1 follows accordingly.

IV. Circumventing Cycles by Partitioning

One of the major challenges of the *3-Permutations* protocol is to cope with temporary cycles, which may be introduced when employing randomization into the failover strategy. For packets with large hop counts, which indicate the existence of a cycle, we effectively provided different failover routes. In the following we present the *Intervals* routing protocol that (i) does not introduce any cycles in the packets routing paths w.h.p. and (ii) is purely destination based. This comes however at the cost of smaller maximum resilience against failures.

The *Intervals* protocol works as follows. We assume that every node is given a unique ID or address, which is known to the other nodes. Therefore we can enumerate the nodes by $v_1, v_2, ..., v_n$. Let now α be some small constant $0 < \alpha < 1$. We partition the nodes of the graph into consecutive sets of

Input: A packet with destination d

1: **if** (v, d) is intact **then** forward p to d and **return**

2: Forward p to first directly reachable node in $\pi_{v,d}$

Fig. 5. *Intervals* protocol. Point-of-view of some node v

size $I = n/4\log_{1/\alpha} n$. That means, the i-th set R_i contains nodes with addresses in the range

$$\Big[(i-1)\cdot\frac{n}{4\log_{1/\alpha} n}+1\ ,\ i\cdot\frac{n}{4\log_{1/\alpha} n}\Big].$$

Assume that the value α is chosen such that both, the interval bounds and the number of intervals, are integers. The next step is similar to Section III. Every node tries to directly forward a packet to the desired destination d, if the direct link is available. Otherwise, again a permutation $\pi_{v,d}$ of nodes is consulted and the packet is sent to the first reachable partner in $\pi_{v,d}$. The crucial difference to the *3-Permutations* protocol is the following: for some node v that lies in the interval R_i, the permutation $\pi_{v,d}$ is a permutation of the nodes in $R_{(i+1)} \setminus \{d\}$. Hence, only edges ranging from nodes in the set R_i to nodes in R_{i+1} are considered as possible failover edges. To allow for a proper protocol, we assume that nodes of the rightmost interval choose failover edges into R_0. We show in the upcoming analysis that this protocol does not create any cycles in the routing paths w.h.p. The following statement allows the adversary to fail up to $O(n/\log n)$ many edges. Note that this protocol is purely destination based and therefore any deterministic scheme operating under this constraint would allow $\Omega(n/\log n)$ load to be created by the adversary [14].

Theorem 2. *Assume the adversary is allowed to fail up to $\alpha \cdot I$ many edges, for some arbitrary constant $0 < \alpha < 1$ where $I = n/(4\log_{1/\alpha} n)$. Then, when considering all-to-one routing to any destination d, the* Intervals *protocol guarantees a maximum of*

$$O(\log n \cdot \log\log n)$$

load at any node except d and edge w.h.p. Additionally, no packet performs more than $O(\log n)$ hops w.h.p.

For $\alpha = 1/e$ above statement provides the maximum resilience of $n/(4e\log_e n)$. Furthermore, assume the adversary fails $\Omega(n/\log n)$ destination edges (v, d), with all such nodes v being in the same interval R_i. Then, similar as in the case of the *3-Permutation* protocol, a balls-into-bins argument [55] shows that after all nodes in R_i send their flows to randomly chosen nodes in R_{i+1}, at least one node has load $\Omega(\log n/\log\log n)$ w.h.p.

Concerning the notation we carry over everything defined in Section III-A. Additionally we extend the notation for the sets of failed edges as $\mathcal{F}_i^{(in)}$ and $\mathcal{F}_i^{(d)}$. These sets only contain failed edges started in the i-th interval and, in case of $\mathcal{F}_i^{(in)}$, have partners in the interval $i+1$. Just as in the protocols description we denote the set of nodes inside the i-th interval as R_i where $I = |R_i|$.

We remark that the result of Theorem 2 also holds, if we require that $|\mathcal{F}_i^{(in)}| < \varepsilon \cdot I$ and $|\mathcal{F}_i^{(d)}| < \gamma \cdot I$ *for any* interval R_i, as long as $\varepsilon+\gamma \leq 1-\Delta$ for some constant $\Delta > 0$. As the sets $\mathcal{F}_i^{(in)}$ and $\mathcal{F}_i^{(d)}$ are specified for some fixed destination d, we require this property for all possible destinations d.

Regarding the memory complexity, for a fixed destination each node needs a permutation of $O(n/\log n)$ nodes, hence $O(n)$ bits in total. As in case of the *3-Permutations* protocol, the set of permutations $\{\pi_{v,d} \mid d \in V\}$ some node v requires can be derived from a single permutation π_v, and only the first $3\log_{1/\alpha} n$ entries of each permutation need to be stored (see corresponding description on page 4). This allows for a reduction of the *total* memory required per node to $O(\log^2 n/\log(1/\alpha))$.

A. Analysis

A main motivation behind Section IV is to eliminate the need to perform any kind of cycle resolution. To that end we start by showing that our protocol does not introduce any cycles into the packets routing paths and at the same time derive that the hop count of *any* packet remains in $O(\log n)$.

To show that the maximum load occurring lies in $O(\log n \cdot \log\log n)$, we take a similar approach as in Section III-B. That is, we fix some node $v \in V_G \cap R_i$ in some interval. Then, all the sources of packets that are forwarded over the edge (v, d) form a tree rooted in v. We again argue that, in expectation, the number of nodes per level of this tree decreases exponentially fast and reuse parts of Section III-B.

a) Cycles: Some packet located in R_i has only two possibilities for its next hop. Either it is directly forwarded to the destination, or it is forwarded to a node of the set R_{i+1}. To end up in a cycle it needs to traverse a sequence of $4\log_{1/\alpha} n$ intervals, hitting a node $v \in V_B$ with every hop. This is unlikely and formalized as follows.

