CASA: Congestion and Stretch Aware Static Fast Rerouting

Klaus-Tycho Foerster
Faculty of Computer Science
University of Vienna, Austria
klaus-tycho.foerster@univie.ac.at

Yvonne-Anne Pignolet
Dfinity
Switzerland
yvonneanne.pignolet@dfinity.org

Stefan Schmid
Faculty of Computer Science
University of Vienna, Austria
stefan_schmid@univie.ac.at

Gilles Tredan
LAAS-CNRS
France
tredan@laas.fr

***Abstract*—To meet the stringent requirements on the maximally tolerable disruptions of traffic under link failures, many communication networks feature some sort of static failover mechanism for fast rerouting. However, configuring such static failover mechanisms to achieve a high degree of robustness is known to be challenging, in particular when packet tagging or dynamic node state cannot be used. This paper initiates the systematic study of such local fast failover mechanisms which not only provide connectivity guarantees, even under multiple link failures, but also account for the *quality* of the resulting failover routes, with respect to *locality* (i.e., route length) and *congestion*. Failover quality has received less attention in the literature so far, yet it is increasingly important to support emerging applications.**

We first show that there exists an inherent tradeoff in terms of achievable locality and congestion of failover routes. We then present *CASA*, an algorithm providing a high degree of robustness as well as a provable quality of fast rerouting. *CASA* combines two crucial static resilient routing techniques: combinatorial designs and arc-disjoint arborescences. We complement our formal analysis with a simulation study, in which we compare our algorithms with the state-of-the-art in different scenarios and show benefits in terms of stretch, load, and resilience.

I. Introduction

Failures are the norm in large-scale communication networks, including data center [1], backbone [2] or enterprise [3] networks. As many of these networks have become a critical infrastructure of our society, ensuring a high degree of resilience and availability, even under multiple failures [4], [5], [6], is important. It is hence not surprising that network reliability is one of the main network carrier concerns [7], [8]

Reactive approaches that *re*compute new paths after learning about link failures are known to result in poor performance, packet loss, or even transiently inconsistent routes [9]. Reaction times are particularly high if reconvergence is *decentralized* (one of the motivations for Google's move to SDN [10]); but also if failures are handled centrally, e.g., at an SDN controller, the delay due to round-trip time and processing, can be undesirably high [11]. Proactive approaches try to overcome this problem by "preparing" the network for failures, e.g., by supporting the pre-installation of backup routes. We are hence particularly interested in *static resilience* mechanisms which rely on the pre-installation of (conditional) failover rules to forward packets, as they are supported by many networks (e.g., failover group tables in OpenFlow).

However, the computation of "good" failover rules poses a non-trivial algorithmic problem: it requires (algorithmic) maneuvers which, using local information about link failures only, ensure that traffic is steered to the intended destinations in a reliable manner. In particular, the failover mechanisms face the challenge that additional failures which occur downstream are not yet visible when forwarding decisions need to be taken. This requires the allocation of static resilient routes to be robust to additional failures and provide connectivity despite being oblivious to such failures. Today, it is still an open question [12], [13] whether being oblivious comes at a price, in the sense that static failover routing cannot leverage the full connectivity of the underlying physical network without a global view of all failures in the network.

While much existing work on failover routing focused on providing connectivity, *efficient* failover depends on additional criteria and especially on the resulting *load*: If a flow is rerouted onto a long path after a failure (i.e., does not preserve *locality* and has a large stretch), this results in an overhead in terms of bandwidth resources consumed along the failover path (*load sum*) and may also introduce a high latency. A high latency may also be introduced due to congestion on the resulting paths (*max load*). Accordingly, and in the light of emerging applications which come with more stringent availability, resource and latency requirements, we argue that these properties are gaining in importance. In this paper, we hence initiate the study of static resilient routing on multi-hop networks which accounts for locality and load. Our work is motivated by the following main observations:

1) High topological connectivity and path diversity alone may be insufficient to ensure an *efficient* failover, as failover routes need to be pre-installed and can only rely on local knowledge of link failures.
2) There is an inherent tradeoff between the congestion (load) and the length of failover routes (stretch). See Fig. 1 for an example. Accordingly, we investigate whether there exist efficient failover mechanisms finding a good tradeoff of the two criteria, where possible.

Contributions: We first derive non-trivial lower bounds of load and stretch for local rerouting schemes. We then present *CASA*[1] , a deterministic rerouting algorithm which relies on an intriguing combination of two crucial techniques for static resilient routing: *combinatorial designs* (so far only considered

[1] *CASA*is an acronym for *Congestion And Stretch Aware static fast rerouting*.

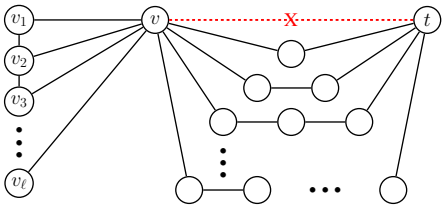

Fig. 1: 2-connected graph with l nodes on the left of v and l multi-hop paths between v and t. In this graph the load and stretch cannot be optimized at the same time if the dashed link between v and t fails. An algorithm optimizing for stretch would route the flows on short paths, inducing high load. On the other hand, an algorithm optimizing for load would distribute the flows over many (long) paths, resulting in high stretch factors.

in single-hop networks) and *arborescences* (known to provide connectivity, so far without load and stretch bounds).

We make the case for going beyond destination-based routing and leveraging also packet source addresses (e.g., the IP address, as usual in oblivious routing [14]): this, as we show, not only helps to reduce load but also to bypass Chiesa et al.'s conjecture [13]. To this end, we propose an algorithm called *SquareOne* that guarantees resilience to $k-1$ link failures in k-connected graphs using a very simple backtracking approach.

In addition to our formal analysis, we compare *CASA* to state-of-the-art algorithms in simulations, both on synthetic and real-world networks. Our simulation results highlight the benefits of our approach in terms of stretch, load, and resilience. In our simulations, *SquareOne* also proves to achieve a good performance in many practical scenarios.

Organization: The remainder of this paper is organized as follows. In Section II we introduce our system model and the problem we study. In Section III we present lower bounds for stretch and load. In Section IV we describe algorithms and their achieved bounds, followed by Section V where we evaluate our algorithms in simulations. After discussing related work in Section VI, we conclude in Section VII.

