Load-Optimal Local Fast Rerouting for Dense Networks

Michael Borokhovich[1] Yvonne-Anne Pignolet[2] Stefan Schmid[3] Gilles Tredan[4]
[1] AT&T Labs [2] ABB Corporate Research, Switzerland [3] University of Vienna, Austria [4] CNRS-LAAS, France

***Abstract*—Reliable and highly available computer networks must implement resilient fast rerouting mechanisms: upon a link or node failure, an alternative route is determined quickly, without involving the network control plane. Designing such fast failover mechanisms capable of dealing with multiple concurrent failures however is challenging, as failover rules need to be installed *proactively*, i.e., ahead of time, without knowledge of the actual failures happening at runtime. Indeed, only little is known today about the design of resilient routing algorithms.**

This paper introduces a general framework to reason about and design local failover algorithms which minimize the resulting load after failover on dense networks, beyond destination-based routing. We show that due to the inherent locality of the failover decisions at runtime, the problem is fundamentally related to the field of distributed algorithms *without coordination*. We derive an intriguing lower bound on the inherent network load overhead any local fast failover scheme will introduce in the worst-case, even though globally seen, much more balanced traffic allocations exist. We then present different randomized and deterministic failover algorithms and analyze their overhead load. In particular, we build upon the theory of *combinatorial designs* and develop a novel deterministic failover mechanism based on *symmetric block design theory* which tolerates a maximal number of link failures while ensuring low loads.

I. Introduction

Computer networks have become a critical infrastructure of our information society. Accordingly, there are increasingly stringent requirements on such networks, especially regarding dependability (availability and fault-tolerance).

The Context: The Need for Fast Failover. The ability to quickly recover from failures is a key requirement for dependable computer networks. Especially link failures are common and frequent today [20], and link failures do happen concurrently [2], [6], [8]. Even without physically disconnecting the underlying topology, these link failures can cause *routing failures* disrupting communications between some hosts.

The Problem: Slow Coordination. The reconvergence times in traditional routing systems after failures are known to be high. In a nutshell, in these traditional routing systems, whenever a link or node fails, routing tables are recomputed by executing the (distributed) routing protocol again. These recomputations result in relatively long outages after failures, leading to high packet loss rates [36].

While recent advances in routers have reduced reconvergence times, they are still too high for critical services which are sensitive to periods of traffic loss that are orders of magnitude shorter than this.

The Solution: No Coordination. Modern computer networks hence include *pre-computed* backup routes and rules for fast failover, allowing for very fast failure detection and re-routing. These local inband re-routing mechanisms are often meant as a first line of defense, and the resulting fast but simple rerouting is just a temporary solution, before the control plane rigorously optimizes the flow allocation for the new network topology. A most well-known example is *Fast Reroute* in MPLS (e.g., [26]) where, upon a link failure, packets are sent along a precomputed alternate path without waiting for the global recomputation of routes.

Another example, particularly relevant in datacenters [29], are failover schemes based on ECMP: when a link is detected to be unavailable (e.g., using LLDP neighbor discovery), flows are load-balanced (i.e., re-hashed) among the remaining shortest paths.

These mechanisms avoid the complexities involved in distributed coordination among switches or routers, but are completely local approaches: the reaction of a router only depends on the status of its incident links, and a router does not inform other routers about failures. In this case, the disruption time can be limited to the small time taken to detect the adjacent failure and invoke the backup routes.

The Challenge: Multiple Failures. The challenge of designing resilient local fast rerouting mechanisms is that these mechanisms need to rely on local knowledge only: in contrast to dynamic routing tables which may change in response to link failures (e.g., using link reversals [11]), failover routing tables are usually *statically preconfigured*. However, rerouting traffic along efficient paths based on local decisions only is challenging in the presence of *multiple failures*: a real and frequently studied threat, also in datacenters, e.g., due to shared risk link groups [28] (see also RFC 8001), attacks [33], network virtualization, cascading overload, or simply node failures which affect all incident links [2], [9], [12], [27], [21], [17].

Things become even more difficult if packet tagging (i.e., keeping information about observed failures along the packet trajectory in the packet header itself) is unavailable or undesired: while including information in the packet header can be used to keep track of observed failures along the path of the specific packet [3], tagging comes with overheads (in terms of header space, additional rules, and time) and can also cause problems in the presence of middleboxes [24].

In this paper we focus on simple routing algorithms, which do not require any dynamic state in the packet header nor at the routers themselves. In particular, we consider the well-established *oblivious* (i.e., non-adaptive) routing model [25].

The fundamental question is then [6]: how resilient can static forwarding tables be? That is, how many link failures can failover routing tolerate before connectivity is interrupted (i.e., packets are trapped in a forwarding loop, or hit a dead end) without invoking the control plane or using tagging? At first sight, it seems difficult to implement a high degree of fault-tolerance in a setting where routers are restricted to pre-configured failover rules, have a local view, and cannot resort to packet tagging. Furthermore, mechanisms based on ECMP are limited to failover to shortest paths [17].

Our Contributions. This paper presents the design of very resilient fast failover schemes for dense networks (as they appear most prominently in datacenters), tolerating multiple link and node failures, while keeping the network load low: asymptotically matching an intriguing lower bound stating that due to the inherent locality of the failover scheme, a certain load cannot be avoided, even though seen globally, better solutions would exist. Our re-routing algorithms take the source and destination of a flow into account, and do not require packet tagging.

We formally prove that our failover schemes provide an *optimal* resilience (in the sense that no scheme can tolerate more link failures) while minimizing link loads for many important traffic models, including the frequently studied permutation routing model [25], [35] or all-to-one routing [18].

Our approach is based on the insight that resilient local failover mechanisms can be seen as distributed algorithms without coordination: a subfield of distributed computing where devices solve a problem in parallel without exchanging information among them. In particular, we establish a connection to *combinatorial design* theory [32] and present a novel failover mechanism building upon *symmetric block designs*.

We focus on oblivious routing, where all packets of the same (micro-)flow will be forwarded the same way (namely based on source and destination only). However, we conjecture that our techniques are relevant or even optimal in many other scenarios as well (in particular for adaptive routing).

Our analytical worst-case results are complemented by simulations, studying fault-tolerance, failover path lengths and loads on different (but dense) topologies and in particular the Clos datacenter network.

Background & Preliminaries. Our approach is generic, and relevant for any resilient routing mechanism based on a static failover technology. In particular, it applies to Software-Defined Networks (SDNs) and their standard protocol, OpenFlow. In a nutshell, an SDN outsources and consolidates the control over a set of network switches to a logically centralized controller. As this controller is decoupled from the data plane, interactions with the controller introduce non-trivial latencies and overheads. Accordingly, OpenFlow offers a local fast failover mechanism which can provide high-throughput forwarding in the face of multiple simultaneous failures without communication with the controller: an OpenFlow switch can be pre-configured with a set of failover rules for each flow. Different flows can be defined e.g., based on layer-2, layer-3 and layer-4 header fields. The failover rules become active based on the status of the links incident to the given switch, without contacting the controller.

If a local fast failover scheme is implemented at the hardware level, it can react near-instantaneously to link failures. Our mechanism can be implemented in OpenFlow based on *failover group tables* designed specifically to detect and overcome port failures. A group has a list of action buckets and each bucket has a watch port as a special parameter. The switch monitors liveness of the indicated port. If it is down, this bucket will not be used and the group quickly selects the next bucket (i.e., the backup tunnel) in the list with a watch port that is up.

The deterministic failover mechanism presented in this paper is based on combinatorial design theory [32]. In a nutshell, combinatorial mathematics deal with the existence, construction and properties of systems of finite sets whose arrangements satisfy generalized concepts of balance and/or symmetry. Traditionally, combinatorial designs are built around Balanced Incomplete Block Designs (BIBDs), Hadamard matrices and Hadamard designs, symmetric BIBDs, Latin squares, resolvable BIBDs, difference sets, and pairwise balanced designs (PBDs). Other combinatorial designs are related to or have been developed from the study of these fundamental designs. We refer the reader to [32] for more background.

Organization. The remainder of this paper is organized as follows. Section II introduces our problem statement and formal model. We derive a load lower bound in Section III, and characterize resilient oblivious routing schemes in Section IV. In Section V, we present our approach together with a formal analysis. Section VI evaluates the performance of our failover schemes by simulation, followed by a discussion of related work in Section VII. The paper is concluded in Section VIII.

II. Problem Statement & Model

Consider an SDN-network $G = (V, E)$ with n OpenFlow switches (*nodes*) $V = \{v_1, \ldots, v_n\}$ connected by bidirectional links E. Each node v stores two kinds of flow rules:

1) The *original flow rules*, describing the "regular" forwarding behavior for arriving packets of a given flow[1].
2) The *(conditional) failover flow rules*, describing how packets of a given flow arriving at v should be forwarded in case of incident link or node failures. Both the original and the failover flow rules are pre-installed by the controller and are static.

We note that while the failover rules may be communicated by a centralized (SDN) controller ("preprocessing phase"), the actual failover will not require any communication ("runtime"). In general, we will focus on *oblivious routing schemes* in this paper: in oblivious routing, the route of a packet does not depend on other packets, and in particular, is independent of the load in the network. Moreover, if not stated otherwise, we do not consider the possibility to match the inport in our rules.

We consider an initial network where all nodes are directly connected (a clique, e.g., a backbone network). The communication pattern C is represented by a list of source and destination pairs of nodes. For simplicity we call the i^{th} item in C flow i, with source s_i and destination d_i and assume unit resource demands (e.g., bandwidth). For ease of presentation, we assume

[1]Note that multiple flows may have the same source and destination node. However they may belong to different connections, e.g., TCP connections.

that there are at most n flows in the first part of the paper and later present an extension for more flows.

Definition 1 (Load Overhead)**.** *Let $G = (V, E)$ be a graph, and $e \in E$ an edge. The load overhead $\phi(e)$ is the number of additional flows f_i crossing edge e due to rerouting. Henceforth, let $\phi = \max_{e \in E} \phi(e)$ denote the maximum overhead load (often called simply* load *in the remainder of the paper).*

We study failover schemes that pursue two goals:

1) *Correctness:* The route taken by each flow is a valid path; there are no forwarding loops. In this paper, we will ensure correct paths even under a large number of failures (a resilience property).
2) *Balanced overhead:* The resulting flow allocations are "load-balanced", i.e., minimize the overhead load of the maximally loaded link in G after the failover: $\min \max_{e \in E} \phi(e)$.

Note that flows that follow their path without rerouting do not contribute to the overhead load. To analyse the load overhead 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[2]), we need some more definitions. In general, to study the limits of the failover scheme, we focus on worst-case overhead load: we assume the link failures are determined by an adversary knowing the resilient routing protocol.

Definition 2. *Let F be a set of failed links, $F \subset E$. Given a communication pattern C, a* worst case scenario *constitutes a set of failed links F that generate the worst overhead 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 an overhead 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 overhead load $\phi(e)$ under a link failure set F is ϕ and there are no link failure sets smaller than f generating the same overhead load.*

Besides considering n arbitrary flows, we also consider two well-studied more specific communication patterns: all-to-one communication (one flow to a specified node from all other nodes) and permutation routing (one flow from each node to one other node in the network).

The following table describes the use of the main variables.

Variable	Meaning
n	number of nodes in network
F	set of failed links
f	number of failures, $f = \|F\|$
$\phi(e)$	load overhead load on link e
ϕ	maximum load overhead

III. A Load Lower Bound

Let us first investigate the limitations of local failover mechanisms from a conservative worst-case perspective. Concretely, we will show that even in a fully meshed network (i.e., a *clique*), a small number of link failures can either quickly disrupt connectivity (i.e., the forwarding path of at least one source-destination pair is incorrect), or entail a high load. This is true even though the remaining physical network is still well connected: the minimum edge cut (short: *mincut*) is high, and there still exist many disjoint paths connecting each source-destination pair.

