Load-Optimal Local Fast Rerouting for Resilient Networks

Yvonne-Anne Pignolet[1] Stefan Schmid[2] Gilles Tredan[3]
[1] ABB Corporate Research, Switzerland [2] Aalborg University, Denmark [3] 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 presents a deterministic local failover mechanism which we prove to result in a minimum network load for a wide range of communication patterns, solving an open problem. Our mechanism relies on the key insight that resilient routing essentially constitutes a distributed algorithm *without coordination*. Accordingly, 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 $\Omega(n)$ link failures in an n-node network and in the worst-case, while always ensuring routing connectivity. In particular, we show that at least $\Omega(\phi^2)$ link failures are needed to generate a maximum link load of at least ϕ, which matches an existing bound on the number of link failures needed for an optimal failover scheme. We complement our formal analysis with simulations, showing that our approach outperforms prior schemes not only in the worst-case.

I. Introduction

Computer networks, including enterprise, datacenter, and carrier 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).

A. 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 [19], and link failures do happen concurrently [1], [6], [8]. Even without physically disconnecting the underlying topology, these link failures can cause *routing failures* disrupting communications between some hosts.

B. 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, sometimes in the order of seconds, leading to high packet loss rates [30].

While recent advances in routers have reduced reconvergence times to under a second for carefully configured networks using link state IGPs, this is still too high for critical Internet services which are sensitive to periods of traffic loss that are orders of magnitude shorter than this.

The problem is particularly cumbersome in the Wide-Area Network (WAN) which needs to be operated near capacity for efficiency [14], [15]. In case of an uninformed or slow failover, a single link failure can have a severe impact, in terms of buffer overflows and packet drops. Indeed, a more predictable and fast failover was also one of the key reasons for Google's move to SDN [28].

C. 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 where, upon a link failure, packets are sent along a precomputed alternate path without waiting for the global recomputation of routes. 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.

D. 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.

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, tagging comes with overheads (in terms of header space, additional rules, and time) and can also cause problems in the presence of middleboxes [21].

Indeed, in this paper we are interested in most 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 [23].

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. Moreover, it has recently been shown that there is an inherent tradeoff between the robustness and the resulting worst-case *network load* [3].

E. Our Contributions

This paper presents the design of a very resilient fast failover scheme, tolerating multiple link and node failures, while keeping the network load low (asymptotically matching an existing lower bound [3]). Our re-routing algorithms are oblivious, and do not require packet tagging.

We formally prove that our failover scheme provides an optimal resilience while minimizing link loads for many important traffic models, including the frequently studied permutation routing model [23], [29] or all-to-one routing [3], [5].

Our approach is based on the insight that resilient local failover mechanisms can essentially 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 [26] and present a novel failover mechanism building upon *symmetric block designs*.

We focus on oblivious routing, where all packets of the same TCP 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).

F. Background & Preliminaries

Our approach is very general, 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 could potentially 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 failover mechanism presented in this paper is based on combinatorial design theory [26]. 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 ones. We refer the reader to [26] for more background.

G. Organization

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

II. Problem Statement & Model

We assume an SDN-network $G = (V, E)$ with n OpenFlow switches (or simply *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 packets of a given flow[1] arriving at v.
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 have been pre-installed by the controller and are static.

We focus on *oblivious routing schemes* in this paper: in oblivious routing, the route of a packet does not depend on

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

other packets, and in particular, is independent of the load in the network.

We consider an initial network where all nodes are directly connected. The communication pattern C of the flows routed on the network is represented by a list of source and destination pairs of nodes. For simplicity we will call the i^{th} item in C flow i, with source s_i and destination d_i. For ease of presentation, we will assume that there are at most n flows in the first part of the paper and later show how to extend the approach 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 aim to 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 and permutation routing.

III. Characterizing Oblivious Resilient Routing Schemes

Our proposed failover scheme can be best described in the form of a matrix (similar to the one used by Borokhovich and Schmid [3] for all-to-one routing). 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 upcoming example in Figure 1).

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

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 $m_{i,\cdot}$ (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 $(m_{i,j}, d)$ 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 $(m_{i,j+2}, d)$ failed, the link $(m_{i,j+2}, m_{i,j+3})$ will be tried, etc.