II. Model

The communication network is modeled as a graph $G = (V, E)$ connecting n nodes (switches, routers, hosts) V using undirected links E. We study k-connected networks which stay connected even after removing $k-1$ arbitrary links.

For routing decisions we assume that forwarding rules can match packet header fields as well as the in-port (the port from which a packet arrives at v, already used in [15] for fast reroute), and depending on this match, define the outgoing port to which a packet is forwarded at v. In other words, the focus of this paper is on *oblivious* (i.e., *static*) routing algorithms which do not rely on any dynamic state at nodes (e.g., counters) or in packets: we do not allow packet tagging. While tagging can improve the robustness of routing [16], [17], it is often undesirable in practice to change header fields.

The failover mechanism needs to be *statically preconfigured*: at the time the failover rules are installed, the set of link failures F is not known yet. The mechanism must be configured such that for any possible *local* link failures, a failover reaction is taken which provides connectivity, stretch and load guarantees *independently* of the additional failures that may be encountered downstream. It turns out that the ability to match in-ports can improve resilience and load of fast rerouting schemes. In particular, thanks to the possibility to match the in-port and the source of a packet, a node can be traversed multiple times during failover, without ending up in an infinite loop. That is, failover routes may not be simple paths but form *walks*, e.g., consider a network with a dead-end, forcing the packets to return along the same link [16].

Our goal is to devise r-resilient deterministic local rerouting algorithms with stretch and load guarantees. For worst-case load, we consider all-to-one traffic to a target node $t \in V$: every node communicates with unit demand to t. We study:

1) *Resilience (r):* A packet routed according to algorithm A *eventually* reaches t despite up to r link failures encountered on its walk to the destination. As such, only a finite number of loops (repeated visits of the same node) is allowed on the walk to the destination.
2) *Congestion resp. Load (ϕ):* Flow allocations are "load-balanced", minimizing the number of rerouted flows on links in G. That is, we aim to minimize $\phi = \max_{e \in E} \phi(e)$, where $\phi(e)$ describes the number of flows crossing link e due to rerouting.
3) *Locality resp. Stretch (s):* Flow allocations are "low-stretch", minimizing the (additive) detour packets take to reach their target: $s := \max_{v \in V} |W_{v,t}| - \mathrm{dist}(v,t)$, where $W_{v,t}$ describes the walk by the packet from v to t in the (re)routing algorithm A and $\mathrm{dist}(v,t)$ is the number of hops on a shortest path in the original failure-free graph G_0.

Simply put, we want to devise algorithms that maximize resilience while minimizing load and stretch. As a consequence, the resulting algorithms ensure a low latency overhead even under multiple link failures. To analyse the performance of a failover scheme in a network with f failed links (we express node failures in terms of the node's incident links which fail with it), we need some more definitions. In general, to study the limits of the failover scheme, we focus on worst-case performance: we assume the link failures are determined by an adversary knowing the resilient routing protocol.

Definition 1. *Let F be a set of failed links, $F \subset E$. A* worst case scenario *constitutes a set of failed links F that generate the worst load ϕ, chosen by an omniscient adversary knowing the failover scheme. $F_o(\phi)$ is defined as the set of* "optimal attacks" *(in terms of minimal required number of failures) leading to a load ϕ. That is, $\forall \phi \leq n, \forall F \in F_o(\phi)$, there is at least one (non-failed) link e such that the load $\phi(e)$ under a link failure set F is ϕ and there are no link failure sets smaller than $|F|$ generating the same load.*

Note that while we describe our algorithms in terms of conditional failover rules, our lower bounds and algorithms are general and apply to any static rerouting mechanism coping with local failure information.

III. Lower Bounds

We have seen above that requiring higher resilience may force algorithms to choose worse routes in term of load and

stretch. There are graphs where it is not possible to minimize load and stretch at the same time, e.g., on the graph depicted in Fig. 1. Next, we lower bound load and stretch individually.

A. A Load Lower Bound

We derive a non-trivial lower bound for oblivious routing schemes where routes do not only depend on the destination (but e.g., can also depend on the source) without requiring additional state maintained in nodes or packets.

Theorem 1. *For any local r-resilient failover scheme ($0 < r < k$) without disconnecting any source-destination pair, there exists a failure scenario which results in a link load of at least $\sqrt{r}$ on any graph.*

Proof: Let G be a k-connected graph, and let A be an r-resilient algorithm on G. Consider an all-to-one communication pattern where all nodes send messages to some node $t \in V$. Let the degree of the destination be $\deg(t) \geq k$, due to the fact that G is k-connected, and let $V_t = v_1, \ldots, v_{\deg(t)}$ be the direct neighbors of t. Observe that each of these neighbors sends a flow to t, for a total flow of $\deg(t)$ from the node set V_t. Observe also that since G is k-connected, there exist at least k link-disjoint paths between any two neighbors of t.

Let K be the graph induced from the subset $V_t \cup \{t\}$ of G. In addition we add for each pair $v_i, v_j \in V_t$ a link (v_i, v_j) to K representing the (multiple) link-disjoint options for v_i to transmit a message to v_j using paths from $G \setminus K$.

Observe that the resulting graph K is a complete graph on which an all-to-one communication pattern to t must be realized. This situation has already been addressed in [18], and the corresponding lower bound still holds: it is possible to generate $\sqrt{r}$ load for $r < k$ failures. We summarize the key steps of the proof in the next paragraph.

Pick a node v_i and its corresponding flow. Since A is correct it must lead the flow from v_i to t in the absence of failures. Let a_i^0 be the last link taken by this flow to reach t. Assume now a_i^0 has failed. Since A is $r \geq 1$ resilient, there must similarly exist a link used to reach t despite the failure of a_i^0. Let a_i^1 be this link. Observe that this process can be repeated r times for node v_i, but also for all the other nodes of V_t, providing us with a collection of A's strategical choices, which can be formalized as $(a_i^j)_{1 \leq i \leq \deg(t), 0 \leq j < r} \in \{(t, v_1), \ldots, (t, v_{\deg(t)})\}^{\deg(t) \times r}$. By carefully analyzing this set (that can conveniently be seen as a matrix), it is possible to identify at least one link that appears *early* (that is, before the $\sqrt{r}$ first failures) and *often* (that is, in $\sqrt{r}$ flows). Exposing this link for all those $\sqrt{r}$ flows creates a load of $\sqrt{r}$ and requires at most $(\sqrt{r})^2 = r$ failures. Let F be this strategy. Now observe that this strategy F has the same impact on G as on K. ∎

Interestingly, it has been proved in [18] that if a failover rule only depends on destination addresses, the situation is even worse, with linear load, even in the most densely connected graph, a clique. This shows that for low load it is crucial to use more than the destination for routing decisions. Accordingly, in the following, we will propose schemes which distribute the flows based on the source address as well.