Theorem 1. *No local failover scheme can tolerate $n-1$ or more link failures without disconnecting source-destination pairs, even though the remaining graph (i.e., after the link failures) is still $n/2$-connected.*

Proof. We consider a physical network that is fully meshed and the all-to-one traffic pattern where all nodes communicate with a single destination v_n. To prove our claim, we will construct a set of links failures that creates a loop, for any local failover scheme. Consider a flow $v_1 \to v_n$ connecting the source-destination pair (v_1, v_n). The idea is that whenever the flow from v_1 would be directly forwarded to v_n in the absence of failures, we fail the corresponding physical link: that is, if v_1 would directly forward to v_n, we fail (v_1, v_n). Similarly, if v_1 forwards to (the backup) node v_i, and if v_i would send to v_n, we fail (v_i, v_n), etc. We do so until the number of intermediate backup nodes for the flow $v_1 \to v_n$ becomes $\lfloor \frac{n}{2} \rfloor$. This will require at most $\lfloor \frac{n}{2} \rfloor$ failures (of links to v_n) since every such failure adds at least one backup node.

In the following, let us assume that the last link on the path $v_1 \to v_n$ is (v_k, v_n). We simultaneously fail all the links (v_k, v_*), where v_* are all the nodes that are not the intermediate nodes on the path $v_1 \to v_n$, and not v_1. So, there are $n - \lfloor \frac{n}{2} \rfloor - 2$ nodes v_* (the minus 2 accounts for v_1 and v_k). By failing the links to v_*, we left v_k without a valid routing choice: All the remaining links from v_k point to nodes which are already on the path $v_1 \to v_n$, and a loop is inevitable.

In total, we have at most $\lfloor \frac{n}{2} \rfloor + n - \lfloor \frac{n}{2} \rfloor - 2 = n - 2$ failures. Notice, that the two nodes with the smallest degrees in the graph are the nodes v_n (with degree of at least $\frac{n}{2} - 1$) and v_k (with degree of at least $\frac{n}{2}$). The latter is true since the first $\lfloor \frac{n}{2} \rfloor$ failures were used to disconnect links to v_n, and another $n - \lfloor \frac{n}{2} \rfloor - 2$ failures were used to disconnect links from v_k. All the other nodes have a degree of $n - 2$.

The network is still $\frac{n}{2} - 1$ connected: the mincut of the network is at least $\frac{n}{2}$. Consider some cut with k nodes on the one side of the cut, and $n - k$ nodes on the other side. Obviously, one of the sets has a size of at most $\lfloor \frac{n}{2} \rfloor$; let us denote this smaller set by S. If S includes at least one of the nodes $V \setminus \{v_k, v_n\}$, then the number of outgoing edges form the set is at least $n - 2 - (|S| - 1)$, thus the mincut is at least $\frac{n}{2} - 1$. If S includes only both v_k and v_n, the mincut is at least $n - 1$ (the link (v_k, v_n) was failed). If only one of the nodes $\{v_k, v_n\}$ is in S, then the mincut is at least $\frac{n}{2} - 1$. □

Regarding the maximal link load we have:

Theorem 2. *For any local failover scheme tolerating f link failures $(0 < f < n)$ without disconnecting any source-destination pair, there exists a failure scenario which results in a link load of at least $\widehat{\lambda} \geq \sqrt{f}$, although the minimum edge cut (mincut) of the network is still at least $n - f - 1$.*

[2]Obviously, a node which failed can no longer be reached. Our approach addresses a more general problem.

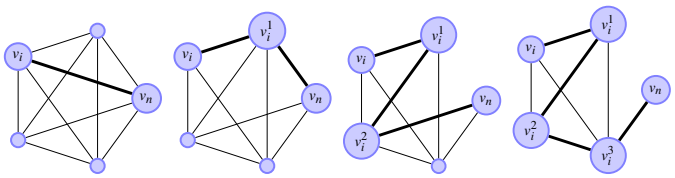


Fig. 1. *From left to right:* failover path $(v_i \to v_n)$ where each time the last hop to v_n is failed.

Proof. Let us first describe an adversarial strategy that induces a high load: Recall that in the absence of failures, each node v_i $(i \neq n)$ may use its direct link to v_n for forwarding. However, after some links failed, v_i may need to resort to the remaining longer paths from v_i to v_n. Since the failover scheme $\mathcal{S}$ tolerates f failures and v_i remains connected to v_n, $\mathcal{S}$ will fail over to one of $f+1$ possible paths. To see this, let v_i^j $(j \in [1, \ldots, f])$ be one of the f possible last hops on the path $(v_i \to \cdots \to v_i^j \to v_n)$, and let us consider the paths generated by $\mathcal{S}$:

$$
\begin{aligned}
&(v_i \to v_n),\\
&(v_i \to \cdots \to v_i^1 \to v_n),\\
&(v_i \to \cdots \to v_i^1 \to \cdots \to v_i^2 \to v_n),\\
&\cdots\\
&(v_i \to \cdots \to v_i^1 \to \cdots \to v_i^2 \to \cdots \to v_i^f \to v_n).
\end{aligned}
$$

E.g., the path $(v_i \to \cdots \to v_i^1 \to v_n)$ is generated if the first failure is link (v_i, v_n), and the path $(v_i \to \cdots \to v_i^1 \to \cdots \to v_i^2 \to v_n)$ if the second failure is link (v_i^1, v_n) (see Fig. 1 for an illustration). Notice that the last hop v_i^j is unique for every path; otherwise, loop-freeness would be violated.

For each $i \in [1, \ldots, n-1]$ (i.e., for each possible source) consider the set $A_i = \{v_i, v_i^1, \ldots, v_i^{\sqrt{f}}\}$, and accordingly, the multiset $\bigcup_i A_i$ is of size $|\bigcup_i A_i| = (n-1)(\sqrt{f}+1)$ many nodes. Since we have $n-1$ distinct nodes (we do not count v_n), by a counting argument, there exists a node $w \in \bigcup_i A_i$ which appears in at least $\sqrt{f}$ sets A_i.

If for each i such that $w \in A_i$, the adversary will cause v_i to route to v_n via w, then the load of the link (w, v_n) will be at least $\sqrt{f}$. This can be achieved by failing at most $\sqrt{f}$ links to v_n in each such set A_i. Thus, the adversary will fail $\sqrt{f} \times \sqrt{f} = f$ links incident to v_n, while the maximum loaded link (w, v_n) will have a load of at least $\sqrt{f}$.

It remains to prove that the network remains highly connected, despite these failures: The proof is simple. In a clique network without failures, the mincut is $n-1$. In the worst case, each link failure will remove one link from some cut, and hence the mincut must eventually be at least $n-f-1$. By the same argument, there are at least $n-f-1$ many disjoint paths from each node v_i to the destination: initially, without failures, there are $n-1$ disjoint paths (a direct one and $n-1$ indirect ones), and each failure affects at most one path. □

Interestingly, we can prove analogously that if failover rules only depend on destination addresses, the result is even worse.

Theorem 3. *Consider any local destination-based failover scheme in a clique graph. There exists a set of f failures $(0 < f < n)$, such that the remaining graph will have a mincut of $n-f-1$ and $\widehat{\lambda} \geq f$.*

Proof. In order to construct a bad example, we first fail the direct link (v_1, v_n), and hence v_1 will need to reroute to some path with the last node before v_n being some node v_i. When we fail the link (v_i, v_n), v_i will have to reroute and some other node v_j will become the last hop on the path to v_n. We repeat this strategy to fail the links from the newly selected last hop and the destination v_n. This results in a routing path $v_1 \to \cdots \to v_i \to \cdots \to v_j \to \cdots \to w \to v_n$ with at least f intermediate nodes. Since the algorithm is destination-based, i.e., forwarding rules depend only on the destination of a packet, the load on the link (w, v_n) is at least $f+1$: all the nodes on the path $v_1 \to v_n$ send their packets along the same route. □

IV. Characterizing Resilient Routing Schemes

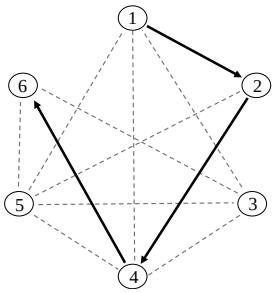
A naive solution to implement load-optimal fast failover would be to use rules defining a forwarding behavior for all possible combinations of failures. Indeed, at first sight such a combinatorial approach may seem unavoidable in order to minimize the load. However, the number of required rules would be combinatorial (exponential) in the number of ports; as we will show, this overhead is unnecessary.

Our proposed failover scheme can be best described in the form of a matrix. The matrix indicates, for each of the n flows (one per row), the backup forwarding sequence. That is, any failover scheme $\mathcal{S}$ can be represented in a generic *matrix form* $M = [m_{i,j}]$ (see also upcoming example in Figure 2):

$$
M = \begin{bmatrix}
m_{1,1} & m_{1,2} & \ldots & m_{1,n} \\
\vdots & \vdots & \ddots & \vdots \\
m_{i,1} & m_{i,2} & \ldots & m_{i,n} \\
\vdots & \vdots & \ddots & \vdots \\
m_{n,1} & m_{n,2} & \ldots & m_{n,n}
\end{bmatrix}.
$$

Any failover scheme instance $\mathcal{S}$ will always forward a message directly to the destination, if the corresponding link is available. Otherwise, if a message of the i^{th} flow from source s_i cannot reach its destination d_i directly via (s_i, d_i), it will resort to the sequence of alternatives represented as the row i in the matrix M (the "backup nodes" for the i^{th} flow), as described in Algorithm 1. Node s_i will first try to forward to node $m_{i,1}$, if this link is not available to node $m_{i,2}$, and so on. More generally, if a message with source s_i is currently at node $m_{i,j}$ it will be forwarded directly to the destination d_i, if the link $(m_{i,j}, d_i)$ is available. Otherwise, the failover scheme will try to send it to $m_{i,j+1}$, $m_{i,j+2}$, etc. In other words, if the link $(m_{i,j}, m_{i,j+1})$ is not available, then the link $(m_{i,j}, m_{i,j+2})$ is tried next, and so on. If $(m_{i,j}, m_{i,j+2})$ is available, the message will be forwarded to node $m_{i,j+2}$. If this node cannot reach d_i, that is $(m_{i,j+2}, d_i)$ failed, the link $(m_{i,j+2}, m_{i,j+3})$ will be tried, etc.

In general, we can observe that in order to avoid loops (and provide maximal resilience), each row should contain each non-source/non-destination nodes exactly once. To make the analysis and description simpler, we also allow the source and destination nodes to appear in each row: the failover scheme



$$M_1 = \begin{bmatrix} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & 5 & 6 \\ \mathbf{2} & \mathbf{3} & \mathbf{4} & 5 & 1 & 6 \\ \mathbf{3} & \mathbf{4} & 5 & 1 & 2 & 6 \\ 4 & 5 & 1 & 2 & 3 & 6 \\ 5 & 1 & 2 & 3 & 4 & 6 \\ 5 & 1 & 2 & 3 & 4 & 6 \end{bmatrix}$$

Fig. 2. Example: Rerouting of flow i from 1 to 6, according to M, where the i^{th} row without source and destination nodes is $[2, 3, 4, 5]$. If the links $\{(1, 6), (2, 6), (2, 3)\}$ failed, packets of this flow are first forwarded to node 2 from node 1. Since there is no direct link from 2 to 6, the next entry in the row, 3, is considered. As the link between 2 and 3 is missing and the next entry is 4, packets are then forwarded to node 4, from where they can reach their destination.