Algorithm 1 Rerouting given a Failover Matrix M

```
Upon receiving a packet of flow $i$ at node $v$:
1: if destination not reached yet, $d_i \neq v$ then
2:    if $(v, d_i)$ available then
3:       forward packet to $d_i$
4:    else
5:       $j$ = index of $v$ in $i^{th}$ row + 1, $m_{i,j-1} = v$.
6:    while $m_{i,j} = s_i$ or $(v, m_{i,j})$ unavailable do
7:       $j = j + 1$
8:    forward packet to $m_{i,j}$
```

Note that if some failover node $m_{i,j}$ is visited by the i-th flow, many different link failures may be the cause. However, for a given flow (row i), it holds that in this case one link pointing to node $m_{i,1}$, one link pointing to node $m_{i,2}$, etc. are in the set of failed links.

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 simply ignores them when they occur. In this case, each row is a permutation of all nodes.

Figure 1 illustrates the use of the failover matrix for a flow from node 1 to node 6, when the links $\{(1, 6), (2, 6), (2, 3)\}$

[2]Obviously, a node which failed can no longer be reached. While our approach is more general, it is only interesting under node failures if the remaining connectivity is still high.

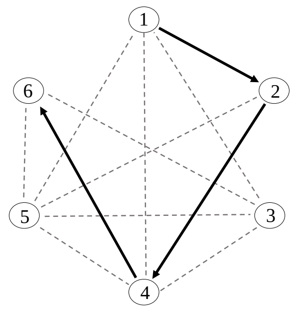


Fig. 1. Example: Rerouting of flow i from 1 to 6, according to M, where the i^{th} row without source and destination nodes is $[2, 3, 4, 5]$. If the links $\{(1, 6), (2, 6), (2, 3)\}$ failed, packets of this flow are first forwarded to node 2 from node 1. Since there is no direct link from 2 to 6, the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

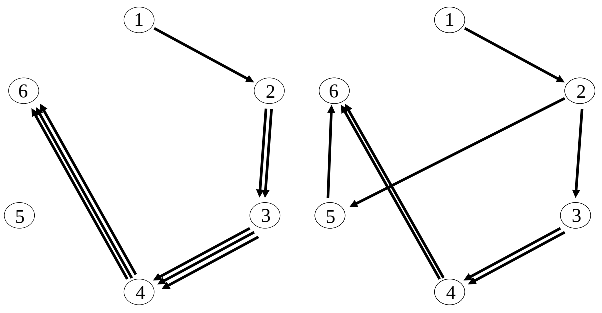


Fig. 2. Example: Rerouting of flows from nodes 1,2, and 3 due to link failures $\{(1, 6), (2, 6), (2, 3)\}$ according to M_1 (*left*) and M_2 (*right*) respectively. A failover matrix rerouting flows to similar paths can lead to a high overhead load (*left*). Accordingly, failover matrixes should be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

failed. Observe that while the specific permutation does not matter for correctness, it matters in terms of performance. Figure 2 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 $\{(1, 6), (2, 6), (3, 6)\}$ failed. On the left, the resulting failover routes for the following matrix are shown:

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

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.

IV. Resilient Oblivious Routing

A. The Case for Latin Square Failover Matrices

Before we present the proposed scheme 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 [3], [5]: 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). It is known from prior work that for all-to-one routing the highest overhead load is induced if links towards the destination node v_n are failed [3]. 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. 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 can simply refer to them by $i_1, i_2, \ldots$, whenever we consider 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 will 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 as high as possible (that is, t is used as a last resort), and ii) ensure that the failover routes used by u and v are as different as possible, ideally $P_t(u) \cap P_t(v) = \emptyset$: this prevents the adversary from "reusing" links failed on the failover path of packets from u when targeting the flow of packets from v.

When generalizing this brief analysis of node t 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. Led by this intuition, we now focus on **latin squares**: that is, failover matrices where each node appears exactly once on each row and each column. Since there are no repetitions on rows, forwarding loops in failover paths are avoided.

However, not all latin squares are good failover matrices. As an example, let us analyze the following latin square $M = [m_{ij}]_{1\le i,j\le n-1}$ 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 $\forall 1 \le j, j' \le (n-1), j \neq j' \Rightarrow m_{ij} \neq m_{ij'}$. However, this scheme results in a high load: if the adversary fails the f first links to destination d (that is, $F = \{(v_i, d), i = 1 \ldots f\}$), the f first nodes will all route through (v_{f+1}, d): we have $\phi = \theta(f)$.