Lemma 11. *Let p be an arbitrary packet to be routed to destination d. Then, its routing path does not contain any cycles and it reaches the destination within $4 \cdot \log_{1/\alpha} n + 1$ hops w.p. at least $1 - n^{-4}$.*

Proof: Consider some packet p originating from some node $v \in V_B$ in an arbitrary interval R_i. In the next hop, p travels to some node v' in R_{i+1}. If $v' \in V_G$ the packet is then forwarded directly to d. Now, $|\mathcal{F}_{i+1}^{(d)}|$ bad nodes exist in R_{i+1}. In the worst-case the adversary fails $|\mathcal{F}_i^{(in)}|$ edges (v, w) with $w \in R_{i+1} \cap V_G$. Hence, the probability that v' is a bad node is at most

$$\frac{|\mathcal{F}_{i+1}^{(d)}|}{I - |\mathcal{F}_i^{(in)}|} < \alpha.$$

Here we used that $|\mathcal{F}_i^{(in)}| + |\mathcal{F}_{i+1}^{(d)}| \leq \alpha I$. Therefore, the probability that p hits bad nodes in $4 \cdot \log_{1/\alpha} n$ consecutive hops is at most $1/n^4$. The result immediately follows. ∎

b) Carrying over Previous Results: In the following we show that the result of Lemma 5 also holds when nodes follow the *Intervals* protocol. That is, for some fixed node $w \in V_G$ we construct a tree as follows. Assume $w \in R_i$ and let $L_0 = \{w\}$ denote the root of this tree. The j-th level of the tree associated with v is defined by $L_j = \{v \mid (v \in R_{i-j} \cap V_B) \wedge (\pi_v(k) \in L_{i-1}) \wedge (\forall \ell < k : (v, \pi_v(\ell)) \in \mathcal{F}^{(in)}) \wedge ((v, \pi_v(k)) \notin \mathcal{F}^{(in)})\}$.

That is, L_j is the set of nodes whose packets reach v in exactly j hops (for easier readability we neglect the fact that wrap-around might occur). Formally we show the following.

Lemma 12. *Let $w \in V_G$ be arbitrary and assume that $w \in R_i$. Furthermore let $X_j = |L_j|$, where L_j is the j-th level of the tree associated with v. Then, it holds that $E[X_{j+1}] \leq X_j \cdot \alpha$.*

Proof: Let $w \in V_B \cap R_i$. Assume that due to the failed edges by the adversary, X_j many nodes of the interval R_{i-j} route packets over w to the destination. Consider now some node $v \in R_{i-(j+1)} \cap V_B$. According to the assumption of Theorem 2 at most $\varepsilon \cdot I$ such nodes exist. Hence v hits one of the X_j nodes w.p. smaller than $X_j/(I - |\mathcal{F}^{(in)}_{i-(j+1)}|)$. Now, since we have $|\mathcal{F}^{(d)}_{i-(j+1)}| = |R_{i-(j+1)} \cap V_B|$, we obtain that

$$E[X_{j+1}] \leq |F^{(out)}_{i-(j+1)}| \cdot \frac{X_j}{I - |\mathcal{F}^{(in)}_{i-(j+1)}|} \leq \alpha \cdot X_j$$

■

With this we established a statement similar to the first part of Lemma 3. It is easy to see that the size of each level X_j can be modelled with a sum of independent Poisson trials, when constructing the tree level-by-level. Furthermore, Lemma 11 establishes the property of Corollary 1 and since Lemma 12 guarantees that the levels shrink exponentially fast in expectation, the statement $X_i < C \log n$, C large enough, follows by applying Chernoff bounds. That said, we established all necessary requirements and a simple repetition of the corresponding analysis allows us to reuse Lemma 4 and Lemma 5. Summarizing, we get the following and conclude the proof of Theorem 2.

Corollary 3. *Let $w \in V_G$ be a good node and let $\{X_i\}$ be defined as in Lemma 12. Then it holds that $\sum_i X_i = O(\log n \cdot \log\log n)$ w.p. $1 - \text{polylog}\, n/n^3$.*

V. Further Reducing the Congestion

In this section we present a third protocol, called *Shared-Permutations*, that improves the bound of the maximum load observed in Theorem 1 and Theorem 2 under the assumption that the nodes share a common but randomized permutation over V. This could for example be achieved by computing parts of the routing tables starting from the same seed for the random generator, which is unknown to the adversary. While this assumption is indeed a weakness inherent to this protocol, it can be offset in case the adversary does not compromise one of the nodes directly: if all nodes manage to agree on a new permutation from time to time, this may invalidate previously obtained information by the adversary about the traffic flow. We also assume that the packet headers are equipped with a hop field of size $O(\log\log n)$ bits, which is initially set to 0 and may be accessed by the nodes of the network.

The *Shared-Permutations* protocol works as follows. Again we consider an arbitrary but fixed destination d. Every node $v \in V$ is equipped with permutations $\pi_{0,d}, \pi_{1,d}, ..., \pi_{C_1,d}$ of all nodes $V \setminus \{d\}$, where C_1 is a value $O(\log n)$ to be specified later. Now, contrary to the *3-Permutation* protocol, these permutations are assumed to be *globally* agreed upon without being known to the adversary. Furthermore, each permutation is chosen u.a.r out of the set of all possible permutations. Additionally we assume that v stores C_2 additional permutations $\pi_{v,j,d}$ on $V \setminus \{d\}$, only known to v itself and chosen u.a.r. Here $j \in \{E_2, E_2 + 1, \ldots, E_2 + C_2 + 1\}$ for $E_2 = C_1 + 1$ and C_2 is another value in $O(\log n)$.

Input: A packet with destination d and hop count $h(p)$

1: **if** (v, d) is not failed **then** forward p to d, set $h(p) \leftarrow h(p) + 1$ and **return**
2: **if** $h(p) < E_2$ **then** $v' \leftarrow$ successor of v in $\pi_{h(p),d}$
3: **if** (v, v') is not failed **then** forward p to v'
4: **else** $h(p) \leftarrow E_2$.
5: **if** $h(p) \geq E_2$ **then** forward p to first directly reachable node according to $\pi_{v,h(p),d}$
6: $h(p) \leftarrow h(p) + 1$

Fig. 6. *Shared-Permutations* protocol. Point-of-view of some node v

Assume now that a packet p with destination d arrives at node $v \in V$ and denote its current hop counter by $h(p)$. First of all, if the link (v, d) is not failed the packet is directly forwarded to the destination. Otherwise if $h(p) < E_2$, the node v forwards it via a link (v, v') where v' denotes the node following v in the global permutation $\pi_{h(p),d}$. In case this link is failed, v raises the hop counter of p to E_2 instead and forwards it to the *first non-failed* edge according to $\pi_{v,E_2,d}$. The case we did not consider yet is $h(p) \geq E_2$. In this case p is routed over the first reachable partner in $\pi_{v,h(p),d}$. Finally, in every case, $h(p)$ is increased by one. A pseudo-code describing this algorithm is given in Fig. 6. The common global permutations allow the flow to be distributed more evenly among the network, reducing the congestion to $O(\sqrt{\log n})$, even if $\Omega(n)$ edges are failed by the adversary.