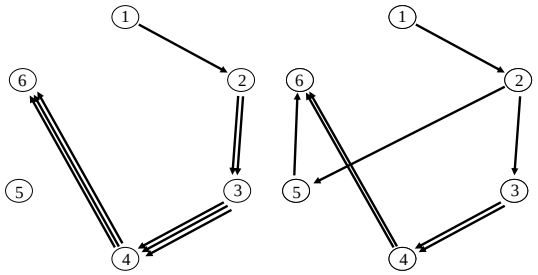


Fig. 3. Example: Rerouting of flows from nodes 1,2, and 3 due to link failures $\{(1, 6), (2, 6), (3, 6)\}$ according to M_1 (*left*) and M_2 (*right*) respectively. A failover matrix rerouting flows to similar paths can lead to a high overhead load (*left*). Accordingly, failover matrixes should be designed where node repetitions in row prefixes are minimized (*right*).

Algorithm 1 Rerouting given a Failover Matrix M

Upon receiving a packet of flow i at node v:

if destination not reached yet, $d_i \neq v$ **then**
  if (v, d_i) available **then**
    forward packet to d_i
  else
    let j be v's index in row i /* $m_{i,j} = v$ */
    while $m_{i,j+1} = s_i$ or $(v, m_{i,j+1})$ unavailable **do**
      $j = j + 1$
    forward packet to $m_{i,j+1}$

simply ignores them when they occur. Thus, each row is a permutation of all nodes in our schemes.

Figure 2 illustrates the use of a failover matrix for a flow from node 1 to node 6, when the link $\{(1, 6), (2, 6), (2, 3)\}$ fail. Observe that while the specific permutation does not matter for correctness, it matters in terms of performance. Figure 3 shows an example for $n = 6$ in a scenario with flows from each node to node 6 (no flow from node 6 to another node). We assume that the links $\{(1, 6), (2, 6), (3, 6)\}$ fail. On the left, the resulting failover routes for matrix M_1 are shown, where the i^{th} flow originates from node i. The elements in bold indicate the prefixes of the rows that are used for rerouting. The resulting maximum overhead load is 3 on $(4, 6)$: the load of 3 flows aggregates along the failover path. On the right, a failover scheme resulting in load 2 only is shown. For example, this can be achieved with the following failover matrix:

$$M_2 = \begin{bmatrix} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & 5 & 6 \\ \mathbf{2} & \mathbf{5} & 1 & 3 & 4 & 6 \\ \mathbf{3} & \mathbf{4} & 5 & 1 & 2 & 6 \\ 4 & 1 & 2 & 5 & 3 & 6 \\ 5 & 3 & 4 & 2 & 1 & 6 \\ 5 & 1 & 2 & 3 & 4 & 6 \end{bmatrix}.$$

Intuitively, the bad performance of M_1 comes from the similarity of each node's scheme: as nodes all rely on similar failover schemes, the failover flows will all end up on the same route, leading to a high link congestion.

V. Resilient Oblivious Routing

A. How to Create Good Failover Matrices?

Before we present the proposed schemes to compute resilient oblivious routing matrices, let us first make some observations. We will first focus on the fundamental *All-to-One Routing* scenario which is often considered in related works [18]: all nodes communicate to a single destination d, let us say v_n (we assume v_n does not communicate to anyone else, so we consider $n - 1$ flows only). The following auxiliary claim shows that for all-to-one routing the highest overhead load is induced if links towards the destination node v_n are failed. In this case, the adversary can "reuse" failures: if the adversary removes the links between v_i and v_n, then the occurrence of v_i in any failover sequence implies a higher number of flows on the subsequent node in the failover sequence.

Claim 4. *For the all-to-one routing, the highest load is induced if links towards the destination node v_n are failed.*

Proof. To achieve a load of f on some link, the adversary first needs to bring at least f flows to some node w. Consider a failover sequence $m_{i,\cdot}$ in which w is located at j's position, i.e., $m_{i,j} = w$. In order to route the flow $v_i \to v_n$ to node w, the adversary needs to fail at least j links (every failure requires at most a single additional backup node). Thus, the adversary can remove the links to the destination from every node $m_{i,k}, k < j$ and from the source v_i. The optimality is due to the fact that once one of the *nodes* $m_{i,k}, k < j$ appears in other sequences, these failures are automatically reused: the links $(m_{i,k}, v_n)$ already failed. If the adversary would instead choose to fail other *links* (not towards the destination), e.g., $(m_{i,j}, m_{i,j+1})$, the failures can only be reused if the same *link* (and not only an endpoint) appears in other sequences before w. Therefore, we conclude that the strategy of failing the links to the destination is optimal: (1) it requires no more failures to bring a specific flow to w than any other strategy, and (2) link failures to the destination can strictly be reused more often than the failures of links to any other nodes. □

Thanks to this claim, we can assume that F consists of links $(v_{i_1}, v_n), (v_{i_2}, v_n), \ldots$ for some $i_1, i_2, \ldots$ only. Accordingly, we refer to them by $i_1, i_2, \ldots$ for all-to-one routing.

Consider two flows originating from u and v in a system relying on a failover scheme represented by M. Both flows

cannot reach the destination, so they are rerouted to their fail-safe paths, trying the failover paths as described earlier. If they both use the same node t in their failover paths, the links from nodes earlier in the corresponding rows of the failover matrix to the destination must have failed. That is, $m_{u,a} = m_{v,b} = t$ for some indices a, b. Thus the flow from u will transit through t if all the previous failover routes have failed: $\{(m_{u,i}, v_n), 1 \leq i \leq a\} \subseteq F$. Similarly, the flow from v will transit through t if $\{(m_{v,i}, v_n), 1 \leq i \leq b\} \subseteq F$. As a shorthand notation, we refer to the set of elements of row i before t as $P_t(i) = \{m_{i,j} | m_{i,t_i} = t, 1 \leq j \leq t_i\}$, the prefix of t in row i (this can include the source and destination of the flow, although they are ignored in the failover scheme). The number of failed links is hence at least the number of elements in the union of these prefixes minus the occurrence of the destination node of the flows: $f \geq a + b - |P_t(u) \cap P_t(v)| - 2$. This relation provides two techniques to ensure that the link (t, v_n) has a low load: i) makes sure a and b are maximized (that is, t is used as a last resort), and ii) ensure that failover routes used by u and v are as different as possible, ideally $P_t(u) \cap P_t(v) = \emptyset$: thus the adversary cannot "reuse" failed links on the failover path of packets from u when targeting packets emitted by v.

When generalizing this analysis to all nodes of the system, it is interesting to observe that i) and ii) conflict: on the one hand, several different nodes must be used early in failover schemes (to prevent a large intersection size), on the other, nodes should be used as late as possible on failover sequences so that no congestion can easily happen on their link to v_n.

B. Randomized Failover Schemes

Let us now use our observations above to design good failover matrices. A first naive approach may be to choose the matrix entries $m_{i,j}$ (i.e., the "failover ports") uniformly at random from the set of next hops which are still available, and depending on the source and destination address, in order to balance the load. However, note that a random and *independent* choice will quickly introduce loops in the forwarding sequences: it is likely that a switch will forward traffic to a switch which was already visited before on the failover path.

Thus, our randomized failover scheme **RFS** will choose random *permutations*, i.e., for a source-destination pair (v_i, v_n), the sequence $m_{i,1}, m_{i,2}, \ldots, m_{i,n-2}$ (with $m_{i,j} \in V \setminus \{v_i, v_n\}$) is *always loop-free* (deterministically). Technically, **RFS** draws all $m_{i,j}$ uniformly at randomly from $V \setminus \{v_i, v_n\}$ but eliminates repetitions (e.g., by redrawing a repeated node). We can show that **RFS** is almost optimal, in the following sense.

Theorem 5. *Using the* **RFS** *scheme, in order to create a maximum load of $\lambda = \sqrt{f}$, the adversary will have to fail at least $\Omega\left(\frac{f}{\log n}\right)$ links* w.h.p.*, where $0 < f < n$.*

Proof. To create a link load of $\sqrt{f}$ with the minimal number of link failures, the adversary must route at least $\sqrt{f}$ flows to some node w. Given the $\sqrt{f}$ load on the node, in the best case (for the adversary), the entire flow will be forwarded by w on a single outgoing link (e.g., the link to the destination v_n). We will show that w.h.p., it is impossible for the adversary to route more than $\sqrt{f}$ flows to a single node.

The adversary can create a high load for some node w only if: 1) Node w is located close to the beginning of many sequences (i.e., is in a small "prefix" of the sequences); thus, a small number of failures is sufficient to redirect the flow to w. 2) Many nodes appearing before w in the sequence prefixes occur early in many other prefixes as well; thus, the adversary can "reuse" failed links for other source-destination pairs. Note that while these two requirements conflict, we use them to prove the lower bound on the number of required failures: the set of $\sqrt{f}$ sequences with the largest number of node repetitions in the w-prefixes also have the shortest w-prefixes.

With this intuition in mind, let us compute the probability that a node w appears more than approximately $\log n$ times at position j. Let Y_i^j be an indicator random variable that indicates whether w is located at position $j \in [1, \ldots, n-2]$ in sequence $i \in [1, \ldots, n-1]$. Let $Y^j = \sum_{i=1}^{n} Y_i^j$ be a random variable representing the number of times that w appears at position j. Since the failover sequences are random, $\Pr(Y_i^j = 1) = \frac{1}{n-2}$ (w is neither the source nor the destination) and thus, $\forall j, \mathbb{E}\left[Y^j\right] = \frac{n-1}{n-2}$. Applying the Chernoff bound on the sum of n *i.i.d.* Poisson trials, we obtain (for any $\delta > 0$):

$$\Pr\left(Y^j > (1+\delta)\mathbb{E}\left[Y^j\right]\right) \leq 2^{-\delta\mathbb{E}\left[Y^j\right]}$$

$$\Pr\left(Y^j > \frac{(1+3\log n)(n-1)}{n-2}\right) \leq 2^{-(3\log n)\times\frac{n-1}{n-2}} \leq 2^{-3\log n} = 1/n^3.$$

Let $z = \frac{(1+3\log n)(n-1)}{n-2}$ and thus $\Pr\left(Y^j > z\right) \leq 1/n^3$.

We can now apply a union bound argument[3] over all possible nodes w and positions j, which yields that with probability at least $1 - \frac{1}{n}$, any node appears at most z times at each position.

The adversary needs to select the $\sqrt{f}$ sequences with the shortest w-prefixes. For a chosen sequence i, let us denote by k_i the prefix length for node w (the prefix length includes w itself). Since each node will appear no more than z times at each position (with probability of at least $1 - \frac{1}{n}$) the minimum length of a *total prefix* for any node w can be derived. Let us denote the minimum *total prefix* by k. Clearly, k is minimized for the shortest possible prefixes k_i. According to the analysis above, with high probability, there are no more than z prefixes of length 1, no more than z prefixes of length 2, and so on. Therefore:

$$k = \sum_{i=1}^{\sqrt{f}} k_i \geq \sum_{i=1}^{z} 1 + \sum_{i=1}^{z} 2 + \cdots + \sum_{i=1}^{z} \frac{\sqrt{f}}{z}$$

$$= z\left(1 + 2 + \cdots + \frac{\sqrt{f}}{z}\right) \geq \frac{f}{2z} \geq \frac{f}{8\log n}. \quad (1)$$

Eq. 1 is true since for $n \geq 6$, $\frac{(1+3\log n)(n-1)}{n-2} \leq 8\log n$.