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

B. *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 let $F \in F_o(\phi)$ be an optimal attack set (i.e., a worst-case set of link failures inducing a 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 $|F| \ge |\bigcup_{i\in T} P_w(i)| - 1$ (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 necessarily contains the shortest prefixes: $|F| \ge \sum_{i\in T}(|P_w(i)| - 1) = (t-1)(t-2)/2$, for $t = |T|$, because an occurrence of the destination 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 $M^k = [m_{ij}]_{1\le i<n, 1\le j\le k}$ 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

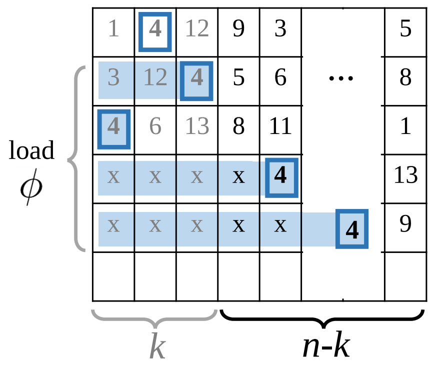


Fig. 3. Computation of load and worst-case failure sets. Let the flows $2, 3, 4, 5$ lead to the worst-case load $\phi = 4$ on the link $(4, v_n)$. In the failover matrix M, we highlight the occurrence of 4 with a square in each row, and color the background of the prefixes of the rows responsible for the load in blue. The number of failures responsible for 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, which allows us to bound the number of distinct elements.

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.

F	set of failed links
ϕ	maximum load induced by F
M	failover matrix $[m_{i,j}]_{1\le i<n, 1\le j<n}$, with element $m_{i,j}$ at the j^{th} position at the i^{th} row.
M^k	k-block / submatrix of the first k columns of M
$M^k(i)$	set of the first k elements of the i^{th} row of M
$P_v(i)$	set of nodes in the prefix of node v in row i

TABLE I
OVERVIEW OF NOTATION USED IN THIS PAPER

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 3 depicts an example of how load accumulates on a link.

Theorem 1. *Let $k \le \sqrt{n}$ and M^k a latin submatrix such that the size of the intersection of two rows is at most 1. That is, for $\forall i, j \le n, i \neq j$ it holds that $|M^k(i) \cap M^k(j)| \le 1$. Let $F \in F_o(\phi)$. If $\phi \le k$ 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 3, 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, $|P_w(i)| \le k$, 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 $|F| \ge |\bigcup_{i\in T} P_w(i)| - \phi$. Using the partition into T_1 and T_2 we have $|\bigcup_{i\in T} P_w(i)| \ge |\bigcup_{i\in T_1} P_w(i) \cup \bigcup_{i\in T_2} P_w(i)| \ge |\bigcup_{i\in T_1} P_w(i) \cup \bigcup_{i\in T_2} M^k(i)|$, 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.

C. *BIBDs* or: *How to create submatrices of low 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 1. 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 [26] 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 [26])**.** *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 collection (i.e., 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 exactly λ blocks.*

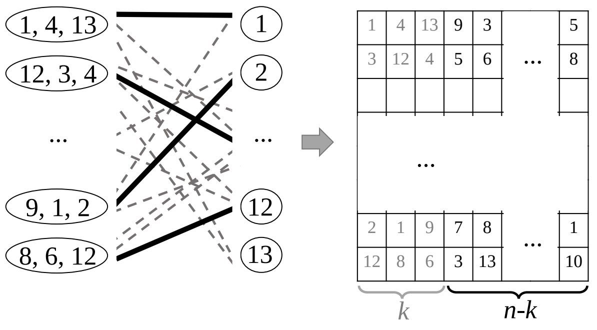


Fig. 4. Construction of BIBD failover matrix M for $n = 13$, $k = 3$. Given the blocks of the BIBD, a regular bipartite graph is constructed with edges (dashed) between the i^{th} block and j if j is in the i^{th} block (*left*). As a next step, a matching on this graph is computed (bold black edges) which defines the first column of M (*right*). After removing the edges of this matching, the procedure is repeated on the remaining graph (k times). When no more edges are left, the procedure is repeated for the complement of the graph computed first, to fill the $n - k$ last columns of the matrix.