Theorem 3. *Assume that the adversary is allowed to fail $\alpha \cdot n$ edges total, where $\alpha < 1$ is a constant*[5]*. When performing all-to-one routing to any destination d, the* Shared-Permutations *protocol guarantees a maximum flow of*

$$O(\sqrt{\log n})$$

on any node (except d) and edge w.h.p. Additionally, no packet traverses more than $O(\log n)$ hops w.h.p.

Assuming it is possible for the nodes to agree on common permutations that are not known to the adversary, the maximum load can be decreased by more than a factor $\sqrt{\log n}$ compared to the protocols in Sections III and IV. Note that this result breaks the $\Omega(\log n/\log\log n)$ lower bound of the *3-Permutations* and *Intervals* protocols.

Regarding space complexity, our nodes are required to store $O(\log n)$ permutations of n nodes per destination. Therefore in the most simple case we require $O(n^2 \cdot \log^2 n)$ bits at most. However, the same improvements as described in Sections III and IV can be made to store the permutations more efficiently and achieve a memory complexity of $O(\log^3 n/\log(1/\alpha))$

[5]Just as in our first algorithm, α can be any constant that lies in the range $0 < \alpha < (n-1)/n \approx 1$.

bits per node. Note that the protocol requires knowledge of the values C_1 and C_2, which can both be set to $5\log_{1/\alpha} n$. These values are given in Lemma 14 and Lemma 17, together bounding the maximum number of hops any packet performs until it reaches the destination d w.h.p. If α is not known to the nodes, then a slow growing function in $\omega(\log n)$ can be used for C_1 and C_2, which comes at the cost of slightly increased memory complexity.

A. Analysis

Throughout the analysis we consider a fixed destination d and omit the corresponding index from all permutations. We use the notation defined in Section III-A and start by neglecting any failed edges in the set $\mathcal{F}^{(in)}$. Next, we assume that each node sends 1 packet with destination d from each node $v \in V$. We consider the number of packets that have i hops while still not having reached the destination d and see that this set decreases exponentially fast. Additionally no packet traverses more than $C_1 < E_2$ many hops w.p. at least $1 - n^{-2}$. Therefore, without any inner edge failures, the only relevant permutations for our failover strategy are $\pi_0, ..., \pi_{E_2-1}$. Finally we account for the failures in $\mathcal{F}^{(in)}$ and make use of the permutations $\pi_{v,j}$. We consider the maximum load caused by the flows after reaching hop value E_2 separately and deduce that this value is $O(\sqrt{\log n})$ w.h.p.

a) Staying in Line: We start by neglecting the failures in $\mathcal{F}^{(in)}$, i.e we assume first that $|\mathcal{F}^{(in)}| = 0$ and $|\mathcal{F}^{(d)}| < \varepsilon \cdot n$. Furthermore, assume that every node $v \in V \setminus \{d\}$ starts sending a single packet to destination d. Then, the number of packets that pass through some node v is equivalent to the number of flows passing through v. Notice, that due to the global permutations, no node is visited by more than 1 packet with the same hop value. Consider the set of packets hop-for-hop and denote H_i as the set of packets that reached hop i at some point without reaching the destination. Clearly $|H_0| = (n-1)$ and $|H_1| = |V_B|$. Additionally, a fixed packet lands on a node V_B with its next hop with probability roughly $|V_B|/n \leq \varepsilon$. Only in case this event occurs, the packet cannot proceed directly to the destination on its subsequent hop. This implies that the number of remaining packets decreases exponentially with every further hop. In the supplementary material we give a full proof to the following statement.

Lemma 13. *Assume $|\mathcal{F}^{(in)}| = 0$. Let H_i denote the set of packets have not reached d after i hops. Then, for $|H_i| = \Omega(\log n)$ it holds w.p. at least $1-n^{-4}$ that $|H_{i+1}| \leq |H_i| \cdot \sqrt{\varepsilon}$.*

We use above result to show that packets reach d with $O(\log n)$ hops. The full proof is given in the supplementary material.

Lemma 14. *Fix some packet p with destination d. Then, assuming $\mathcal{F}^{(in)} = 0$, it requires at most C_1 steps to reach the destination w.p. at least $1-n^{-3}$. Here C_1 is a value bounded above by $5\log_{1/\varepsilon} n$.*

Recall the following invariant: when fixing some node v and hop value i, the node v receives *at most* 1 packet with such hop value. This leads to following result.

Lemma 15. *Consider some node $v \in V$ and assume $|\mathcal{F}^{(in)}| = 0$. Then, if every node sends 1 packet with destination d, v is visited by packets $O(\sqrt{\log n})$ times w.p. at least $1 - n^{-3}$.*

Proof: Let i^* be the first time such that H_{i^*} reaches size $O(\log n)$. We start by showing that throughout hops $1 \leq i < i^*$ the node v is visited by at most $O(\sqrt{\log n})$ packets in total. For this range of i, we know according to Lemma 13 that the size of H_i decreases exponentially fast, i.e $|H_{i+1}| < |H_i| \cdot \beta$ for some constant $0 < \beta < 1$ that depends on ε. Fix now some node v and let $Y_i = 1$ iff v receives a packet with hop value i at some point, and 0 otherwise. Similar as in the proof of Lemma 13, we argue that the packets with hop value i are distributed uniformly – and independently of any earlier hops – among the nodes of the network. Hence $P[Y_i = 1] = H_i/(n-1)$ and we are interested in $Y = \sum_{i=1}^{i^*} Y_i$. Note that $P[Y_i = 1] = H_i/n < \varepsilon \cdot \beta^i$, where we wrote n instead of $n-1$ for ease of readability. Then

$$P[Y = k] = \sum_{\substack{S \subseteq \{1,...O(\log n)\} \\ \text{with } |S|=k}} \Big(\prod_{j \in S} P[Y_j = 1] \cdot \prod_{\ell \in \{1...O(\log n)\} \setminus S} 1 - P[Y_\ell = 1] \Big).$$

The second product can be crudely bounded above by 1. The first product reaches its maximum value for $S = \{1, ..., k\}$ and is bounded by $(\varepsilon\beta \cdot \varepsilon\beta^2 \cdot ... \cdot \varepsilon\beta^k)$. Therefore we get

$$P[Y = k] < \left(\frac{e \cdot C \log n}{k} \right)^k \varepsilon^k \cdot \beta^{k(k-1)/2}.$$