In conclusion, we know that in order to achieve a load of $\sqrt{f}$, the adversary has to fail the entire total prefix of w that consists of at least $\frac{f}{8\log n}$ nodes. However, the nodes in the prefixes are not necessarily all distinct, and the number of links the adversary needs to fail only depends on the *distinct* nodes in the *total prefix* of the node w. The latter is true due to the

[3]The union bound argument says that the probability of the union of the events is no greater than the sum of the probabilities of the individual events.

fact that the best adversarial strategy is to fail only the links to the destination since in this case every such failure is reused once the same node appears again in the *total prefix* of w (see Claim 4). Hence, we next compute the minimum number of distinct nodes D in any set of k random nodes. As we are interested in lower bounding D, we can choose k minimal, i.e., $k = \frac{f}{8 \log n}$. The analysis follows from a *balls-and-bins* argument where bins represent node IDs and balls are the k positions that should be failed by the adversary. Thus, D is a number of occupied bins (i.e., bins that contain at least one ball). Let D_i be a binary random variable indicating that the i-th ball falls into an empty bin (i.e., $D = \sum_{i=1}^{k} D_i$). So, $\Pr(D_i = 1) \geq \frac{n-1-k}{n-1}$. Since $k = \frac{f}{8 \log n}$ and $f < n$, we obtain $\Pr(D_i = 1) \geq \frac{n-1-k}{n-1} \geq \frac{8 \log n - 1}{8 \log n} \geq 0.8$. Thus, $\mathbb{E}[D] = k\mathbb{E}[D_i] \geq 0.8k$. Due to Chernoff (for any $\delta \in (0, 1]$):

$$\begin{aligned}\Pr(D \leq (1-\delta)0.8k) &\leq \Pr(D \leq (1-\delta)\mathbb{E}[D]) \\ &\leq e^{-\mathbb{E}[D]\delta^2/2} \leq e^{-0.8k\delta^2/2}.\end{aligned}$$

By taking $\delta = 0.5$ we obtain $\Pr(D \leq 0.4k) \leq e^{-0.1k}$.

It remains to prove that this bound still holds under the union bound for all $\binom{n-1}{\sqrt{f}}$ possible sets of sequences that the adversary can choose. In other words, we have to ensure that $\binom{n}{\sqrt{f}} e^{-0.1k} \leq \frac{1}{n}$ (we took a larger number as $\binom{n}{\sqrt{f}} \geq \binom{n-1}{\sqrt{f}}$).

$$\begin{aligned}\binom{n}{\sqrt{f}} e^{-0.1k} &\leq n^{\sqrt{f}} e^{-0.1k} = n^{\sqrt{f}} e^{-\frac{f}{80 \log n}} \qquad (2) \\ &= e^{\sqrt{f} \ln n - \frac{f}{80 \log n}} = e^{f\left(\frac{\ln n}{\sqrt{f}} - \frac{1}{80 \log n}\right)} \\ &\leq e^{f\left(\frac{\log n}{\sqrt{f}} - \frac{1}{80 \log n}\right)}.\end{aligned}$$

For $f \geq 82^2 \log^4 n$, we have $(\frac{\log n}{\sqrt{f}} - \frac{1}{80 \log n}) \leq \frac{-2}{82^2 \log n}$, and hence $\binom{n}{\sqrt{f}} e^{-0.1k} \leq e^{\frac{-2f}{82^2 \log n}} \leq e^{-2 \log^3 n} \leq \frac{1}{n^2}$. Since $\binom{n}{\sqrt{f}} \Pr(D \leq 0.4k) = \binom{n}{\sqrt{f}} \Pr(D \leq \frac{0.4f}{8 \log n}) \leq \frac{1}{n^2}$, w.h.p., any set of $\sqrt{f}$ sequences needs $\Omega(\frac{f}{\log n})$ failures. □

C. Deterministic Failover for Few Failures

Theoretically, the result of Theorem 5 can be derandomized, i.e., the **RFS** scheme can *deterministically* ensure low loads. The idea is that we could *verify* whether an (improbable) situation occurred and the random sequences generated by **RFS** yield a *high* load (we just need to check all possible loads at any w); if so, another set of random permutations is generated. However, this verification is computationally expensive.

We hence next want to find deterministic schemes. In particular, we propose an optimal failover scheme (which matches our lower bound in Section IV), for small f. Similar to **RFS**, the deterministic failover scheme **DFS** is again defined by *failover matrix* $m_{i,j}$; however, here $m_{i,j}$ will simply refer to a node's *index* (and not the node itself): We define the index of any node v_ℓ to be $\ell - 1$, i.e., the nodes $\{v_1, v_2, \ldots, v_n\}$ are mapped to the indices $\{0, 1, \ldots, n-1\}$. Given a destination node v_n, **DFS** is defined by the following index matrix:

$$\begin{aligned}&1, 2, 4, 8, \ldots, \left(0 + 2^{\lfloor \log n \rfloor}\right) \mod n \\ &2, 3, 5, 9, \ldots, \left(1 + 2^{\lfloor \log n \rfloor}\right) \mod n \\ &3, 4, 6, 10 \ldots, \left(2 + 2^{\lfloor \log n \rfloor}\right) \mod n \\ &\ldots\end{aligned}$$

In general, the index in sequence $i \in [1, \ldots, n-1]$ at position $j \in [1, \ldots, \lfloor \log n \rfloor]$ is $m_{i,j} = (i-1) + 2^{j-1} \mod n$. E.g., if the link (v_1, v_n) fails, v_1 reroutes via the node with index 1, i.e., via v_2; and so on. We can show the following result.

Theorem 6. *The* **DFS** *scheme achieves a maximum load of* $\widehat{\lambda} = O(\sqrt{f})$ *in any scenario with* $f < \lfloor \log n \rfloor$ *failures.*

Proof. We will prove something even stronger: the adversary cannot choose link failures such that any *node* w forwards more than $\sqrt{f}$ flows. Clearly, an upper bound on the node load is an upper bound on the (incident) links: in the worst case, w will forward all traffic to the same link. To create a high load at some node w, the adversary needs to find failover sequences in the matrix $m_{i,j}$ where the node w appears close to the *beginning* of the sequence, and fail *all* the links (v_i, v_n), where v_i is a node preceding w in a sequence: i.e., the adversary fails the *total prefix* of w. Note that failing the links to the destination is the best strategy for the adversary as failures are automatically reusable in other sequences (see Claim 4).

The following two claims will help us to show that the adversary wastes its entire failure budget in order to achieve a maximum load of $\sqrt{f}$.

Claim 7. *Every node index participates in* $\lfloor \log n \rfloor$ *sequences.*

Proof. The **DFS** failover matrix is defined as $m_{i,j} = (i-1) + 2^{j-1} \mod n$, where $i \in [1, \ldots, n-1]$ and $j \in [1, \ldots, \lfloor \log n \rfloor]$. From this construction, it follows that there are no index repetitions in the matrix columns. Since there are $\lfloor \log n \rfloor$ columns, the claim follows. □

Claim 8. *For any node index* ℓ*, all* ℓ*-prefixes (sets of indices preceding* ℓ *in the sequences) are disjoint.*

Proof. Let us define $m = i - 1$ and $k = \ell - 1$. The index in sequence $m \in [0, \ldots, n-2]$ at position $k \in [0, \ldots, \lfloor \log n \rfloor - 1]$ is $m + 2^k \mod n$. Consider a sequence m' where the index w appears at position k' and a sequence m'' where the index ℓ appears at position k''. Without loss of generality, assume that $k'' > k'$. Let $m' + 2^{k^*} \mod n$ and $m'' + 2^{k^{**}} \mod n$ represent the indices in the prefixes of ℓ in sequences m' and m'' accordingly. Assume by contradiction that these indices are the same. We have

$$\begin{aligned}m' + 2^{k'} &= m'' + 2^{k''} \mod n \\ m' + 2^{k^*} &= m'' + 2^{k^{**}} \mod n \text{ (assumption)}\end{aligned}$$

and hence $m' - m'' = 2^{k''} - 2^{k'} + n \cdot C_1$ and $m' - m'' = 2^{k^{**}} - 2^{k^*} + n \cdot C_2$. Therefore $2^{k^{**}} - 2^{k''} + 2^{k'} - 2^{k^*} = n \cdot C_3$ where C_1, C_2 and C_3 are some integer constants.

Notice that $\max(2^{k^{**}}, 2^{k''}, 2^{k'}, 2^{k^*}) < n$, so the only possible values for C_3 are: $\{-1, 0, 1\}$. Moreover, $(2^{k^{**}} - 2^{k''}) < 0$,

while $(2^{k'} - 2^{k^*}) > 0$, and since the absolute value of these differences is bounded by $2^{\lfloor \log n \rfloor - 1} \leq 0.5n$, we can write: $-0.5n < 2^{k^{**}} - 2^{k''} + 2^{k'} - 2^{k^*} < 0.5n$. Thus, 0 remains the only possible value for C_3. The values $\{2^{k^{**}}, 2^{k''}, 2^{k'}, 2^{k^*}\}$ are distinct since there are no repetitions in the columns of the sequence matrix. Since $2^{k''} > 2^{k^{**}} + 2^{k'} + 2^{k^*}$, due to a geometric series argument (the largest element is greater than the sum of all previous elements), we can state that $2^{k^{**}} - 2^{k''} + 2^{k'} - 2^{k^*} < 0$. We conclude that there is no integer constant C_3 satisfying our assumption $m' + 2^{k^*} = m'' + 2^{k^{**}} \mod n$ (i.e., there are two identical indices in the ℓ-prefixes). □

Armed with these claims, we are ready to continue with the proof. Since all prefixes are disjoint, the adversary cannot reuse failures of one flow for another. Thus, the adversary will be able to route *one* flow to w using a single failure (by finding a sequence in which w appears at the first position); to add another flow, the adversary takes a sequence m_i in which w is located at position 2 and will fail the links (v_i, v_n), and $(v_{(m_{i,1})+1}, v_n)$. And so on. Thus, the number of used failures can be represented as $1 + 2 + 3 + 4 + \cdots + L \leq f$ where L is the number of flows passing through w on the way to the destination v_n. So:

$$1 + 2 + 3 + 4 + \cdots + L \leq f$$
$$\frac{L(L+1)}{2} \leq f \Leftrightarrow L < \sqrt{2f}$$

Note that the index of the destination node ($n-1$ in our case) can appear inside the failover sequences. In this case, the index will be skipped since the link to it from the source already failed. By skipping one index, we shorten the failover sequence by 1, and since every sequence has length $\lfloor \log n \rfloor$, our failover scheme holds for any $f < \lfloor \log n \rfloor$. □

D. Deterministic Failover for Arbitrary Failures

We now have a randomized failover scheme as well as a failover scheme for a small number of failures. Can we find deterministic failover schemes which apply to arbitrary failures?

The answer is yes, and in the following, we establish an interesting connection to block designs and distributed algorithms without communication. We first consider failover matrices which are **latin squares**: that is, failover matrices where each node appears exactly once on each row and each column. This is useful since there are no repetitions on rows, and hence forwarding loops in failover paths are avoided.

However, while latin squares provide a high resilience, not all latin squares are good failover matrices in terms of load. As an example, let us analyze the following latin square $M = [m_{ij}]_{1 \leq i,j \leq n-1}$ where relay nodes are tried in a round-robin fashion: $m_{ij} = (i + j - 1) \mod (n-1)$. This scheme cannot lead to forwarding loops because $\forall 1 \leq j, j' \leq (n-1), j \neq j' \Rightarrow m_{ij} \neq m_{ij'}$. However, this scheme results in a high load: if the adversary fails the l first links to destination d (that is, $F = \{(v_i, d), i = 1, \ldots, l\}$), the l first nodes will all route through (v_{l+1}, d): we have $\phi = \theta(l)$.

In the next section we investigate which additional properties latin squares must have to constitute good failover matrices. As we will see, the *intersection* of prefixes of rows is crucial.