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

Symmetric BIBDs feature a useful intersection property.

Fact 1 (Thm 2.2 in [26])**.** *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 $(n, k, \lambda)$-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  $M^k = [m_{ij}]_{1\leq i\leq n, 1\leq j\leq k}$
```

The transformation of blocks into rows is illustrated in Figure 4.

Theorem 2. *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, we observe that because the blocks are sets, no node is ever repeated in a row. Since P is a perfect matching, no node is repeated on a column. Consequently, M is a latin submatrix. $\square$

The construction of Algorithm 2 will be very useful to transform blocks into failover matrixes that provide the guarantees of Theorem 1.

D. Failover Matrix Creation

With the above we now construct a failover matrix M (summarized in Algorithm 3 and illustrated in Figure 4) 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 is sufficient 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 $M^k = [m_{ij}]_{1\leq i\leq n, 1\leq j\leq k} = Alg2(X,A)$
4: Let $M^C = Alg2(X, \{B_i, B_i = X \setminus A_i, 1 \leq i \leq n\})$
5: return  $M = M^k \oplus M^C$, where $\oplus$ concatenates columns
```

Theorem 3. *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, $(X, \{B_i, 1 \leq i \leq n\})$ is a BIBD (regardless of its intersection size), as the complement of a BIBD is also a BIBD ([26] 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 can occur as each node appears at most once per row of the matrix.

Since M is a latin square satisfying the conditions of Theorem 1, we conclude that for a load up to $\phi \leq k \leq \sqrt{n}$, the number of failed links is $\theta(\phi^2)$, which implies asymptotical optimality. $\square$

In order to construct the corresponding BIBDs (for $k-1$ being a prime tower), we can leverage the following theorem.

Theorem 4 (Thm 2.10 in [26])**.** *For every prime power $q \geq 2$, 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 $2^{2r}+2^r+1 \leq n < 2^{2r+2}+2^{r+1}+1$. 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 can be chosen among the permutations of the nodes that have not been used yet to guarantee a loop-free behavior. Using this construction, the load deteriorates by at most a factor of 4, since every prefix is used in at most three other rows.

E. 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 [23], [29]) 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 5. *Let $k \leq \sqrt{n}$ and M^k is a latin submatrix such that the size of the intersection of two rows is at most 1, i.e., $\forall i, j \leq n, i \neq j$ it holds that $|M^k(i) \cap M^k(j)| \leq \lambda$. Let $F \in F_o(\phi)$. If $\phi \leq k$ 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 1), 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 1), 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. □

F. 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 6. *Given a $BIBD$-failover matrix M, $\Omega(\phi^2)$ link failures are necessary to generate a failover 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. 5.

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. 3). 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. Therefore we can 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 in this case. □

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

Theorem 7. *Given a $BIBD$-failover matrix M, $\Omega((\phi - \delta^o - \delta^i) \cdot \sqrt{n} + \phi^2)$ link failures are necessary to generate a load of $\phi < \sqrt{n}$.*

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 of elements link failures).

We now lower bound the size of the set F of these link failures. 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. Let us 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. 6 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. In Case *ii)* where w is the source of flows, only u needs to be in the prefixes, contributing to lower bound of $\Omega(\phi^2)$. Thus the necessary failure set size 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 where w and u must in be in the prefix of failures 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. In this case, the following construction can be used. 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. 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 $\phi < \log n$.*

V. Simulations

We complement our formal analysis with a simulation study. In particular, in this section we shed light on the load distribution in different failure scenarios and under different alternative routing schemes.

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.a) shows the maximum link load (across all links, depicting the average, the maximum and the minimum over 100 runs, for a 183-node networks[3]) 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. 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 1, our scheme performs well under random failures. For comparison, especially for large failure sets, the stochastic failover scheme based on random permutations proposed in [3] (indicated as "*OPODIS*" in the figures) does not perform as well as the failover scheme based on BIBDs. In addition, our approach has the advantage of providing deterministic guarantees, and not just probabilistic ones.

Figure 5.b) shows the corresponding results for permutation routing. Under permutation routing, the load is much lower.