Now assume $k = C' \cdot \sqrt{\log n}$ for some sufficiently large constant C'. In this case

$$P[Y = k] < O\left(\sqrt{\log n}\right)^{C'\sqrt{\log n}} \cdot \left(\frac{1}{n}\right)^5.$$

Clearly the first term lies in $o(n)$. When increasing k further, the probability only gets smaller. Applying the union bound over the remaining $O(\log n) - C'\sqrt{\log n}$ larger values of k, we get $P[Y \geq C'\sqrt{\log n}] < n^{-3}$. Adding 1 for the packet that was initialized on v yields the result. ∎

Clearly this implies that both, the maximum node load and the edge load are $O(\sqrt{\log n})$ in the case of $|\mathcal{F}^{(in)}| = 0$.

b) Accounting for Inner Edge Failures: We consider two copies, $S^{(out)}$ and S of our initial graph after the adversary failed at most αn edges according to its strategy. In $S^{(out)}$ we repair all failures $\mathcal{F}^{(in)}$, which results in ignoring inner edge failures just as described above. Again we consider the equivalent point-of-view of each node sending a single packet to d instead of a consecutive flow. The idea in the following is to consider only $S^{(out)}$ and each time an inner-edge (u, v) is chosen for communication that is failed in the original graph, the packet is copied and placed with hop count E_2 on u in S. This way S contains packets with hop count of at least E_2. The packets in $S^{(out)}$ however continue as if the edge was intact. Note that, by Lemma 14, S w.h.p. only consist of packets that are redirected because of inner edge failures. The idea behind the analysis is the following: Let $S^{(out)}$ run until all packets reached the destination d and determine the

number of packets starting in S. We then let the system S run and it is easy to see that we can majorize the load some node v receives in the original process by adding up the loads of v in $S^{(out)}$ and S respectively. This is because in $S^{(out)}$ we do not remove packets but copy them to S instead.

In Lemma 15 we already established the load some node v receives in $S^{(out)}$. To bound the load such a node receives in S, we start by showing that only few packets are initialized in S. The full proof is given in the supplementary material.

Lemma 16. *Consider the number of packets p that reach a load of E_2 at some point. This number lies in $O(\log n \cdot \sqrt{n})$ w.p. at least $1 - 2 \cdot n^{-3}$.*

As all packets in S have hop count at least E_2, only the *local* permutations $\pi_{v,j}$ are used as part of our failover strategy. While this leads to nodes possibly receiving multiple packets of the same hop value, the number of initial packets lies in $O(\log n\sqrt{n})$ only. We give a detailed proof to the following statement in the supplementary material.

Lemma 17. 1) *Fix some arbitrary packet in S. Then, it reaches the destination after at most C_2 hops for $C_2 < 3\log_{1/\alpha} n$ w.p. at least $1 - O(n^{-3})$.*
2) *Each node in S is reached by at most $O(\sqrt{\log n})$ packets in total and w.p. at least $1 - O(n^{-3})$.*

We established that in both systems, $S^{(in)}$ and S, each node has a load of $O(\sqrt{\log n})$. Theorem 3 follows accordingly.

VI. Simulations

To complement our theoretical analysis, we compare an adapted version of our protocols against other state-of-the-art local failover strategies [18], [24] in the widely deployed Clos datacenter topology [22], [23]. More precisely, we consider the special case of the Clos topology with 3 layers, sometimes simply referred to as fat-tree. All source code used to derive the results in this section can be found on GitHub [58].

For our experiments we considered 8 different protocols in total, which can be described as follows.

Our own protocols: Abbreviated by `ThreeP-D`, and `Interval-D` we consider variants of our *3-Permutations* and *Intervals* protocol from Sections III and IV, adapted to the Clos topology. Additionally, we consider two further variants of these protocols denoted by `ThreeP-ID` and `Interval-ID`. The ending `-ID` indicates that for these protocols, we select the permutations which are used for forwarding not only depending on the destination of the packet but also the inport from which the packet arrives. We employ these additional variants of our protocol as the reference protocols require support for destination and inport-based forwarding. A detailed description of these protocols is given in Section VI-A.

Related Approaches: We consider the state-of-the art local failover protocols *DetCirc*, *PRNB*, *CASA* and *SquareOne* [18]. Throughout our experiments, we refer to them as `A-Det`, `A-PRNB`, `A-CASA` and `Square1`, respectively. The first two protocols are slightly modified versions of the *HDR-Log-K-Bits* and *Bounced-Rand-Algo* originally presented in [24], and versions of all of these protocols have also been evaluated using simulations in [18]. The first three protocols have in common that they are so-called *arborescence-based* routing protocols. To route a packet toward some destination d in a topology which is ℓ-connected, the protocols use a set of pre-computed sub-trees $\{T_0, T_1, ..., T_{\ell-1}\}$ called arborescences. Each such tree has d as its root, consists of all nodes in the topology and does not share any directed edges with other trees in the set. Packets then start on some arborescence T_i and are routed along the edges in the tree until they hit the destination. In case the packet arrives at a failed edge, another arborescence T_j with $j \neq i$ is selected along which the packet may continue its path towards d. This procedure is repeated until the packet arrives at d. The aforementioned protocols differ in the way this alternate arborescence T_j is selected in case a failed edge is encountered. In `A-Det`, j is selected deterministically and set to $(i+1) \mod \ell$. In `A-PRNB`, j is selected uniformly at random out of $\{0, 1, ..., \ell - 1\} \setminus \{i\}$. Finally, `A-CASA` uses a sophisticated pre-computed matrix which is constructed via so-called balanced incomplete block designs (BIBD) to deterministically select j.

The `Square1` protocol operates differently. In this protocol, packets with destination d and source s are routed over one of the ℓ shortest edge-disjoint paths from s to d. At first, the packet attempts to follow the shortest such path to reach d. However, in case a failed edge is encountered, the packet needs to traverse back to s and follow the next-shortest path instead. This is repeated until the destination d is reached.

A. Engineering Protocols for the Clos Topology

In this section we discuss the required adaptation of our protocols to be employed in the Clos topology as well as the computation of the arborescences and edge-disjoint shortest-paths required by the related protocols. We start with a short definition of the Clos topology, which is required to explain the required modifications to our own protocols.