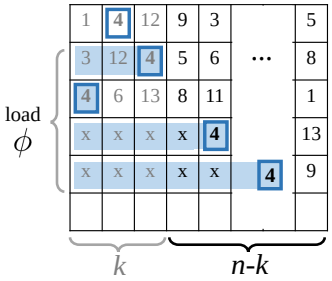


Fig. 4. Load overhead ϕ and worst-case failure sets: Let the flows $2, 3, 4, 5$ create the worst-case load $\phi = 4$ on the link $(4, v_n)$. In the failover matrix M, we highlight the occurrence of 4 with a square in each row, and color the background of the prefixes of the rows creating the load in blue. The number of failures leading to the use of the blue prefixes depends on the number of unique elements in their union. Two prefixes of length at most k share at most one element, hence we can bound the number of distinct elements.

E. *BIBDs* or: *How to create submatrices of low intersection?*

1) Performance of Latin Square Schemes: Let us now take a closer look at how a high load can arise at a node. A link $e = (w, x)$ carrying load ℓ, by definition, serves on the failover route of ℓ different flows. In particular, there are ℓ rows in the failover matrix which include w, the head node of the link, "early on", in a short prefix of the row: the current set of failures leads to a failover routing to w.

Accordingly, if the maximum load is ϕ then there is a node w where this maximum load is manifested and ϕ rows of M are responsible for generating this load. In other words, these ϕ rows form a set T, where the links from the predecessors of w to v_n in these rows (ignoring the destination node) are all in the failure set, i.e., $\exists w \in V$ s.t. $\bigcup_{i \in T} P_w(i) \setminus \{v_n\} \subseteq F$.

Let M be a latin square failover matrix and $F \in F_o(\phi)$ an optimal attack set of worst-case failures causing maximum load. We now aim to lower bound the minimal size of F. Let (w, v_n) be the link on which the load is ϕ. We have $f \geq |\bigcup_{i \in T} P_w(i)| - 1$ (we deduct one to account for the destination node). In the best case (from a load perspective), for instance when $\phi << n$, two rows do not intersect: $f = \sum_{i \in T} |P_w(i)|$. Since M is a latin square, it holds for all $i, j \in T$ that the position of w in the rows differs. If F is of minimal cardinality, F must contain the shortest prefixes: $f \geq \sum_{i \in T}(|P_w(i)| - 1) = (t-1)(t-2)/2$, for $t = |T|$, because an occurrence of the t in the prefix of w in row i is ignored.

This optimistic estimation technique captures the core of our performance analysis scheme. The only technical problem is now to limit the intersection size between the rows affected by the failures. Of course, since any row ultimately contains the n nodes, we must work on the first columns of failover matrix M. Let $M^k = [m_{ij}]_{1 \leq i < n, 1 \leq j \leq k}$ denote the k-block of M, the submatrix of the failover matrix consisting of the k first columns of M and let $M^k(i)$ the set of the first k elements of the i^{th} row of M. We say that a matrix is a latin matrix if it can be the k-block of a latin square, that is, no element occurs more than once in each row and in each column.

We now formalize the statement for the minimal number of link failures necessary to generate a load ϕ, depending on the intersection size of short prefixes. Figure 4 depicts an example of how load accumulates on a link.

Theorem 9. *Let $k \leq \sqrt{n}$ and M^k a latin submatrix such that the size of the intersection of any two rows is at most 1. That is, for $\forall i, j \leq n, i \neq j$ it holds that $|M^k(i) \cap M^k(j)| \leq 1$. Let $F \in F_o(\phi)$. If $\phi \leq k$ then $f = \Omega(\phi^2)$.*

Proof. Let w be a node that carries a load of ϕ on its link to v_n due to F. Consider the set of failover sequences that contribute to this load (the rows with the blue background in Figure 4, i.e., the set of flows T for which all nodes in the prefix of their rows are in the failure set, $\bigcup_{i \in T} P_w(i) \cup F = F$. Observe that $|T| = \phi$. Partition T in two subsets: T_1 for flows whose prefix for w is shorter than k, $|P_w(i)| \leq k$, T_2 for all other flows and let $t_1 = |T_1|$ and $t_2 = |T_2|$, $t_1 + t_2 = \phi$. All the links to v_n from the predecessors of w on the rows of T in M must be in the set of failed links, unless the destination v_n is in the prefix. Hence, it holds that $f \geq |\bigcup_{i \in T} P_w(i)| - \phi$. Using the partition into T_1 and T_2 we have $|\bigcup_{i \in T} P_w(i)| \geq |\bigcup_{i \in T_1} P_w(i) \cup \bigcup_{i \in T_2} P_w(i)| \geq |\bigcup_{i \in T_1} P_w(i) \cup \bigcup_{i \in T_2} M^k(i)|$, where the last inequality is due to the fact that $M^k(i) \subseteq P_w(i)$ for all $i \in T_2$. Leveraging the inclusion-exclusion principle $|\bigcup_{i \in T_1} P_w(i) \cup \bigcup_{i \in T_2} M^k(i)| \geq |\bigcup_{i \in T_1} P_w(i)| + |\bigcup_{i \in T_2} M^k(i)| - |\bigcup_{i \in T_1} P_w(i) \cap \bigcup_{i \in T_2} M^k(i)|$. We now analyse each of the three cardinalities. Using the inclusion exclusion principle again, we have $|\bigcup_{i \in T_1} P_w(i)| \geq \sum_{i \in T_1} |P_w(i)| - \sum_{i,j \in T_1, i \neq j} |P_w(i) \cap P_j(w)|$. $\sum_{i<j \in T_1} |P_w(i) \cap P_j(w)| = 0$ as the intersection of the first k elements of the matrix contains w only. Due to the latin property, it hence holds that $|\bigcup_{i \in T_1} P_w(i)| \geq \sum_{i \in T_1} |P_w(i)| = \sum_{i \in T} |P_w(i)| \geq t_1(t_1 - 1)/2$. Analogously, we can write $|\bigcup_{i \in T_2} M^k(i)| \geq \sum_{i \in T_2} |M^k(i)| - \sum_{i<j \in T_2} |M^k(i) \cap M^k(j)| = k \cdot t_2 - t_2(t_2 - 1)/2$. The last term, $|\bigcup_{i \in T_1} P_w(i) \cap \bigcup_{j \in T_2} M^k(j)|$ is at most $|\bigcup_{i \in T_1} M^k(i) \cap \bigcup_{j \in T_2} M^k(j)|$, which in turn is equal to $|\bigcup_{i \in T_1, j \in T_2} (P_w(i) \cap M^k(j))|$ due to the distribution law. $|P_w(i) \cap M^k(j)| \leq 1$ for all $i \neq j$, thus the whole term can be upper bounded by $t_1 \cdot t_2$. Therefore, $f + \phi \geq t_1(t_1-1)/2 + k \cdot t_2 - t_2(t_2-1)/2 - t_1 \cdot t_2 = t_1(t_1-1)/2 + t_2(k-t_1) - t_2(t_2-1)/2$. $k - t_1 \geq t_2$, as $k \geq \phi = t_1 + t_2$. Consequently, $f + \phi \geq t_1(t_1 - 1)/2 + t_2^2 - t_2(t_2 - 1)/2 = \Omega(t_1^2 + t_2^2) = \Omega(\phi^2)$, concluding the proof. □

This theorem is an important tool in the analysis of latin square failover schemes, as it directly describes the relation between the intersection size of k-length row prefixes and the optimal attack cost f. More precisely, if we manage to create matrices which have a large k-block with such intersection properties, then we can guarantee a constant approximation of the optimal resilience for up to $O(k^2)$ failures.

2) Using BIBDs to Minimize Intersection: Given n nodes, the problem is now to generate n different failover sequences of length k with guarantees on the *size of the intersection*. Of course, this generation is trivial for $k << n$. For the performance of the resulting scheme however, the objective is to find constructions for larger k: for instance if $k = \sqrt{n}$, we have an optimal solution as the attacker would need to fail $\theta(k^2) = \theta(n)$ links to reach the limits of Theorem 9. Constructing such sets is however challenging.

Fortunately, two closely related problems are well-studied: the problem of generating *block designs* (that is, families of subsets of elements), and its geometric counterpart, generating projective planes of high order. We here choose the first approach, and next quickly introduce the relevant definitions. The interested reader can refer to [32] for an overview of the rich field of block designs. For our construction we will use *symmetric balanced incomplete block designs (BIBDs)*.

Definition 3 (BIBD, Def 1.2 and 2.1 in [32])**.** *Let $v, k,$ and λ be positive integers such that $v > k \geq 2$. A (v, k, λ)-*balanced incomplete block design *is a* design *(X, A) such that the following properties are satisfied:*

1) *X is a set of v elements called points, $|X| = v$.*
2) *A is a multiset of b non-empty subsets of X called* blocks*, where each block contains exactly k points.*
3) *Every pair of distinct elements is contained in λ blocks.*

A BIBD where $b = v$ is called symmetric*.*

Symmetric BIBDs feature a useful intersection property.

Fact 1 (Thm 2.2 in [32])**.** *Given a symmetric (v, k, λ)-BIBD, it holds for all $1 \leq i, j \leq v$, where $i \neq j$, that $|A_i \cap A_j| = \lambda$.*

The only remaining problem is that blocks are not rows: even once we have generated our n blocks of size k, we need to order the failover routes within each block such that the resulting matrix M^k is a k-block of a latin square. The following procedure will be used to construct the first k elements of row i using the elements of A_i. It leverages k sequential perfect matchings in the bipartite graph, associating to each row its set of backup nodes from the block A_i.

Algorithm 2 Transforming Blocks into Latin Rows

1: input : a (n, k, λ)-BIBD (X, A)
2: output: M^k: n rows of size k
3: Let $G = (U, V, E)$ a bipartite graph s.t., $|U| = |V| = n$ and $(i, j) \in E$ iff $x_j \in A_i$
4: **for** $j \in \{1, \ldots, k\}$ **do**
5: Let $P : U \to V$ a perfect matching of G
6: **for** $i \in \{1, \ldots, n\}$ **do**
7: $m_{ij} \leftarrow P(i)$
8: $G = (U, V, E \setminus \{(i, m_{i,j}) | 1 \leq i \leq n\})$
9: **return** $M^k = [m_{ij}]_{1 \leq i \leq n, 1 \leq j \leq k}$

Theorem 10. *Algorithm 2 returns a latin block with $|M^k(i) \cap M^k(j)| = \lambda$ for all $1 \leq i < j \leq n$.*

Proof. Let us first show that Algorithm 2 always terminates. This will happen iff a perfect matching P is always found. Observe that at Line 3, by definition of a BIBD, G is a k-regular bipartite graph (with $|U| = |V|$). It therefore contains a perfect matching (due to Hall's Theorem). Observe that after the first execution of Line 8, G is now a $k - 1$ regular graph (since a perfect matching was removed). This will be repeated until $j = k$, where G is merely a matching.

Regarding correctness, observe that no node is ever repeated in a row as the blocks are sets. Since P is a perfect matching, no node is repeated in columns. Hence, M is a latin submatrix. □

The construction of Algorithm 2 will be very useful to transform blocks into failover matrixes that provide the guarantees of Theorem 9.

3) Failover Matrix Creation: With the above we now construct a failover matrix M (summarized in Algorithm 3) given a symmetric BIBD. As a first step, Algorithm 3 exploits a symmetric $(n, k, 1)$-BIBD (X, A) to create the first k-submatrix of M. The remaining submatrix is constructed such that each row and column of the complete matrix is a permutation, and thus we have a latin square. Together with the theorems from previous sections, this suffices for a constant approximation.

Algorithm 3 Construction of Failover Matrix

1: input: $(n, k, 1)$-BIBD (X, A)
2: output: latin square failover matrix M
3: Let $M^k = [m_{ij}]_{1 \le i \le n, 1 \le j \le k} = Alg2(X, A)$
4: Let $M^C = Alg2(X, \{B_i, B_i = X \setminus A_i, 1 \le i \le n\})$
5: **return** $M = M^k \oplus M^C$, where $\oplus$ concatenates columns

Theorem 11. *Algorithm 3 returns a latin failover matrix M with intersection properties representing a failover scheme that is optimal up to a constant factor.*

Proof. We prove first termination and then correctness.