B. A Stretch Lower Bound

Regarding the minimum additive stretch, we can derive the following lower bound, which generalizes and improves upon the girth-based results of [19]:

Theorem 2. *Consider any local failover scheme for a graph G with resilience r. Let $W_{G,t,r}$ be a shortest walk through nodes in $V \setminus \{t\}$ that contains $r+1$ neighbors of t. The additive stretch of the failover scheme is at least $\max_{t \in V} |W_{G,t,r}|$.*

Proof: For a deterministic r-resilient scheme, an adversary can fail r different links incident to the destination. Hence, any local scheme can be forced to visit at least $r+1$ neighbors of the destination, where the first r visited neighbors have failures on their connecting link to the destination, and only the $(r+1)$-th neighbor has a working connection to t. Note that such a described walk must exist, as the graph has a resilience of r, i.e., when an adversary uses its r failures incident to a destination t, all neighbors of t (and in general, all nodes, under any r failures) still form a connected component. ∎

For example, consider a local failover scheme with resilience 2 for 2-dimensional $x \times y$ torus graphs: for $x, y \geq 4$, a shortest walk through $2+1 = 3$ neighbors of any destination t has a length of exactly 4, i.e., its additive stretch is at least 4.

In general, finding a shortest path through some nodes is NP-hard, but it is tractable for many situations [20], e.g., for constant r, even under link capacity constraints [21].

IV. Upper Bounds and Algorithms

A. First Observations

We first observe that going beyond destination-based forwarding, and including also the source address (as it is usually performed in oblivious routing), cannot only reduce network load but also help to reach the maximum oblivious resilience possible, as desired in Chiesa et al. [13]. To this end, when a failure is encountered, we backtrack to the source and select another (pre-determined) disjoint path to the destination. We note that our proof relies on the source, see also [13, §4.2].

Theorem 3. *Given a k-connected graph, there is an oblivious deterministic $(k-1)$-resilient routing scheme.*

Proof: We prove the following claim for a single destination t: Let $r+1$ be the number of link-disjoint paths between a source v and a destination t; then there is an oblivious deterministic r-resilient routing scheme from v to t. The theorem then follows by a simple application for all source-destination pairs. Since the choice of one of the $r+1$ paths can depend on source, destination and in-port, and due to link-disjointness, nodes can determine which route packets were following so far. If they cannot forward the message towards the destination along the currently used path, either because the corresponding outgoing link has failed, or because the message has been received from the next node on this path, they send the message to their predecessor on the path from the source. Now, the message will be able to backtrack to the source (as the $r+1$ paths are not just arc-disjoint, but

link-disjoint), from where the next path is tried. Due to the fact that no link belongs to more than one of the $r+1$ paths from the source to the destination, the message will reach the destination if at most r links fail. ■

Note that such source-destination link-disjoint paths can be computed efficiently with, e.g., (min cost) flow formulations [22]. While the approach described in the proof of Theorem 3 provides optimal resilience, it does not consider stretch or load. In the remainder of this section we study resilient routing schemes with such additional guarantees.

Our algorithms will leverage *rooted spanning arborescences*, a known approach to implement robust routing [12], [23]. We will quickly revisit these concepts in the following. Let (u, v) denote a directed arc from node u to v. A directed subgraph T is an r-rooted spanning arborescence of G if (i) $r \in V(G)$, (ii) $V(T) = V(G)$, (iii) r is the only node without outgoing arcs and (iv), for each $v \in V \setminus \{r\}$, there exists a single directed path from v to r.

When it is clear from the context, we use the term 'arborescence' to refer to a t-rooted spanning arborescence, where t is the destination node. A set of arborescences $\mathcal{T} = \{T_1, \dots T_k\}$ are arc-disjoint if no pair of arborescences in $\mathcal{T}$ share common arcs, i.e., if $(u, v) \in E(T_i)$ then $(u, v) \notin E(T_j)$ for all $i \neq j$. It is known that k arc-disjoint arborescences exist in any k-connected graph [24] and can be computed efficiently [25].

Chiesa et al. [12] showed how decompositions of G into T can be used to define failover routes: If a packet encounters a failed link at node v, then v forwards the packet along a different arborescence T_j. The crucial question studied in this paper is which arborescence to use when a packet hits a failed link (i.e., a node where the next link to be used is unavailable). In the following, we say that a packet is routed *according to an arborescence* T_i if a packet is forwarded along the unique directed path of T_i towards the destination.

When choosing the next arborescence to be used in case of failure in an arbitrary *circular* order, we have a $(\lfloor k/2 \rfloor - 1)$-resilience for k arc-disjoint arborescences as each link will be used at most twice due to a failure.

B. Avoiding Unnecessary Load

We next introduce the ideas underlying *CASA*, which provides efficient failover routes even under multiple failures. In particular, we observe that by varying the order of the arborescences used, we can give load-balancing guarantees for arborescence-based routing as well. This helps avoid the following problem: when a failure incurs and all affected flows use the same rerouting arborescence, then the links of this arborescence can be overloaded.

For example, consider all flows that use the link (v, t), v being some neighbor of t in their default routes. For a destination of degree $\deg(t)$, there is such a link which carries $n/\deg(t)$ flows. If this link fails, and all these flows are rerouted to t via another neighbor of t, called v', then the link (v', t) will experience a load of $n/\deg(t)$ due to one single failure. On the other hand, a rerouting scheme that balances the load better, can achieve a maximal load of $n/\deg(t)/(\deg(t)-1)$ for the same scenario, by distributing the flows to all remaining neighbors of t.