The Clos topology with 3 layers consists of $k/2 \cdot k/2 + k \cdot k = \Theta(k^2)$ nodes (or routers), each having at least k ports. These nodes are partitioned into $k/2$ *blocks* and k *pods*, which we assume to be numbered from 1 to $k/2$ and 1 to k, respectively. Each block contains exactly $k/2$ many nodes, which we again assume to be numbered from 1 to $k/2$. Each pod consists of two sets of $k/2$ nodes each. The first set we call the *top* nodes while the second set we call *bottom* nodes. Bidirectional links are inserted such that in each pod, the top and bottom nodes form a complete bipartite graph. Additionally, the i-th top node in each pod is connected to all nodes in the i-th block and vice versa. Endpoints using this communication infrastructure are connected at the remaining $k/2$ open links at each bottom node. Throughout our experiments, we focus on forwarding flows which have bottom nodes as source and destination.

Adapting our Own Protocols: We start with an explanation of the `Interval-D` protocol. Note that this adapted protocol has been analyzed theoretically after we performed our first experiments in a follow-up paper [54]. First we need to partition the Clos topology into more fine-grained pieces. The top as well as the bottom nodes are split into $K := \lfloor \log(k) \rfloor$ consecutive partitions, deviating in size by at most ± 1. Similarly, the $k/2$

block nodes in each block are also split into K many intervals. Furthermore, consider nodes in the b-th block. Each such node is connected to the b-th node in the top layer of every pod. We also assume that, for each block, the set of such top nodes is partitioned into K what we call *vertical intervals*. In the remaining description, when we say that some node v forwards a packet with destination d to a random node in an interval, then we assume that the random selection follows the approach described in Section IV. That is, the node v consults a random permutation of all nodes in this interval (one such permutation is precomputed for each destination d) and then forwards the packet to the first node u in this permutation, such that the link (v, u) is not failed. When following the `Interval-D` protocol, the forwarding rules for a packet with destination d arriving at a node v then depend on whether v is a block, top or bottom node. First, assume that v is a bottom node in the p-th pod. Then, if $v = d$ nothing needs to be done. Otherwise, let j denote the interval of v. In such case, v forwards the packet to a random top node of pod p that also lies in the j-th interval. Second, if v is a top node in some pod p (in more detail, let v be the i-th top node in p). Then, if the destination also lies in p, the node v attempts to forward the packet directly to d. In case this link is failed, it instead forwards the packet to a random bottom node of p in interval $(j + 1) \mod K$. This way, as soon as the packet lies on some node in the pod of the destination, it will ping-pong between bottom and top nodes until it reaches a top node whose link to d is not failed. However, if v is not in the pod of the destination, then it forwards the packet to a random node in the j-th interval of the i-th block. In the latter case v is a node in some interval j of a block (let it be the b-th block). Each node in block b is connected to the b-th top node in the pod of the destination. In case the link to this top node is not failed, v forwards the packet over this link. Otherwise, v forwards the packet to a random top node in the vertical interval $(j + 1) \mod K$.

Following above description, the `ThreeP-D` protocol, which can be seen as an adaptation of the *3-Permutation* protocol, is now easy to define. This definition is very similar to the `Interval-D` protocol except for two differences: First, we set $K = 1$, i.e., we don't split the block, top, or bottom nodes into further intervals. Second, we assume that each node v does not only store one permutation but 6 permutations per destination d. Depending on the hop count of the arriving packet one of these permutations is selected. More precisely, the i-th permutation with $i \in \{1, 2, ..., 5\}$ is consulted for packets with hop count $[(i-1) \cdot \lfloor \log k \rfloor \ , \ i \cdot \lfloor \log k \rfloor]$. The 6-th permutation is used for packets with any larger hop count. The reason that we use 6 permutations instead of the 3 as defined in Section III is that empirical results indicated that it is beneficial to switch permutations frequently. This, however, requires the employment of additional permutations to avoid the creation of permanent forwarding loops.

Finally, the two additional variants called `ThreeP-ID` and `Interval-ID`. They follow the exact same definition as their protocols of similar name with only one exception. We now assume that nodes store additional sets of these randomly generated permutations: one per combination of possible destination address *and* inport. In contrast to the basic `ThreeP-D` and `Interval-D` protocols, which select the permutations used for forwarding solely depending on the destination address, we assume this selection to be performed randomly for each pair of destination and inport.

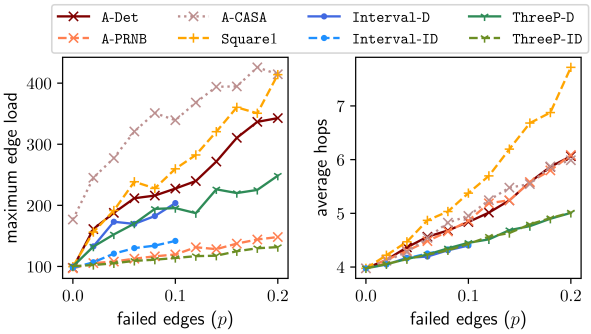

Fig. 7. Average maximum edge load and average amount of hops required to reach the destination when performing *all-to-one* routing.

Employing Related Approaches: While the related protocols we consider are applicable to general graphs, they require the pre-computation of sets of arborescences as well as edge-disjoint shortest paths for every possible destination d. In order to compute the set of arborescences required by the first group of protocols, we employed the *round-robin* approach with swaps presented in [53]. Even with this efficient approach, the calculation of these arborescences for a single destination node d of the Clos topology with $k = 80$ required more then 20 hours on the MACH-2 supercomputer (https://www3.risc.jku.at/projects/mach2/). Some preliminary tests showed that the computation time is proportionate to k^6, which prevented us from using larger topologies in our tests. Similarly, the edge-disjoint shortest paths required for the `Square1` protocol took more than 15 minutes to compute for a single destination d. To avoid costly recomputation of these structures for multiple destination nodes, we only computed them once for some fixed destination d on the bottom layer. We then applied an isomorphism to map these structures towards the remaining possible destination nodes on the bottom layer.

B. Experiments

We conducted two different types of experiments, both in the Clos topology with $k = 80$ (consisting of 8000 nodes).

Experiment 1: Performance under all-to-one model. In the first experiment we examine the performance of the fast rerouting protocols under the all-to-one traffic pattern. Results are given in Fig. 1. Each simulation was started by first failing a p fraction of random edges. We then select a random node on the bottom layer and let each other bottom layer node send one unit of flow towards this destination. After the routing procedure is complete, we measure the maximum edge load as well as the average amount of hops required by any flows to reach the destination. To obtain the results for the plots in Fig. 1 we perform these simulations for increasing values of p (ranging from $p = 0.0$ to $p = 0.2$ in steps of 0.02), repeat the simulation for each p value 40 times and report the average of the resulting maximum edge load and hop values.