Termination: Since M^k is a latin submatrix, all the n values appear exactly once on the first column, and once on the last column. Observe that in Line 4, $(X, \{B_i, 1 \le i \le n\})$ is a BIBD (regardless of its intersection size), as the complement of a BIBD is also a BIBD ([32] Thm 1.32).

Correctness: Observe that M^k and M^C are latin submatrices. To show that the resulting matrix M is a latin square, we need to show that no row contains twice the same id. By definition of $B_i \cap A_i = \emptyset$. So M is a latin square, and therefore the corresponding failover scheme is correct, i.e., no loops occur as each node appears at most once per row of the matrix. Since M is a latin square satisfying the conditions of Theorem 9, we conclude that for a load up to $\phi \le k \le \sqrt{n}$, the number of failed links is $\theta(\phi^2)$. This implies asymptotical optimality for All-to-One routing by matching the lower bound of Thm 2. □

Thanks to Theorem 11 we can build a load-optimal failover scheme given a suitable BIBD. In order to construct the corresponding BIBDs (for $k - 1$ being a prime tower), we can leverage the following theorem.

Theorem 12 (Thm 2.10 in [32])**.** *For every prime power $q \ge 2$, there exists a symmetric $(q^2+q+1, q+1, 1)$-BIBD (a projective plane of order q).*

Using these BIBDs, we can thus construct failover matrices for $n = q^2 + q + 1$ directly. If there exists no prime power q for which $n = q^2 + q + 1$, we can construct a failover matrix as follows. Choose r such that $2^{2r}+2^r+1 \le n < 2^{2r+2}+2^{r+1}+1$. Construct the failover matrix M with a $(q^2 + q + 1, q + 1, 1)$-BIBD for $q = 2^r$. Assign each row of this failover matrix to at most 4 nodes. The remaining $n - 2^{2r} + 2^r + 1$ elements of each sequence are chosen among the permutations of the nodes not used yet to guarantee a loop-free behavior. Using this construction, the load deteriorates by at most a factor of 4, as every prefix is used in at most three other rows.

F. Supporting Other Routing Schemes

1) Resilient Permutation Routing: Having discussed the All-to-One model, we now turn to the permutation routing problem. Permutation routing is a classic and well-studied scenario (e.g., in the context of oblivious routing and Valiant's trick [25], [35]) where given a (worst-case) permutation $\pi : V \to V$, each node v communicates with its image $\pi(v)$. This corresponds to a set of n flows with source v_i and destination $\pi(v_i)$. Hence, in a resilient setting, each flow needs a backup sequence to reach its destination $\pi(v_i)$ for a permutation π. Again, for each flow, we set the conditional failover rules according to the rows of a matrix M.

Note that the permutation routing problem has a fundamentally different structure from all-to-one routing and adversarial link failure strategies have to take all links into account, while for all-to-one routing the adversary can focus on the nodes to induce a high load. Nevertheless, we can apply the BIBD construction presented above to generate efficient failover matrices for this problem as well. We can even re-use the proof structure for the failure set size necessary for a certain load. Since every flow has a different destination it is more difficult for an adversary to reuse link failures and thus we can prove a higher bound than for all-to-one routing.

Theorem 13. *Let $k \le \sqrt{n}$ and M^k a latin submatrix where the intersection size of any two rows is at most 1, i.e., $\forall i, j \le n, i \ne j$ it holds that $|M^k(i) \cap M^k(j)| \le \lambda$. Let $F \in F_o(\phi)$. If $\phi \le k$ then $f = \Omega(\phi \cdot \sqrt{n})$ for permutation routing.*

Proof. Let (w, u) be a link that carries a load of ϕ due to F. Consider the set of affected failover sequences that contribute to this load, denoted by the set of flow T. Observe that $|T| = \phi$. The node w can be the source of at most one flow. Analogously, u is the destination of at most one flow, thus there are at least $\phi - 2$ affected rows in the BIBD failover matrix M with w in the prefix of u and a link failure for each element of those prefixes of u (note that w cannot be the destination of the flows of these rows, as then they would not contribute to a load exiting w). We now need to show that the size of the set F of these link failures is at least $\Omega(\phi \cdot \sqrt{n})$. Due to the prefix intersection properties of the matrix structure we use (Theorem 9), it must thus hold that the prefix length of u exceeds $\sqrt{n}$ for these $\phi - 2$ rows.

To have reached v in such a prefix it must hold that either the link (v_i, v) or a link (v', v) failed, for an element v' in the prefix of v on row i. To reuse a failure of the first type in flows, v_i must occur in the prefix of w in other rows. Again due to the matrix structure (Theorem 9), a multiple reuse of such a failure is hence only possible if the prefix of the reusing rows is at least $\sqrt{n}$ long. The multiple reuse of the second type of failure has the same implication. Thus at least half of the failures affecting the prefixes used are unique. In other words, the failures for the first $\sqrt{n}$ elements of the rows can only be reused at most once and thus $\phi \cdot \sqrt{n}/2 = \Omega(\phi \cdot \sqrt{n})$ failures are necessary. □

2) Arbitrary Traffic Patterns: With these solutions in mind, we are now ready to present our main contribution: a resilient failover routing scheme for arbitrary traffic patterns (for n

flows), i.e., the flows are not restricted to share the same destination nor do we limit the number of flows with the same source.

Given a list of flows, let $\delta^o(v)$ and $\delta^i(v)$ count the number of flows originating from v and destined to v respectively. The maximum values of these quantities is denoted by δ^o and δ^i. If we consider the directed multigraph induced by the list of flows, δ^o and δ^i correspond to the out-degree and in-degree of this multigraph. Using these definition, we show a general lower bound of failures necessary for arbitrary flow sets.

Theorem 14. *Given a $BIBD$-failover matrix M, $\Omega(\phi^2)$ link failures are necessary to generate a load of $\phi < \sqrt{n}$ regardless of the number of flows that share sources and destinations.*

Proof. Let us first consider the case where $\delta^o = 1$, i.e., there is at most one flow originating from each node. This first part of the proof is along the same lines as the first part of the proof of Thm. 13. Let link (w, u) be the link where the maximum load manifests. Node w can be the source of at most one flow, thus there are at least $\phi - 1$ rows (set T) in the BIBD failover matrix M with a link failure for each element of the prefix of w in those rows (note that w cannot be the destination of the flows of these rows, as then they would not contribute to a load exiting w). We now need to show that the size of the set F of these link failures is at least $\Omega(\phi^2)$. We pick one element v in the prefix of w on row i, i being one of the $\phi - 1$ rows in T responsible for the load. For such a link failure to be reused in another row, v would have to appear in the prefix of w in another row. However, in this case, the length of the prefix of w must exceed $\sqrt{n}$, because there cannot be two elements that appear in two prefixes of shorter size, due to the construction of M (Thm. 11). Thus we have two cases where either more than $(\phi - 1)/2$ of the rows in T have i) prefixes of w shorter than $\sqrt{n}$, in which case the necessary number of failures is $\Omega(\phi^2)$ due the argument above, or *ii)* there are more than $(\phi - 1)/2$ rows in T with long prefixes. For the portion of the prefixes of length $\sqrt{n}$ we can use the same argument as before, leading to a number of link failures in $\Omega(\phi \cdot \sqrt{n})$ which clearly exceeds $\Omega(\phi^2)$. If we have several flows originating from the same nodes, then adapting the above analysis leads to at least $\phi - \delta^o$ rows with link failures in the prefixes of w (for the at most δ^o flows originating from w this does not hold, hence we exclude them). Thus this proves that $\Omega((\phi - \delta^o)^2)$ failures are necessary. For the case where $\delta^o > \phi/2$ we could encounter a scenario where $\phi/2$ flows with source w contribute to the highest load on (w, u). Since we only consider the load caused by failover, the destination of these flows cannot be u, as in this case the flows would not contribute to the worst case load. Hence, we focus on the link failures necessary to reach u in the affected rows. In this case, the prefixes of u are of interest and using the same argument as above, the number of link failures can be lower bounded by $\Omega(\phi^2)$ as well. □

When we have a bound on δ^o and δ^i for a flow set, we can prove an even higher bound.

Theorem 15. *Given a $BIBD$-failover matrix M, $\Omega((\phi - \delta^o - \delta^i) \cdot \sqrt{n} + \phi^2)$ link failures are necessary for a load $\phi < \sqrt{n}$.*

Proof. Similarly to the proof before, we first consider the case where δ^i is one, i.e., there is at most one flow destined to each node. Let link (w, u) be the link where the maximum load manifests. Thus there are $\phi - 1$ rows (set T) in the BIBD failover matrix M with a link failure for each element of the prefix of u in those rows (there can be one row where u is the target and does not need to appear in the prefix).

We now lower bound f. To contribute to the load, either *i)* w must appear in the prefix of u on at least $\phi - 1 - \delta^o$ rows of T or *ii)* w must be the source of the flow of the remaining at most δ^o nodes. Consider *i)* first. w and u can only appear in one BIBD-block together, thus there are $\phi - 2 - \delta^o$ rows in T where only w can appear in the first $\sqrt{n}$ elements of the rows. The arguments of the proof for Thm. 14 apply here as well and thus at least $(\phi - 2 - \delta^o)\sqrt{n}$ link failures are accumulated for the $\sqrt{n}$-length prefixes of T. For *ii)* where w is the source, only u needs to be in the prefixes, contributing to lower bound of $\Omega(\phi^2)$. Thus the necessary size of F of both cases together is $\Omega((\phi - \delta^o)\sqrt{n} + \phi^2)$. For flow sets where up to δ^i flows share a destination, the number of rows with w and u in the prefix is further reduced, concluding the proof. □

Our approach can be extended for more than n flows, parametrized by the number of failures to be tolerated. We describe next a construction that can be used in this case. A prerequisite is the following observation. For failover matrix with more than n rows we cannot construct a latin submatrix M^k, as we have only n elements to fill the matrix with. However, when maintaining the low intersection size, we can keep the number of failures needed for a certain load high. Consider for example the case when each element can occur twice in each column, but the pairwise intersection of the first k elements of two rows is still one. In this case we can use the same arguments as in the proof of Theorem 9 to show that the number of failures necessary is quadratic in the resulting load. Hence we can split BIBD blocks into smaller blocks and use Algorithm 3 to build failover matrices for more flows with the same load behavior, albeit tolerating fewer failures. More precisely, given a $(q^2 + q + 1, q + 1, 1)$-BIBD for $q = n^{1/2}$ we can construct a $(2^{3/2 \log n - \log k}, k, 1)$- BIBD by partitioning each block into $(q+1)/k$ disjoint subblocks. With these smaller blocks, we can use the same approach as before, for k times more flows. With this BIBD the number of failures to be tolerated for $O(n^{3/2})$ is in the order of $\log^2 n$, resulting in an overhead load in the order of $\log n$.

Corollary 1. *Given λn flows, it holds that $\Omega(\phi^2)$ link failures are necessary to generate an overhead load of $k\phi < k\sqrt{n}$. If $\lambda \in O(\sqrt{n})$, it holds that $\Omega(\phi^2)$ link failures are necessary to generate an overhead load of $\phi < \log n$.*

G. Discussion and Remarks

Single Failover Table. First, we note that our approach requires only one failover table: in OpenFlow terms, a group table with entries of type "Fast Failover". Each group table entry contains a list of buckets, each bucket with a possible outgoing port. To support all possible n^2 source-destination pairs, we will need n^2 groups and n buckets in each group. When a Fast Failover

group is applied, the first bucket that contains a working port is activated, and this port is used to forward the packet. Thus, our approach is different from other approaches (e.g., link reversal routing) which require a state (i.e., store information) in data-plane devices, and which we dismissed for this reason.