A general way to represent the decisions for the next arborescence for oblivious routing schemes, which is also used to describe *CASA*, is to use an $n \times k$ matrix, containing the indices of the arborescences to be used as elements. Each row of this failover matrix is assigned to a source node. Once we have constructed such a matrix and the assignment of nodes to rows, we use them for the decision which arborescence to choose in the case of a failure:

C. Failover Matrix-based Arborescence Routing

Given a failover matrix, consider a packet arriving at node j, with in-port p originating from source node i. Each in-port corresponds to exactly one arborescence due to the fact that we use arc-disjoint arborescences. Thus node j can determine on which arborescence T_l the packet has been routed so far and look up the index of l in row i. If the link leaving node j of arborescences T_l has failed, the next available arborescence is selected in the order of the i^{th} row of the failover matrix after element l in this row. An illustration of this scheme is depicted in Fig. 2, corresponding to the pseudo-code in Algorithm 1.

Algorithm 1 *CASA*: Rerouting given a Failover Matrix M

Upon receiving a packet of flow i at node v:

1: **if** destination not reached yet, $t \neq v$ **then**
2: current arborescence T_l (determined by in-port)
3: **if** next hop on T_l is up **then** forward packet along T_l
4: **else** j = index of v in i^{th} row + 1, $m_{i,j-1} = l$.
5: **while** next hop on $T_{m_{i,j}}$ is down **do** $j = j + 1$
6: forward packet along $T_{m_{i,j}}$

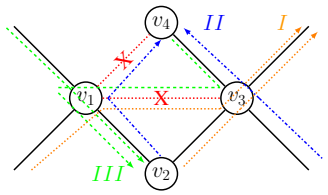

Fig. 2: A subgraph of four nodes and three arborescences. When a packet arrives at node v_1, the in-port determines which arborescence it follows at the moment. A packet arriving at v_1 from v_2 is currently following arborescence II (blue, dash-dotted). If the next link on this arborescence to v_4 is unavailable, then node v_1 can look up the matrix for the next arborescence to follow. If the next arborescence is I (orange, dotted) and the link to v_3 has failed too, it resorts to arborescence III (green, dashed) to v_2. Note that this allows flows that have been following the same arborescence to take different next hops when encountering a failure. E.g., depending on the source node, the next arborescence to be used can differ.

For arborescence routing, this means that we need to ensure that the arborescences used in case of failures are as diverse as possible. Towards this end, we apply a second key technique: we leverage ideas from *block design* using k arc-disjoint arborescences. For simplicity we first assume that k can be written as q^2+q+1 for some prime power q and we construct a latin failover matrix using a $(q^2+q+1, q+1, 1)$-BIBD as in [26]. While the (efficient) construction algorithm is immaterial for the following discussion, it is important to note

that each of the rows in this matrix is a permutation of k elements and that any two of its q-length prefixes of rows have exactly one element in common. We can then use this $k \times k$ matrix to build a $n \times k$ matrix, with one row for each node (and thus for each flow in the all-to-one traffic pattern). The steps to build this larger matrix are described in the proofs of the following theorems. Once we have this matrix, we use them to decided which arborescence to choose under failures.

As we will see in the following, the BIBD failover matrix construction guarantees low load, whereas the right selection of short arborescences ensures a low routing stretch.

Theorem 4. *For $\rho < \sqrt{k}/\lambda$ a circular rerouting scheme with k arborescences can optimally spread the use of the arborescences across λk flows leading to a load $\phi \leq \rho$ with $\Omega(\rho^2)$ failures with a BIBD failover matrix.*

Proof: As a first step, we prove that in all-to-one routing one of the most loaded arcs is incident to the destination under a minimal number of failures. Assuming the contrary, there is an arc from node u to v, $v \neq t$ carrying more load than the arcs incident to the destination. Together with the fact that this arc is used by exactly one arborescence, this implies that there must be another failure on this arborescence. Otherwise the load on the last arc would be at least as high as the load on the arc from u to v. Since we could hence omit this failure and still have the same load, we have reached a contradiction. Thus, we can focus on the arcs to the destination. If the degree of the destination is k, each of them is used by one arborescence. Thus, the arborescence chosen due to the last failure on the path to the destination determines the load. Thanks to the fact that we use circular arborescence routing, the order of the arborescences chosen for a flow does not affect the resilience.

Analogously to the proof of Theorem 1 of [26], k flows that select the next arborescence to route along according to the $(k \times k)$-BIBD-failover matrix ensures that an arborescence is used for rerouting ρ flows if at least $\Omega(\rho^2)$ links fail. Due to the lower bound proved earlier, this is optimal.

This can be extended for more than k flows, parametrized by the number of failures to be tolerated:. for more than k flows we cannot construct a latin submatrix where the first $\sqrt{k}$ rows intersect in exactly one element, as we have only k elements to fill the matrix with. However, when maintaining low intersection size, we can keep the number of failures needed for arborescence reuse high. Consider the case when each element can occur twice in each column, but the pairwise intersection of the first l elements of two rows is still one. In this case we can use the same arguments as above to show that the necessary number of failures is quadratic in the resulting load, if the maximum number of failures affecting any arborescence is at most $2l$.

Hence we can split BIBD blocks into smaller blocks and use the BIBD failover construction to build matrices for more flows with the same asymptotic load behavior, albeit tolerating fewer failures for the guarantees. More precisely, given a $(q^2+q+1, q+1, 1)$-BIBD for $q = \sqrt{k}$ we can construct a $(2^{3\log n/2 - \log\lambda}, \lambda, 1)$-BIBD by partitioning each block into $(q+1)/\lambda$ disjoint subblocks. With the smaller blocks, we can use the same approach as before, for λ times more flows. ■

The scheme can be generalized for values of k which are not q^2+q+1 for some prime power q by selecting a suitable power of 2 and extending the resulting BIBD matrix with repeated rows and permutations of the remaining elements like in [26] to a $k \times k$ matrix. The subsequent steps are the same and the constants in Theorem 4 increase by at most a factor of four.

Note that the circular routing scheme retains its $(\lfloor k/2 \rfloor - 1)$-resilience, regardless of the use of the BIBD scheme. However, the load bound only holds for up to $k/(2\lambda^2)$ failures.