As we can see in the plot on the left-hand side of Fig. 8, all protocols besides A-CASA accumulate similar loads in case no edges are failed. We suspect that the worse behavior of A-CASA stems from the fact that the Clos network we consider is 40-connected. As explained in [18] this protocol works best when this value is a prime power. In our setting, this led to multiple arborescences being (almost) completely unused, which causes the load to be distributed unevenly. It is also important to note that the results for our Interval-D and Interval-ID protocols are reported till $p = 0.1$. We do this because a higher amount of edge failures causes forwarding loops to be created or prevents the routing strategy from working (nodes get disconnected from all nodes in the adjacent interval). We emphasize that this is related to the small value of $k = 80$ we consider in these experiments, which leads to intervals of size only 6 (see the description of the adaptation of our protocol in Section VI-A). For values of $k > 200$ we could not observe this behavior even when failing a $p > 0.25$ fraction of all edges. When increasing the amount of failed edges in the system, only the randomized A-PRNB protocol is able to compete with our ThreeP-ID protocol, which further illustrates the strength of randomized approaches when dealing with edge failures.

When looking at the right-hand side of Fig. 7 we can see the average number of hops required for packets to reach the destination. There we can observe three regimes. First, we have the Square1 protocol which performs the worst. While it starts from a near-optimal average hop count of roughly 4 (note that almost all source nodes in our all-to-one routing approach are 4 hops away from the destination), it increases more rapidly than the other approaches. This is an inherent weakness of this protocol, as each time a packet with source s encounters an edge failure on the way to d, it goes all the way back to s and attempts another route. In the second regime, we can observe all the aborescence based approaches. We think the reason that these protocols perform worse than our protocol is the following: assume that a packet traversing some arborescence T_i encounters a failed edge while being at node v. It will now continue from node v in another arborescence T_j. Now, it is possible that the position of v in T_i is much closer to the destination than in T_j and hence, this packet possibly needs to traverse a long path inside T_j even though it was close to the destination before switching to this tree. In contrast, our protocols avoid sending packets on a long detour. If a packet resides at distance x from the destination and cannot proceed closer due to a failed edge, it is forwarded between nodes in distance x and $x + 1$ until it is able to move to a node in distance $x - 1$.

Experiment 2: Performance under gravity model. We next consider the performance of the protocols under a *gravity model* [20], describing the demands between any pair of two nodes on the bottom layer of the Clos topology. We set the parameters of this model such that, in expectation, the demand between each such pair of nodes is 1. Throughout the routing process we then send a flow from each bottom node to every other bottom node (all-to-all). This flow is assigned a weight corresponding to the demand. For this experiment we also generalize our notion of node and edge load: load is now defined as the sum of all weights of flows that cross the node or edge. Besides these changes, the steps for generating the results of our second experiment are the same as in the first.

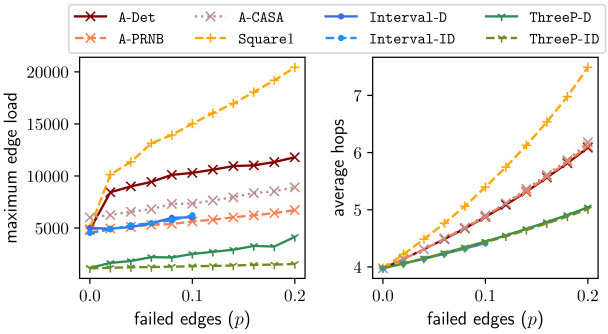

Fig. 8. Average maximum edge load and average amount of hops required to reach the destination when performing *gravity* routing.

As we can see on the left-hand side of Fig. 8 our ThreeP-D and ThreeP-ID protocols achieve lower maximum load than all other approaches (including our Interval-D and Interval-ID approaches). We suspect that this is because in the ThreeP-D and ThreeP-ID protocols, packets are forwarded according to random permutations that span over a larger amount of nodes. In the Interval-D and Interval-ID the selection of possible forwarding partners in any hop is more constrained. This advantage is further emphasized on in the ThreeP-ID protocol as in this protocol nodes are only guaranteed to make the same forwarding decisions for flows that arrive from the same inport and have the same destination. This makes it less likely for multiple flows to follow the same path. When it comes to the protocols of related work, we suspect the higher load to stem from the selection of the set of arborescences and also the selection of the set of shortest-paths for Square1. There exists some discussion around the efficient construction of a good set of aborescences for a fixed destination [53]. However, it seems to be an open question how to select such sets for multiple destinations with the goal of optimizing load in many-to-many traffic patterns. We found that these structures need to be sufficiently edge-disjoint from those used for other destinations. Otherwise, load imbalances occurred when performing gravity routing. In particular, we encountered a "bad" set of arborescences, which lead to an maximum node load of more than 200,000 in all arborescence-based protocols even if no edge is failed. In contrast, a better set of arborescences lead to a maximum node load of at most 40,000. For all of our experiments, we employed the best sets of arborescences we could create in order to minimize the load values w.r.t. all the approaches we considered.

On the right-hand side of Fig. 8 we can see the average number of hops required by the flows to reach their destination. From the point of view of any bottom node, the Clos topology looks exactly the same. Because of this inherent symmetry, the average hop values are very similar as in our all-to-one routing experiments. The reasoning for the three regimes of average hop values that can again be observed in this setting is the same as in the all-to-one experiments.

Takeaway: We observe in our experiments that the ThreeP-ID

protocols guarantee, both, the lowest maximum load as well as average hop count in both experiments. In case it is possible to match the source address, destination address, input and hop count in the packet header it is advisable to use this protocol. In case a slim set of forwarding decisions is required it makes sense to consider the `Interval-D` protocol. It still outperforms all approaches of related work considering the average hop count and only gets outperformed by `A-PRNB` when it comes to ensuring limited load. However, if resilience against a large amount of failures is required, then it is only advisable to utilize this strategy in topologies of sufficiently large size.