Generalization To Approaches That Match Inport. We also note that our load lower bound even holds for failover schemes which support the matching of the inport: the proof does not make any assumptions on how the scheme calculates the failover rules but only assumes that after a failure, the traffic will be rerouted via some alternative node (which is known to the adversary). Extending our matrix-based failover schemes to utilize incoming port information however is non-trivial and an interesting direction for future research. Having that said, we also note that the disadvantage of schemes that use the incoming port is that depending on the implementation, this information may not always be available to the failover application, as it relies on the hardware to expose the information to the upper layers. In contrast, the source-destination information is always available as it is part of a packet header.

Complexity and Runtime. In terms of rule complexity, our approach requires rules that depend both on source and destination, which implies a quadratic number of rules (compared to linear in case of destination-based routing). This is an inherent price one has to pay in order to obtain such a high degree of resilience while keeping load low: if destination-based rules only are used, then there is a much higher (and inherent) lower bound on the load. In terms of runtime, the time taken to compute the load-optimal deterministic failover scheme is dominated by the BIBD computation; all other parts of our algorithm (e.g. the matching) can be computed very fast. Our randomized failover scheme is easier to generate but it suffers from a logarithmic factor in the lower bound for the minimum failure set size with respect to load.

VI. Simulations

We complement our formal analysis with a simulation study. In particular, we shed light on the load distribution in different failure scenarios and under different alternative routing schemes. Furthermore, we explore the performance of a BIBD-based approach in Clos networks, a topology designed for datacenters. To give an indication of the resources required, generating a BIBD-based failover matrix for a network of 200 nodes takes less than 3 seconds on a desktop machine, given a suitable BIBD as an input. Computing this BIBD is more computationally expensive, e.g., creating the corresponding (183,14,1)-BIBD takes around 2 hours on the same machine. However, it only has to be computed once and can be used for networks of varying size.

A. Random Failures

Improved load compared to state-of-the-art. We first investigate random failures, to model more "average case" rather than worst-case failures. Figure 5, top left, shows the maximum link load across all links, depicting the median (line), the maximum (dots) and the interquartile range (error bar) over 100 independent runs, for 200-node networks for all-to-one routing as a function of the number of failures. Clearly, even in the presence of a large number of concurrent failures, using our approach, the max load is low compared to the theoretical possible maximum of 200. More precisely, failover sequences with BIBDs incur a maximum load of less than 6 on average, even if almost 2/3 of the links failed. Even though operating beyond the $n-1$ tolerated failures studied in Theorem 9, our scheme performs well under larger failure sets. For comparison, the stochastic failover scheme based on random permutations proposed in Section V-B (indicated as "*RAND*" in the figures) does not perform as well as the failover scheme based on BIBDs. In addition, the BIBD approach provides deterministic guarantees, and not just probabilistic ones. Figure 5, top right, shows the corresponding results for permutation routing. Under permutation routing, the load is much lower.

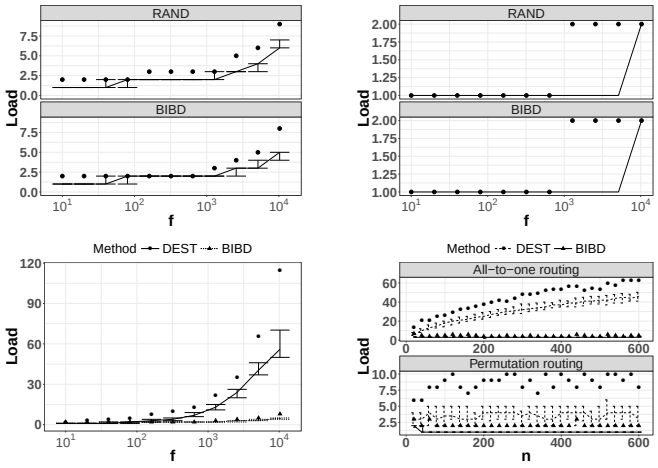


Fig. 5. Comparison of maximum link load induced by random failures with a BIBD scheme, a random permutation scheme (*RAND*) and a random destination-based scheme (*DEST*) for a network of 200 nodes, where not noted otherwise. The x-axis represents the number of link failures f, the y-axis the maximum link load overhead. The dots represent the maximum link load over 100 experiments, the line represents the median, and the bars indicate the interquartile range ($Q3-Q1$). *Top left:* All-to-one routing and random link failures, BIBD vs RAND. *Top right:* Permutation routing and random link failures, BIBD vs RAND. *Bottom left:* All-to-one routing and random link failures, BIBD vs DEST. *Bottom right:* All-to-one and permutation routing and random link failures, for networks with 20 to 600 nodes. In the former $n/2$ random links failed, while in the later $n^{3/2}$ links are down in the experiments.

The power of oblivious routing and remark on destination-based routing. Next, we investigate to what extent our approach benefits from the high path diversity offered by the oblivious BIBD routing policy, where (failover) paths can be arbitrary (and not only destination-based). For comparison, we consider destination-based routing (as it commonly used in legacy IP-networks): destination-based routing schemes are confluent, i.e., once two flows toward the same destination intersect, they will use the same remaining path (the suffix). Observe that in order to implement destination-based routing, we need to set all rows in the failover matrix to the same permutation for all-to-one routing. For random link failures, the best destination-based failover strategy is a randomly chosen permutation. As can be seen in Figure 5 bottom left, if routing is destination-based (referred to as "*DEST*" in the figure), the resulting link load is significantly higher in the all-to-one

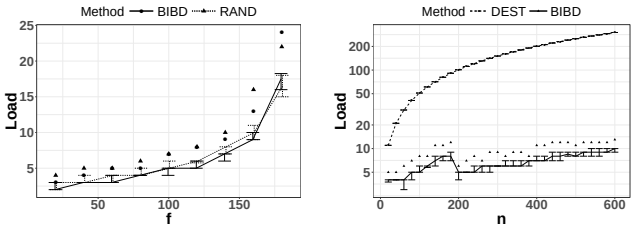


Fig. 6. Comparison of maximum link load induced by targeted failures with a BIBD scheme, a random permutation scheme (*RAND*) and a random destination-based scheme (*DEST*) *Left:* All-to-one routing and targeted link failures, BIBD vs RAND for a network of 200 nodes. *Right:* All-to-one routing and $n/2$ targeted link failures for networks of size n between 20 and 600 nodes with a logarithmic scale for the load.

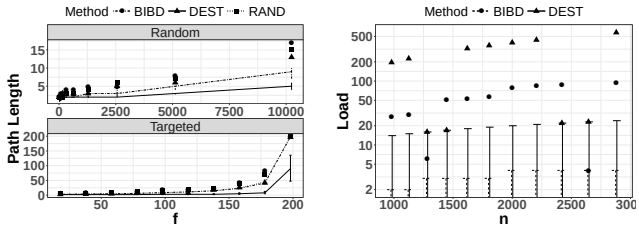


Fig. 7. *Left:* Path length measurements on complete networks of 200 nodes, for varying failure scenarios. Each scenario has been evaluated over 100 all-to-one traffic simulations. *Right:* Maximum load measurements on clos networks of switches with $p = 26$ to 48 ports, containing 980 to 2,880 nodes. We conducted 100 traffic simulations with 5,488 to 27,648 flows from top-of-rack switches to one destination with $p/2$ random (top) or targeted (bottom) failures for each size n. Points (maximum), lines (median) and grayed areas (interquartile range) are highlighted for these 100 values.

scenario. Also when varying the size of the networks under scrutiny and when studying permutation routing, see Figure 5 bottom right, the load under destination-based routing is much higher. Accordingly, we conclude that the higher path diversity offered by routing that takes the source into account in addition to the destination is vital for resource-efficient failover.

B. Targeted Failures

We now turn our attention to scenarios with adversarial failures, Figure VI-A. Indeed, we believe that the key strength of our approach lies in such more challenging failure scenarios. We consider an adversary that targets a particular node v and fails f random links incident to this node v. In other words, the adversary specifically targets the links of one node. For all-to-one routing, the chosen node is the destination node v_n, for permutation routing any node can be picked.

Note that all rows of the BIBD failover matrices offer the same properties due to the fact that they are generated from symmetric BIBDs and form a latin square. As shown, the maximum load is generated by failing links incident to v_n for all-to-one routing.

Improved load for all-to-one and permutation routing. Figure VI-A, left, plots the maximum load observed on any link as a function of the number of failures up to half the network size.[4] Unlike in the random failure experiments discussed above, we now see that the load grows more quickly with an increasing number of failures. Indeed, the results are reminiscent of the formal worst-case analysis presented in the previous section.

When the failover matrix is constructed with random permutations per row, the number of failures necessary to generate a maximum load of ϕ is in $\Omega(\phi^2/\log n)$.

Furthermore we study the load for networks of different size. In Figure VI-A, right, the load in networks with a size between $n = 20$ and $n = 600$ nodes and $n/2$ targeted failures is depicted for BIBD and destination-based routing. Note that the BIBD load grows more quickly until $n = 180$ and starts growing from a lower level with $n = 200$. This is due to the fact that for $n < 183$ a BIBD for $133 = 11^2 + 11 + 1$ elements is used in the construction of the failover matrix, thus the first 11 elements of more and more rows are re-used when n grows. For $n > 183$ a BIBD of $183 = 13^2 + 13 + 1$ elements is used in the failover matrix construction, which of course offers its best performance for values of n close to 183. Note that asymptotically the load bound of $\sqrt{\phi}$ in the number of failures ϕ holds, hence we did not add any optimizations for values $n < 183$. The load for destination-based routing grows linearly in the number of nodes (note the logarithmic scale of the y axis), i.e., the load is $n/2$ and thus much higher than with BIBD-based routing.

Under permutation routing the load is lower and BIBD achieves a more balanced link load than the randomized approach, as additional experiments not shown here demonstrate.

Length overhead When links fail and flows need to be rerouted, the lengths of the resulting paths are of interest too. In our experiments we also measure the maximum path length in 200 nodes networks under failure scenarios of varying intensity. Figure VI-B (left) present those results for both random and targeted failures. In both scenarios, all schemes present a steady increase in the maximum path lengths as the number of faults increases. BIBD and RAND are slightly below twice the path length of destination-based routing, because the variety of source-based failover sequences leads to higher maxima.

C. Clos Networks

To complement our study of complete networks with sparser topologies, we also analyzed the performance of our rerouting scheme for Clos networks [1]. In this popular data center network topology, two layers of aggregation switches are organized as independent pods that are redundantly interconnected through a layer of backbone switches. We hereafter describe this architecture, parametrized only by k, the number of ports per switch.

The two aggregation layers are divided in k pods that share identical configuration. Each pod contains a total of k switches, half on each layer. Each pod's first layer consists in $k/2$ switches connected to $(k/2)^2$ clients, and the second layer consists in $k/2$ switches connected to the backbone switches. In addition, those two layers are fully connected: a switch of the first layer has $k/2$ links to each of the second layer switches, and vice versa. Each of the $(k/2)^2$ backbone switches connect to exactly one layer 2 switch in each pod. This construction therefore contains $(k/2)^2 + k^2$ switches, and can accommodate up to $k^3/4$ clients.

[4]For a linear number of failures ($O(n)$), the load will be in the order of $\sqrt{n}$ for targeted failures in the all-to-one case. Therefore higher failure numbers are not interesting. (For more failures there will no longer exist a route.)

When a packet of flow i arrives at a switch and its default forwarding link has failed, a failover matrix can be used to determine the next hop of the rerouting strategy.