Instead of reducing the fault tolerance, we can also use the same rows for multiple flows. In other words, two or more flows use the same sequence of arborescences for failover and their load increases. More precisely, this option leads to a linear increase in the joint use of the same arborescences. Hence we can trade-off between paying a re-use factor or lowering the failure resistance as we please.

Corollary 1. *A rerouting scheme with k arborescences can spread the use of the arborescences across λk flows for $|F| < \lfloor k/2 \rfloor$ failures incurring a re-use of $\lambda\sqrt{|F|}$.*

D. Adding Locality to the Picture

Next, we investigate how to control the stretch. If we have multiple shortest paths, we can reach optimal resilience and stretch. However, a proof by contradiction shows that only one shortest path arborescence may exist. Hence, we need to select arborescences with low depth. While they do not guarantee shortest paths, we can use them to bound the stretch.

For k *independent* arborescences (no common nodes on paths from v to d on different arborescences), the proof of Theorem 3 can be adapted to circular arborescence routing.

Theorem 5. *Given k independent arborescences of maximal depth d, any order of following the arborescences with backtracking is $(k-1)$-resilient with an additive stretch of $d|F|$.*

Proof: Due to the independence property no link belongs to more than one of the k paths from the source s to the destination t, thus a message reaches the destination if at most k links fail. Moreover, due to the arborescences' depth each path from s to t is at most d hops long and at most $|F|$ paths are tried out, thus the stretch is at most $d|F|$. ■

However, sometimes one can find arborescences where a better bound can be shown. For example, Yang et al. [27] describe a construction of k rooted Independent Spanning Trees (ISTs) for the k-dimensional hypercube, with an additional property: for any node $v \neq t$, the path from v to the child of t in T_i is a shortest path in G. Such ISTs are called *optimal* and can be used for algorithms with resilience and stretch bounds.

Lemma 1. *Given a set of k independent spanning trees $\mathcal{T} = \{T_1, \ldots, T_k\}$ rooted at t, the following is an arc-disjoint arborescence set $\mathcal{T}' = \{T'_1, \ldots, T'_k\}$: $\forall\{u, v\} \in T_i$, where v is closer to the root t on T_i than u, add an arc (u, v) to T'_i.*

Proof: Due to the fact that T_i does not contain any undirected cycles, the link set T_i' cannot contain any directed cycles, thus we have a set of arborescences. To prove their arc-disjointness, assume for the sake of contradiction, that there are two arborescences sharing arc (u, v). This is not possible as it would violate the independence property of the IST. ■

Theorem 6. *Given a set of k optimal ISTs, circular arborescence routing is $(\lfloor k/2 \rfloor - 1)$-resilient with additive stretch $2a|F|$, where a is the maximal distance between two neighbors of t in the graph without t.*

Proof: Using ISTs as arc-disjoint arborescences, circular arborescence routing guarantees that packets reach the destination if at most $\lfloor k/2 \rfloor - 1$ links fail. Whenever a failure leads to change the currently used arborescence, this implies that shortest paths to a different neighbor of the destination are used. As the distance of these neighbors is at most a by definition, a detour of at most a is added for each change of arborescences. $|F|$ failures can thus cause up to $2|F|$ arborescence changes. Hence, the stretch is at most $2a|F|$. ■

In addition to the hypercubes mentioned earlier, there are also other graphs for which constructions of optimal ISTs have been found. Among them are cartesian products of complete graphs [28], hybrid graphs [29], even [30] and odd graphs [31].

For arc-disjoint arborescences without the optimal path lengths to neighbors of the destination, a weaker result holds.

Theorem 7. *Given a set of k arc-disjoint arborescences of maximal depth d, circular arborescence routing is $(\lfloor k/2 \rfloor - 1)$-resilient with additive stretch $2d|F|$.*

Proof: Whenever the arborescence is changed due to a failed link, an additional detour of at most d has to be taken into account. As $|F|$ failed links can cause $2|F|$ arborescence changes, the total stretch is bounded by $2d|F|$. ■

E. Stretch-Load Tradeoff

Having derived the main properties of *CASA*, we next discuss a fundamental tradeoff that any static failover algorithm faces. The order in which arborescences are chosen when encountering failures (i.e., to ensure resilience) has an impact on both load and stretch. More precisely, the two objectives contradict. To illustrate this, first consider a node v that reaches t with a low stretch in arborescence T. Then T must be low stretch for all nodes between v and t in T also have a low stretch in T. This hints that selecting always the lowest stretch backup tree T will lead to a high load on T. To avoid this load, some nodes must deliberately choose another T' with longer paths to t even though T is available.

This situation can be generalized: imagine a set of trees $\mathcal{T} = T_1, \ldots T_k$ sorted by increasing stretch for a node v and such that the stretch using T_i is, say, σ_i. Consider a partition of these arborescences into two sets of low stretch $L = \{T_i, i \leq k/2\}$ and high stretch $H = \mathcal{T} \setminus H$. Imagine a strategy consisting of exploiting first the trees of L and then only the trees of H. This strategy provides a $2\sqrt{k/2} = \sqrt{2k}$ worst case load, for a worst case stretch of $\sigma_{k/2}$ due to the use of the $T_{k/2}$ tree. On the other hand, a strategy that directly uses all $\mathcal{T}$ trees would provide a better load guarantees of $\sqrt{k}$ at the expense of a worse stretch of σ_k due to the assignment of T_k to some flow.

V. Experimental Evaluation

In order to complement the formal analysis of *CASA* and the worst-case guarantees derived above, we conducted an extensive simulation study in which we investigate stretch, load, and resilience properties in different scenarios and compared to existing algorithms. More precisely, we measure the performance of adding a BIBD failover matrix to circular arborescence routing (see the *CASA* Algorithm 1) and of the link-disjoint backtracking scheme from Theorem 3, which we denote as *SquareOne*. For *SquareOne* we iterate through the paths by lengths, setting the shortest one as the default route.

For comparison, we also include the circular arborescence failover scheme from Chiesa et al. [32] (also in [16]). In particular, this allows us to judge the practical impact of *CASA*'s approach of combining BIBDs with circular arborescence routing. Similarly, we furthermore include their randomized arborescence scheme without bouncing [12], to have a good load competitor, e.g., for worst-case failure scenarios[2].