VII. Conclusion and Future Work

In our work we considered three different local failover protocols. Starting with the *3-Permutations* protocol, we presented a protocol which guarantees a load of at most $O(\log^2 n \log\log n)$ w.h.p. even if $\alpha \cdot n$ edges are failed for some constant $0 < \alpha < 1$. Next, we presented the *Intervals* protocol. While this protocol comes with slightly lower theoretical resilience of $O(n/\log n)$, it can also be used in settings where the hop count in the packet header cannot be matched. It achieves a maximum load of $O(\log n \log\log n)$ w.h.p. Finally, we presented a third approach, the *Shared-Permutations* protocol, which is mostly of theoretical interest. In case the nodes have access to some shared permutation, we show that the maximum load can be reduced to at most $O(\sqrt{\log n})$ w.h.p.

We also adapted two of the above approaches to the Clos topology with 3 layers and performed emperical tests. These tests indicate that our protocols ensure a low edge load in this more practical setting as well. The `ThreeP-ID` variant of the *3-Permutations* protocol even outperforms all related approaches when it comes to the maximum edge load. Additionally, all our adapted protocols ensure that packets reach their desired destination in less hops than in related approaches in case multiple edge failures occur. It remains an open question whether the above protocols can also be adopted to more general topologies.

Throughout all our theoretical results we assumed an *oblivious* adversary which selects the set of failed edges. However, some of our algorithms can be extended to deal also with *more adaptive adversaries*: to defeat adversaries who aim to infer network-internal loads (e.g., leveraging physical access or using tomographic techniques), we can regenerate random permutations periodically. That is, the *3-Permutations* and *Intervals* algorithms have the attractive property that they allow to regenerate such permutations quickly, locally, and without coordination: each node can independently regenerate random numbers over time to enhance security. This also allows our algorithms to recover if the low probability event occurs, in which higher loads than the ones specified in our theorems emerge.

There are also slight variations of our failure model which we did not fully cover in our analysis. For example, the case of a *lower amount of edge failures*. In our analysis, we assumed that the adversary destroys up to either linear or $O(n/\log n)$ many edges. We believe that a lower amount of edge failures affects the performance of the algorithms as follows. If $n^{1-\delta}$ edges are destroyed for some constant $\delta > 0$ it can be shown (by a slightly adapted repetition of the existing analysis) that all three of our algorithms guarantee w.h.p. a maximum load of $O(1)$ on most of the nodes.

Finally, it may make sense to analytically consider the case of *randomly selected edge failures* instead of assuming the existence of a malicious adversary. In the context of hardware failures or power outages, it might make sense to model the set of failed edges as selected u.a.r. out of all $\sim n^2/2$ edges. If only a small amount of edges is failed in this way (i.e., $O(n)$), then all but $O(\log n)$ nodes may forward their flows directly to the destination w.h.p. It can be shown that our algorithms then induce a congestion of $O(1)$ w.h.p. The question about a higher amount of edge failures and the expected resilience against such failures remains an open question.

References

[1] A. Atlas and A. Zinin, "Basic specification for IP fast reroute: Loop-free alternates," in *Request for Comments (RFC) 5286*, 2008.

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

[3] Switch Specification 1.3.1, "Openflow," 2013. [Online]. Available: https://bit.ly/2VjOO77

[4] P. François, C. Filsfils, A. Bashandy, B. Decraene, S. Litkowski *et al.*, "Topology independent fast reroute using segment routing," 2014.

[5] ISO, "Intermediate ststem-to-intermediate system (is-is) routing protocol," ISO/IEC 10589, 2002.

[6] J. Moy, "OSPF version 2," RFC editor, https://tools.ietf.org/html/rfc2328, RFC 2328, 1998.

[7] C. Filsfils, P. Mohapatra, J. Bettink, P. Dharwadkar, P. De Vriendt, Y. Tsier, V. Van Den Schrieck, O. Bonaventure, P. Francois *et al.*, "BGP prefix independent convergence," *Cisco, Tech. Rep*, 2011.

[8] A. Kabbani, B. Vamanan, J. Hasan, and F. Duchene, "Flowbender: Flow-level adaptive routing for improved latency and throughput in datacenter networks," in *Proc. of the 10th ACM International Conf. on emerging Networking Experiments and Technologies*, 2014, pp. 149–160.

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

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

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

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

[13] K.-T. Foerster, J. Hirvonen, Y.-A. Pignolet, S. Schmid, and G. Tredan, "On the feasibility of perfect resilience with local fast failover," in *Proc. SIAM APOCS*, 2021.

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

[15] M. Borokhovich, Y.-A. Pignolet, S. Schmid, and G. Tredan, "Load-optimal local fast rerouting for dense networks," *IEEE/ACM Transactions on Networking*, vol. 26, no. 6, pp. 2583–2597, 2018.

[16] H. Wu, Z. Feng, C. Guo, and Y. Zhang, "Ictcp: Incast congestion control for tcp in data-center networks," *IEEE/ACM transactions on networking*, vol. 21, no. 2, pp. 345–358, 2012.

[17] M. Handley, C. Raiciu, A. Agache, A. Voinescu, A. W. Moore, G. Antichi, and M. Wójcik, "Re-architecting datacenter networks and stacks for low latency and high performance," in *Proc. of ACM SIGCOMM*, 2017, pp. 29–42.

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

[19] G. Bankhamer, R. Elsaesser, and S. Schmid, "Local fast rerouting with low congestion: A randomized approach," in *Proc. 27th IEEE International Conference on Network Protocols (ICNP)*, 2020.

[20] M. Roughan, "Simplifying the synthesis of internet traffic matrices," *ACM SIGCOMM CCR*, vol. 35, no. 5, p. 93–96, 2005.

[21] F. Clad, "Disruption-free routing convergence: computing minimal link-state update sequences," Ph.D. dissertation, Strasbourg, 2014.

[22] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM CCR*, vol. 38, 2008.

[23] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano *et al.*, "Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network," *ACM SIGCOMM CCR*, vol. 45, no. 4, pp. 183–197, 2015.

[24] M. Chiesa, I. Nikolaevskiy, S. Mitrovic, A. V. Gurtov, A. Madry, M. Schapira, and S. Shenker, "On the resiliency of static forwarding tables," *IEEE/ACM Trans. Netw. (TON)*, vol. 25, pp. 1133–1146, 2017.

[25] J. Edmonds, "Edge-disjoint branchings," *Combinatorial algorithms*, 1973.

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

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

[28] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[29] M. Chiesa, A. Kamisinski, J. Rak, G. Retvari, and S. Schmid, "A survey of fast-recovery mechanisms in packet-switched networks," *IEEE Communications Surveys and Tutorials (COMST)*, 2021.