Figure VI-B(right), illustrates how the load increases for growing networks under a BIBD and destination-based failover strategy, with $p/2$ targeted link failures where p is the number of ports (and hence the maximum degree) of a switch. The switches feature $p = 28$ to 48 ports, leading to a network size of 980 to 2880 switches and 5488 to 27648 client machines (generating the traffic). The median of the maximum load remains one for BIDB and destination-based routing, while the maximum of the maximum load grows to 100 with BIBD and 500 for destination-based routing. The former is moderate given the number of flows and demonstrates that the BIBD approach is able to balance load successfully despite failures.

VII. Related Work

There exist several empirical studies showing that link failures, even simultaneous ones, do occur in different networks [20], [34], including wide-area [14] and data center networks [12]. For example, it has been reported that in a wide area network, a link fails every 30 minutes on average [16].

Commercial networks today usually rely on routing schemes such as ECMP, OSPF, IS-IS, and MPLS to reroute traffic, which do not come with formal guarantees under multiple failures. Accordingly, backbone networks are usually over-provisioned.

Existing resilient routing mechanisms can be classified according to whether a single link/node failure [10], [22], [37], [38] or multiple ones can be tolerated [8]. Alternatively, they can be classified into static and dynamic ones. Dynamic tables and using link reversals [11] can yield very resilient networks, but dynamic tables are not supported in OpenFlow switches today. Finally, one can also classify existing mechanisms as basic routing schemes [5], schemes exploiting packet-header rewriting (as e.g., failover in MPLS networks [26]), and routing with packet-duplication [13]. While packet-header rewriting can improve resiliency, it can be problematic in practice, especially under multiple failures, as header space and rule space is limited. Hence we do not use header rewriting. Also, we do not allow for flow splitting, which can reduce network loads but raises issues with granularity and packet reordering [16].

The works closest to ours are by Feigenbaum et al. [15], Chiesa et al. [6], and Stephens et al. [30], [31]. Feigenbaum et al. [15] introduces the notion of *perfect resilience*, resilience to arbitrary failures. Chiesa et al. [6] focus on "scalable" static failover schemes that rely only on the destination address, the packets incoming link, and the set of nonfailed links incident to the router. The authors find that per-incoming link destination-based forwarding tables are a necessity as destination-based routing alone is unable to achieve resilience against even a single link failure, and, moreover, entails computationally hard problems. In [18], Chiesa et al. consider randomized algorithms for static routing schemes whose rules depend only on the inport (where the packet arrives) and the destination.

Stephens et al. [30], [31] present a new forwarding table compression algorithm called Plinko, which however cannot provide resilience guarantees in all possible failure scenarios.

However, in contrast to our paper, none of these papers studies the implication on the *network load* of different failover mechanisms: an important concern in traffic engineering. Moreover, existing work often focuses on destination-based routing algorithms only, which inherently entails high loads (as shown in this paper) and also ignores one of the key advantages of software-defined networks in terms of traffic engineering. Finally, much existing work (e.g., based on randomized or stateful routing) is not OpenFlow-compatible.

One contribution of our paper is to observe a connection to the field of local algorithms without coordination. Accordingly, in terms of techniques, the paper closest to ours is by Malewicz et al. [19], as well as the seminal work by Dolev et al. [7]. The authors study scheduling for "disconnected cooperation": in their setting, a set of initially isolated, distributed processors need to schedule their work *before* starting communication. The goal is to come up with a deterministic schedule which minimizes the number of redundantly executed tasks: the so-called *waste*. This applies in decentralized environments where processors may meaningfully carry on with the computation regardless of any other component (e.g., due to the idempotency of tasks). Given a set of n nodes and $n < t$ tasks, where n is a prime power, Malewicz et al. present a deterministic, design-theoretic construction of an optimal schedule.

Bibliographic note. Preliminary results of this paper have been presented at OPODIS 2013 [4] and at DSN 2017 [23].

VIII. Conclusion

In order to guarantee connectivity, this paper leveraged an intriguing connection between local failover mechanisms and combinatorial block designs. In particular, we developed a deterministic failover scheme defining an almost optimal tradeoff between resilience and network load: the resulting bounds are off by a constant factor of the optimal bound. Our work hence settles an open question: while mechanisms such as Fast Reroute have been in place for many years, the fundamental tradeoffs regarding their level of resiliency and resource overheads such as load were long not well understood.

An attractive property of our approach is that the required number of failover rules is low: the number of rules only depends linearly on the number of failed links incident to the switch, and not on the number of possible combinations of possible link failures (which would be exponential). Interestingly, as we prove, despite this compact representation, we do not lose anything in terms of failover optimality with respect to the overhead load and fault-tolerance).

The main open question of our work regards solutions for sparse and specific networks.

Acknowledgments. The authors would like to thank Chen Avin for many discussions and inputs.

References

[1] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. *ACM SIGCOMM Computer Communication Review*, 38(4):63–74, 2008.

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

[3] 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.

[4] M. Borokhovich and S. Schmid. How (not) to shoot in your foot with sdn local fast failover: A load-connectivity tradeoff. In *Proc. 17th Conference on Principles of Distributed Systems (OPODIS)*, 2013.

[5] 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.

[6] 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.

[7] S. Dolev, R. Segala, and A. Shvartsman. Dynamic load balancing with group communication. In *Proc. SIROCCO*, pages 111–125, 1999.

[8] T. Elhourani, A. Gopalan, and S. Ramasubramanian. Ip fast rerouting for multi-link failures. In *Proc. IEEE INFOCOM*.

[9] T. Elhourani, A. Gopalan, and S. Ramasubramanian. Ip fast rerouting for multi-link failures. In *Proc. IEEE INFOCOM*, pages 2148–2156, 2014.

[10] G. Enyedi, G. Rétvári, and T. Cinkler. A novel loop-free ip fast reroute algorithm. In *Dependable and Adaptable Networks and Services*. 2007.

[11] E. Gafni and D. Bertsekas. Distributed algorithms for generating loop-free routes in networks with frequently changing topology. *Communications, IEEE Transactions on*, 29(1):11–18, Jan 1981.

[12] P. Gill, N. Jain, and N. Nagappan. Understanding network failures in data centers: measurement, analysis, and implications. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 350–361, 2011.

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

[14] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer. Achieving High Utilization with Software-Driven WAN. In *Proc. ACM SIGCOMM*, 2013.

[15] J. Feigenbaum et al. Ba: On the resilience of routing tables. In *Proc. ACM Symposium on Principles of Distributed Computing (PODC)*, 2012.

[16] H. H. Liu, S. Kandula, R. Mahajan, M. Zhang, and D. Gelernter. Traffic engineering with forward fault correction. In *Proc. ACM SIGCOMM Computer Communication Review*, volume 44, pages 527–538, 2014.

[17] V. Liu, D. Halperin, A. Krishnamurthy, and T. E. Anderson. F10: A fault-tolerant engineered network. In *Proc. USENIX NSDI*, 2013.

[18] M. Chiesa et al. On the resiliency of randomized routing against multiple edge failures. In *Proc. ICALP*.

[19] G. Malewicz, A. Russell, and A. A. Shvartsman. Distributed Scheduling for Disconnected Cooperation. *Distributed Computing*, 18(6), 2005.

[20] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C.-N. Chuah, and C. Diot. Characterization of failures in an ip backbone. In *Proc. IEEE INFOCOM*, volume 4, pages 2307–2317, 2004.

[21] M. Menth, M. Duelli, R. Martin, and J. Milbrandt. Resilience analysis of packet-witched communication networks. *IEEE/ACM transactions on Networking (toN)*, 17(6):1950–1963, 2009.

[22] S. Nelakuditi, S. Lee, Y. Yu, Z.-L. Zhang, and C.-N. Chuah. Fast local rerouting for handling transient link failures. *IEEE/ACM Transactions on Networking (ToN)*, 15(2):359–372, 2007.

[23] Y.-A. Pignolet, S. Schmid, and G. Tredan. Load-optimal local fast rerouting for dependable networks. In *Proc. 47th IEEE/IFIP Conference on Dependable Systems and Networks (DSN)*, 2017.

[24] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu. SIMPLE-fying Middlebox Policy Enforcement Using SDN. In *Proc. SIGCOMM*, 2013.

[25] H. Räcke. Survey on oblivious routing strategies. In *Proc. of the 5th Conference on Computability in Europe (CiE)*, pages 419–429, 2009.

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

[27] A. Shaikh, C. Isett, A. Greenberg, M. Roughan, and J. Gottlieb. A case study of ospf behavior in a large enterprise network. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages 217–230. ACM, 2002.

[28] L. Shen, X. Yang, and B. Ramamurthy. Shared risk link group (srlg)-diverse path provisioning under hybrid service level agreements in wavelength-routed optical mesh networks. *IEEE/ACM Transactions on Networking (ToN)*, 13(4):918–931, 2005.

[29] 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. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 183–197. ACM, 2015.

[30] B. Stephens, A. L. Cox, and S. Rixner. Plinko: Building provably resilient forwarding tables. In *Proc. 12th ACM HotNets*, 2013.

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

[32] D. R. Stinson. *Combinatorial designs: constructions and analysis*. Springer Science & Business Media, 2007.

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

[34] D. Turner, K. Levchenko, A. C. Snoeren, and S. Savage. California fault lines: understanding the causes and impact of network failures. *ACM SIGCOMM Computer Communication Review*, 41(4):315–326, 2011.

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

[36] M. Walraed-Sullivan, A. Vahdat, and K. Marzullo. Aspen trees: balancing data center fault tolerance, scalability and cost. In *Proc. ACM CONEXT*.

[37] J. Wang and S. Nelakuditi. Ip fast reroute with failure inferencing. In *Proc. SIGCOMM Workshop on Internet Network Management*, pages 268–273, 2007.

[38] B. Zhang, J. Wu, and J. Bi. Rpfp: Ip fast reroute with providing complete protection and without using tunnels. In *Proc. IWQoS*, pages 1–10, 2013.

Michael Borokhovich received B.Sc. (2005), M.Sc. (2009) and Ph.D. (2013) degrees in Communication Systems Engineering from Ben-Gurion University in Israel. The main research topics included fast failover in OpenFlow SDN networks, distributed algorithms, and optimization. Between 2014 and 2015, he was a Postdoc at UT Austin in Texas where he worked on efficient algorithms for distributed graph engines. Between 2015 and 2017, Michael was with AT&T Labs-Research, where he worked on ONAP (Open Network Automation Platform), and VNFs (virtual network functions). Currently, Michael is with Amazon, where he builds innovative SDN solutions for AWS networking.

Yvonne-Anne Pignolet is a Principal Scientist at ABB Corporate Research, Switzerland, where she leads and contributes to projects in distributed systems and networking for automation. While the application domains vary (including power grids, mining, manufacturing, building automation), her work is centered around the efficient design, engineering and operation of networked systems. Before joining ABB in 2011, she was a post-doctoral researcher at IBM Research Zurich Laboratory, Switzerland, working on wireless sensor nodes, and at Ben Gurion University of the Negev, Beer Sheva, Israel, studying complex network evolution. She completed her Msc (2006) and PhD (2009) at ETH Zurich, Switzerland.

Stefan Schmid is Professor at the Faculty of Computer Science at the University of Vienna, Austria. He received his MSc (2004) and PhD (2008) degrees from ETH Zurich, Switzerland. In 2009, Stefan Schmid was a postdoc at TU Munich and the University of Paderborn, between 2009 and 2015, a senior research scientist at T-Labs in Berlin, Germany, and from the end of 2015 till 2018, an Associate Professor at Aalborg University, Denmark. His research interests revolve around fundamental and algorithmic problems arising in networked systems.

Gilles Tredan is a CNRS researcher at LAAS (Laboratoire d'Architecture et d'Analyse des Systmes) in Toulouse, France. He obtained a PhD degree in computer science from University of Rennes 1 in November 2009, under the supervision of Achour Mostefaoui. From January 2010 to September 2011, he worked as a postdoc in the FG Inet group, Berlin. Gilles likes graphs and algorithms.