Similar to the evaluation in [32], we evaluate all four algorithms on one hundred 8-connected 8-regular random networks (RR) with 100 routers each[3] (each node as a destination, 10K experiments), for all-to-one traffic with randomly picked destinations. The eccentricity is between 3 and 4 and the average shortest path around 2.4. Furthermore, we also study well-connected cores of various autonomous systems [33], see Table I (again, each node as a destination, 1583 experiments).

AS	1239 A	2914 B	3356 C	7018 D
Number of nodes	389	225	377	204
Number of links	3621	1696	4736	1667
Eccentricity	6	6	6	6
Avg shortest path length	3.06	2.48	3.14	3.17

TABLE I: Properties of 8-connected cores of various ASes

We study two failure scenarios: (i) In the *random scenario*, we fail links uniformly at random, in numbers also greatly beyond the connectivity of the network. Hence, some nodes might be disconnected[4], but usually many paths to the destination remain. (ii) In the *targeted scenario*, links directly connected to the destination fail. Thus, a failover scheme might use an expensive rerouting in the last moment.

In our plots, we mark the algorithms as *CASA* (Algorithm 1), *SquareOne* (Theorem 3), *DetCirc* ([32]), and *PRNB* ([12]). The stretch and load is measured with respect to the situation in a failure-free network, showing the multiplicative stretch respective load overhead induced by the failed links.

[2] We also evaluated their randomized scheme with bouncing, it performed like its no-bouncing alternative. [3] We note that all schemes only required minimal preprocessing to compute the routing rules on the random graphs, $<$ 1s for *SquareOne*, and up to 2s for the arborescence decompositions. [4] which we then omit as sources

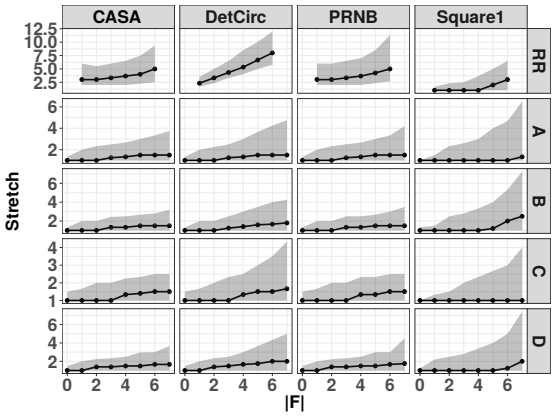

Fig. 3: Plots of the *stretch* for *targeted* link failures F.

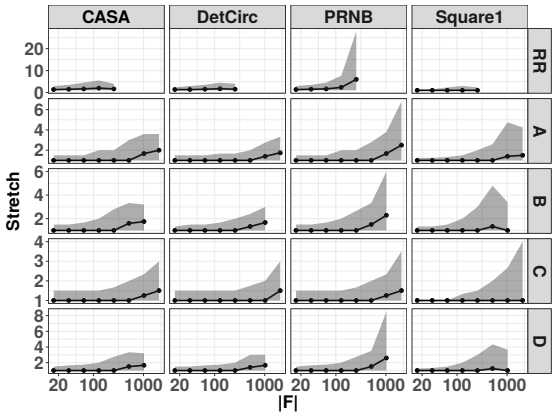

Fig. 4: Plots of the *stretch* for *random* link failures F.

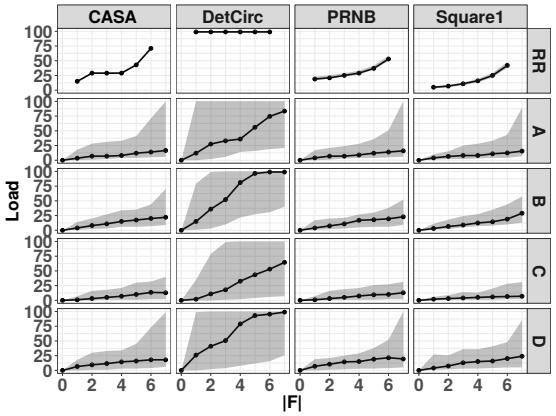

Fig. 5: Plots of the *load* for *targeted* link failures F.

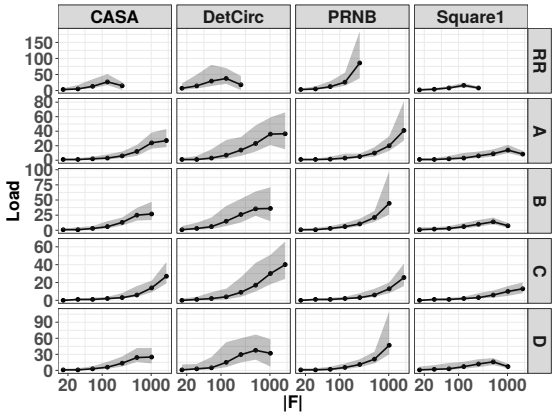

Fig. 6: Plots of the *load* for *random* link failures F.

A. Stretch Performance

We start by discussing the stretch of all four algorithms, plotted in Fig. 3 and 4: the median values are shown as dots, whereas the grey-shaded area depicts the values between the 10th and 90th percentiles.

1) Targeted failures (Fig. 3): Regarding the median stretch values, *CASA*, *DetCirc*, and *PRNB* perform quite similar, with *DetCirc* being slightly worse on random graphs. Our backtracking scheme *SquareOne* is significantly better, reaching mostly median stretch values of just 1, which might seem counter-intuitive at first. However, up to 4 link failures will only affect up to half of the sources in *SquareOne*, i.e., half of the network can still route optimally. Considering the 10% and 90% percentiles, *CASA* and *PRNB* perform best on the ASes (with a small bad outlier for *PRNB* on random graphs), with *DetCirc* being close, except for larger failure values. Note that these numbers are much better than the upper bounds derived in Theorem 7, as the depth d of the arborescences produced in the evalution is between 10 and 24, with a median of 15. *SquareOne* is more impacted in the stretch variance, especially for larger failure numbers. Interestingly, on random graphs, *SquareOne* performs best regarding the variance, though only slightly better than BIBD.