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

[31] A. J. González and B. E. Helvik, "Analysis of failures characteristics in the UNINETT IP backbone network," in *IEEE Inter. Conf. on Advanced Information Networking and Applications Workshops*, 2011.

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

[33] B. Yang, J. Liu, S. Shenker, J. Li, and K. Zheng, "Keep Forwarding: Towards k-link failure resilient routing," in *Proc. IEEE INFOCOM*, April 2014, pp. 1617–1625.

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

[35] J. L. Welch and J. E. Walter, "Link reversal algorithms," *Synthesis Lectures on Distributed Computing Theory*, vol. 2, pp. 1–103, 2011.

[36] M. Chiesa, A. Kamisiński, J. Rak, G. Rétvári, and S. Schmid, "Fast recovery mechanisms in the data plane," *TechRxiv*, 2020.

[37] M. Menth, M. Duelli, R. Martin, and J. Milbrandt, "Resilience analysis of packet-switched communication networks," *IEEE/ACM Transactions on Networking*, vol. 17, no. 6, pp. 1950–1963, 2009.

[38] S. Kini, S. Ramasubramanian, A. Kvalbein, and A. F. Hansen, "Fast recovery from dual-link or single-node failures in ip networks using tunneling," *IEEE/ACM Trans. Netw. (TON)*, vol. 18, 2010.

[39] R. Cohen and G. Nakibly, "Maximizing restorable throughput in MPLS networks," *IEEE/ACM Trans. Netw. (TON)*, vol. 18, 2009.

[40] J. Qiu, M. Gurusamy, K. C. Chua, and Y. Liu, "Local restoration with multiple spanning trees in metro ethernet networks," *IEEE/ACM Transactions On Networking*, vol. 19, no. 2, pp. 602–614, 2010.

[41] D. Wang and G. Li, "Efficient distributed bandwidth management for MPLS fast reroute," *IEEE/ACM Trans. Netw. (TON)*, 2008.

[42] K.-W. Kwong, L. Gao, R. Guérin, and Z.-L. Zhang, "On the feasibility and efficacy of protection routing in ip networks," *IEEE/ACM Transactions on Networking (TON)*, vol. 19, no. 5, pp. 1543–1556, 2011.

[43] F. Clad, P. Mérindol, J.-J. Pansiot, P. Francois, and O. Bonaventure, "Graceful convergence in link-state ip networks: A lightweight algorithm ensuring minimal operational impact," *IEEE/ACM Trans. Netw. (TON)*, vol. 22, no. 1, pp. 300–312, 2013.

[44] A. Kvalbein, A. F. Hansen, T. Cicic, S. Gjessing, and O. Lysne, "Multiple routing configurations for fast ip network recovery," *IEEE/ACM Trans. Netw. (TON)*, vol. 17, no. 2, pp. 473–486, 2008.

[45] S. Cho, T. Elhourani, and S. Ramasubramanian, "Independent directed acyclic graphs for resilient multipath routing," *IEEE/ACM Trans. Netw. (TON)*, vol. 20, no. 1, pp. 153–162, 2011.

[46] A. Gopalan and S. Ramasubramanian, "Multipath routing and dual link failure recovery in ip networks using three link-independent trees," in *IEEE ANTS*, 2011, pp. 1–6.

[47] T. Elhourani, A. Gopalan, and S. Ramasubramanian, "Ip fast rerouting for multi-link failures," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 3014–3025, 2016.

[48] K. Lakshminarayanan, M. Caesar, M. Rangan, T. Anderson, S. Shenker, and I. Stoica, "Achieving convergence-free routing using failure-carrying packets," *ACM SIGCOMM CCR*, vol. 37, no. 4, pp. 241–252, 2007.

[49] B. Stephens, A. L. Cox, and S. Rixner, "Scalable multi-failure fast failover via forwarding table compression," in *Proc ACM SOSR*, 2016.

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

[51] P. Francois and O. Bonaventure, "An evaluation of ip-based fast reroute techniques," in *Proc. of the ACM conference on emerging network experiment and technology*, 2005, pp. 244–245.

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

[53] K.-T. Foerster, A. Kamisinski, Y.-A. Pignolet, S. Schmid, and G. Tredan, "Bonsai: Efficient fast failover routing using small arborescences," in *Proc. 49th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2019.

[54] G. Bankhamer, R. Elsaesser, and S. Schmid, "Randomized local fast rerouting for datacenter networks with almost optimal congestion," in *Proc. 35th International Symposium on Distributed Computing (DISC)*, 2021, pp. 9:1–9:19.

[55] M. Raab and A. Steger, "Balls into bins – A simple and tight analysis," in *Randomization and Approximation Techniques in Computer Science*. Springer Berlin Heidelberg, 1998, pp. 159–170.

[56] B. Doerr, *Probabilistic Tools for the Analysis of Randomized Optimization Heuristics*. Springer, 2020.

[57] R. Motwani and P. Raghavan, *Randomized Algorithms*. Cambridge University Press, 1995.

[58] "Simulation source code," https://github.com/gbank/CLOS-Simulations.

Gregor Bankhamer received his Master's degree in Computer Science from the University of Salzburg, Austria, in 2018. Currently he is a Ph.D. student at the same university under supervision of Robert Elsässer. His research interests lie in the area of randomized and distributed algorithms. Currently, he works on randomized routing protocols and the plurality consensus problem.

Robert Elsässer is a Professor at the University of Salzburg, Austria. He received his MSc (1998) and PhD (2002) from the University of Paderborn, Germany. From 2003 to 2011, Robert Elsässer worked as a Junior Professor at the University of Paderborn. During 2005/06 he was a visiting scientist at the University of California at San Diego, USA, and in 2009/10 a visiting professor at the University of Freiburg, Germany. His research interests focus on parallel and distributed algorithms, as well as on the design and analysis of large networks.

Stefan Schmid is a Professor at TU Berlin, Germany. MSc (2004) and PhD (2008) from ETH Zurich, Switzerland, postdoc at TU Munich and the University of Paderborn (2009), senior research scientist at T-Labs in Berlin (2009 to 2015), Associate Professor at Aalborg University, Denmark (2015 to 2018), and Professor at the University of Vienna, Austria (2018 to 2021). His research currently revolves around self-adjusting networks (related to the ERC project AdjustNet) and resilient networks (related to his WWTF project WhatIf).