2) Randomized failures (Fig. 4): A large number of failures (mostly over a 100) is required for the median stretch values to go beyond 1, where all four schemes behave roughly identical. In the 10th to 90th percentiles, *DetCirc* slightly outperforms BIBD and *PRNB*, where *PRNB* again has several outliers for large failure numbers. *SquareOne* outperforms the other schemes for failures up to roughly 100, but for larger failure numbers, the variance can be nearly twice as large.

B. Load Performance

The load results for all algorithms are shown in Fig. 5 and 6, where again the median values are shown as dots and the 10th to 90th percentile in gray-shading.

1) Targeted failures (Fig. 5): For random graphs, already a single failure induces worst-case load for *DetCirc*, as each of the eight arborescences only has a single link connected to the destination. The cores of the four ASes are usually better connected, i.e., failing a single link then only reroutes a part of the traffic along the first arborescence. Notwithstanding, *DetCirc* performs worst under adversarial failures. The performance of *CASA*, *PRNB*, and *SquareOne* is roughly similar, though *SquareOne* is slightly better than *PRNB*, which in turn is slightly better than *CASA*. *SquareOne* benefits from the fact that its walks to the destination do not use the last hops in the same order as *DetCirc* and is not restricted to arborescences, hence the load can be distributed among more links. The random nature of picking arborescences in *PRNB* is advantageous in comparison to the *deterministic CASA*, though the randomness of *PRNB* comes with practical implementation difficulties, as we will discuss later.

2) Randomized failures (Fig. 6): Similar to targeted failures, *DetCirc* performs again worst under load considerations. However, the difference is not as large, as failures upstream actually help spreading the load over multiple arborescences for *DetCirc*. The performance order of the three other algorithms remains as under targeted failures, i.e., first *SquareOne*, followed by *PRNB* and *CASA*, due to the reasons outlined above. Especially for few failures, the load overhead is low. In many graphs the last data point shows less load than the points before. This is due to the fact that the resilience of the scheme is reached, as confirmed in the next subsection.

C. Resilience

The resilience is shown in Fig. 7, plotting results for random link failures (for targeted failures, scenarios, the success rate is 100% four the plotted data points).

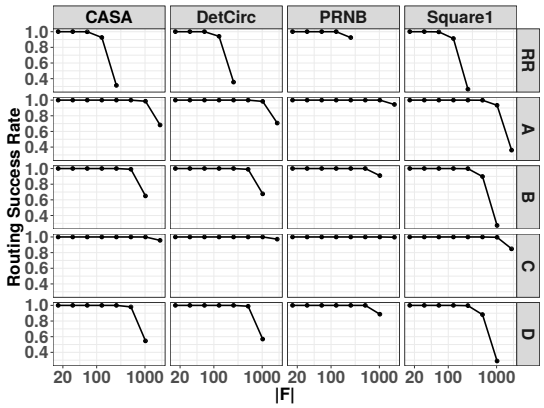

Fig. 7: Plots of the *resilience* for *random* link failures F.

Except for the most extreme failure settings, all schemes route successfully. For the highest link failure settings, (100-1000 failures), the *PRNB* scheme performs best. *DetCirc* and *CASA* perform similarly, except on random graphs: here, when large parts of the network fail, the packets can end up in permanent loops, though with a success rate above 90%.

The backtracking scheme *SquareOne* performs worst for extreme failure scenarios, but the success rate remains over 80%. The reason is that already 8 unlucky failures can disconnect all 8 routes, which becomes more likely when removing hundreds of links. Arborescences, however, are more resilient, as failures upstream do not affect links downstream.

D. Discussion

Briefly summarizing our evaluation results, we observe that *CASA* and *PRNB* [12] both perform quite well, outclassing *DetCirc* in nearly all scenarios. An especially large gap can be seen in targeted failure scenarios, where the deterministic nature of *DetCirc* yields high stretch- and load-values. We note that *CASA* is of course deterministic as well, but our BIBD failover construction can be thought of as a derandomization.

- *CASA* performs similarly to *PRNB*, where *PRNB* however faces practical performance obstacles. A major drawback of the *PRNB* approach presented in [12] is that the randomized forwarding can lead to a high number of packet reorderings and hence low throughput.[5] *CASA* on the other hand delivers packets on deterministic routes, for any failure scenario. Notwithstanding, the truly random nature of *PRNB* leads to higher resilience results once hundreds of links failed in our experiments, albeit at the cost of high load and stretch.
- Furthermore, *CASA* provides *theoretical* performance guarantees, in particular for load, which are absent in *DetCirc*, *PRNB*, and *SquareOne*. Nonetheless, *SquareOne* also performs quite well in our simulations, slightly outperforming *CASA* and *PRNB* for roughly 4-5 targeted or up to ≈ 100 random failures, from which on the situation is reversed. It would thus be interesting to see how *SquareOne* could be extended to give theoretical performance guarantees, as in its current state it is a (apparently well-working) heuristic.
- We used unit load and all-to-one traffic as a reference scenario for the load evaluation to illustrate worst case conditions for unknown traffic patterns. Otherwise, our approach can be applied to optimize for elephant flows.

VI. Related Work

Routing mechanisms which tolerate *multiple* failures have been studied intensively in the literature already, and a most well-known technique relies on link reversals [34]. However, link reversal algorithms require dynamic tables which are not always supported, and they also introduce non-trivial delays of up to $\Omega(|V|^2)$ [35]. Some schemes also exploit packet-header rewriting [23], [36], [37], [38] or packet-duplication [39]. However, the former consumes header space and the latter introduces additional loads, which is undesirable. Another approach is to monitor TCP flows [40] or to pre-compute multiple flow paths s.t. in the event of failures, the ingress switches can rescale the traffic load efficiently without additional computational overhead [41]. Notwithstanding, The packets currently en route are not protected by such schemes.

Designing fast failover mechanisms which do not rely on packet marking however is challenging, and our model is closely related to the papers by Feigenbaum et al. [11], Chiesa et al. [12], [32], Elhourani et al. [23] and Stephens et al. [42], [43] which all study reachability even under multiple failures. In contrast to our work, however, these papers do not account for performance and load aspects of the computed failover paths (and at best provide only trivial stretch guarantees).

The work of Foerster et al. [19] provides stretch guarantees for some special graph classes, such as Hypercubes, Tori, Grids, and Clos-/BCube-topologies. Additionally, the authors provide stretch bounds based on the network's girth. However, their rerouting techniques do not take load into account and do not provide any worst-case guarantees accordingly. To the best of our knowledge, the only works considering load so far are by Borokhovich et al. [18] and Pignolet et al. [26], combined in [44]. Pignolet et al. [26] establish an interesting connection to distributed computing problems without communication,

[5] Additionally, *PRNB* [12] needs to perform *truly random decisions*, as packets can return to a node with identical source, destination, and in-port fields. One solution could be to extend the hash function to non-immutable fields such as the TTL, which requires modifications and uses header space.

and in particular the results by Malewicz et al. [45]. However, this line of research focuses on single-hop topologies only. The extension to multi-hop networks is practically important and was left as the main open question. Our paper suggests that this extension is also non-trivial. Nevertheless, in our work we can leverage (and successfully combine) two key concepts introduced in prior work, namely combinatorial designs (due to Pignolet et al. [26]) and arborescence decompositions (due to Elhourani et al. [23] and Chiesa et al. [12], [32]).

VII. Conclusion

We argued that the performance of a routing scheme should be measured not only in terms of 1) fault-tolerance (resilience), but also with respect to the resulting 2) congestion (load), and 3) locality (stretch). We then presented two algorithms, *SquareOne* and *CASA*, which perform well for all three measures, as we show in extensive simulations. Our main contribution, *CASA*, even provides provable guarantees.

Our work opens interesting research avenues. In particular, it would be interesting to refine our algorithms and bounds toward more specific network topologies arising in different contexts (e.g., data centers). Moreover, once could explore randomized algorithms with probabilistic guarantees.

Acknowledgements We would like to thank Marco Chiesa and Ilya Nikolaevskiy for their support and helpful discussions.

References

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

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

[3] A. Shaikh, C. Isett, A. Greenberg, M. Roughan, and J. Gottlieb, "A case study of ospf behavior in a large enterprise network," in *Proc. IMW Workshop at SIGCOMM*. ACM, 2002, pp. 217–230.

[4] J. Tapolcai, B. Vass, Z. Heszberger, J. Bıró, D. Hay, F. A. Kuipers, and L. Rónyai, "A tractable stochastic model of correlated link failures caused by disasters," in *Proc. IEEE INFOCOM*, 2018.

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

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

[7] Telemark, "Survey," in *Web http://www.telemarkservices.com/*, 2006.

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

[9] J. Liu, A. Panda, A. Singla, B. Godfrey, M. Schapira, and S. Shenker, "Ensuring connectivity via data plane mechanisms," *Proc. NSDI*, 2013.

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

[11] J. Feigenbaum, B. Godfrey, A. Panda, M. Schapira, S. Shenker, and A. Singla, "Ba: On the resilience of routing tables," *PODC*, 2012.

[12] 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.

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

[14] Y. Azar, E. Cohen, A. Fiat, H. Kaplan, and H. Racke, "Optimal oblivious routing in polynomial time," in *Proc. STOC*, 2003.

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

[16] M. Chiesa, I. Nikolaevskiy, S. Mitrovic, A. V. Gurtov, A. Madry, M. Schapira, and S. Shenker, "On the resiliency of static forwarding tables," *IEEE/ACM Trans. Netw.*, vol. 25, no. 2, pp. 1133–1146, 2017.

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

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

[19] K.-T. Foerster, Y.-A. Pignolet, S. Schmid, and G. Tredan, "Local fast failover routing with low stretch," *ACM SIGCOMM CCR*, vol. 1, pp. 35–41, Jan. 2018.

[20] P. N. Klein and D. Marx, "A subexponential parameterized algorithm for subset tsp on planar graphs," in *Proc. SODA*, 2014.

[21] S. Akhoondian Amiri, K.-T. Foerster, and S. Schmid, "Walking through waypoints," in *Proc. LATIN*, 2018.

[22] B. Korte and J. Vygen, *Combinatorial Optimization: Theory and Algorithms*, 5th ed. Springer, 2012.

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

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

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

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

[27] J. Yang, S. Tang, J. Chang, and Y. Wang, "Parallel construction of optimal independent spanning trees on hypercubes," *Parallel Computing*, vol. 33, no. 1, pp. 73–79, 2007.

[28] X.-B. Chen, "Parallel construction of optimal independent spanning trees on cartesian product of complete graphs," *Information Processing Letters*, vol. 111, no. 5, pp. 235–238, 2011.

[29] J.-S. Yang and J.-M. Chang, "Optimal independent spanning trees on cartesian product of hybrid graphs," *The Computer Journal*, vol. 57, no. 1, pp. 93–99, 2014.

[30] J.-S. Kim, H.-O. Lee, E. Cheng, and L. Lipták, "Independent spanning trees on even networks," *Information Sciences*, vol. 181, no. 13, pp. 2892–2905, 2011.

[31] ——, "Optimal independent spanning trees on odd graphs," *The Journal of Supercomputing*, vol. 56, no. 2, pp. 212–225, 2011.

[32] 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.

[33] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson, "Measuring ISP topologies with rocketfuel," *IEEE/ACM Trans. Netw.*, vol. 12, no. 1, pp. 2–16, 2004, http://research.cs.washington.edu/networking/rocketfuel/.

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

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

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

[37] K.-T. Foerster, M. Parham, M. Chiesa, and S. Schmid, "TI-MFA: keep calm and reroute segments fast," in *Proc. IEEE Global Internet Symposium*, 2018.

[38] K.-T. Foerster, M. Parham, S. Schmid, and T. Wen, "Local fast segment rerouting on hypercubes," in *Proc. OPODIS*, 2018.

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

[40] T. Holterbach, E. C. Molero, M. Apostolaki, A. Dainotti, S. Vissicchio, and L. Vanbever, "Blink: Fast connectivity recovery entirely in the data plane," in *Proc. NSDI*, 2019.

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

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

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

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

[45] G. Malewicz, A. Russell, and A. A. Shvartsman, "Distributed scheduling for disconnected cooperation," *Distr. Comp.*, vol. 18(6), pp. 409–420, 2005.