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). Nevertheless, 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. As can be seen in Figure 5.c), if routing is restricted to be destination-based (referred to as "*DEST*" in the figure), the resulting link load is significantly higher in the all-to-one scenario (note that the experiment is only interesting in this scenario, as destinations under permutation routing differ). Accordingly, we conclude that the higher path diversity offered by destination *and* source based routing is vital for a resource efficient failover.

[3]We perform our analysis on a network of 183 nodes, since there exists a (183,14,1)-BIBD which fits perfectly for this size. As discussed in the paper, any network size can be supported, but the construction becomes more cumbersome and a simulation on 150 or 200 nodes does not reveal more information than on 183 nodes.

B. Targeted Failures

We now turn our attention to scenarios with adversarial failures. 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 in [3], 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 5.d) plots the maximum load observed on any link as a function of the number of failures up to $n/2$[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 $\Omega(\phi^2/\log n)$ [3]. The deterministic BIBD failover matrix outperforms the random permutation matrix by around 20%. Under permutation routing the load is lower and BIBD achieves a more balanced link load than the randomized approach from [3].

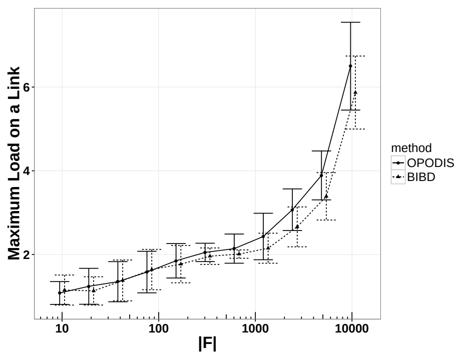
VI. Related Work

There exist several empirical studies showing that link failures, even simultaneous ones, do occur in different networks [19], [27], 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].

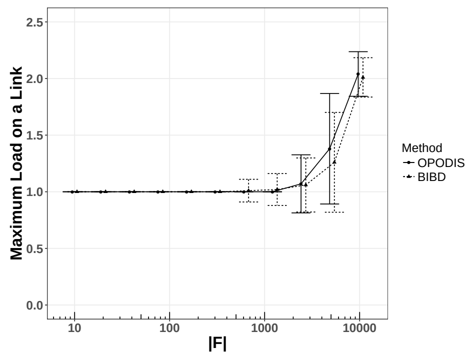
Commercial networks today usually rely on routing schemes such as OSPF, IS-IS, and MPLS reroute traffic, which however do not come with formal guarantees under multiple failures. Accordingly, backbone networks are usually largely over-provisioned.

Existing resilient routing mechanisms can be classified according to whether a single link/node failure [9], [20], [31], [32] 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 [4], schemes exploiting packet-header rewriting, 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. We in this paper hence do not use header rewriting.

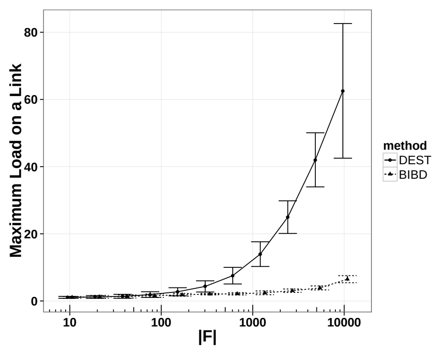
[4] For a number of failure in O(n) the load will be in the order of $\sqrt(n)$ for targeted failures in the all-to-one case, therefore higher failure numbers are not interesting



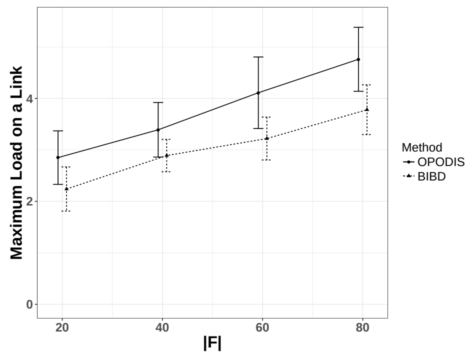
a) All-to-one routing and random link failures



b) Permutation routing and random link failures



c) All-to-one routing and random link failures



d) All-to-one routing and targetted link failures

Fig. 5. Comparison of maximum link load induced by our BIBD scheme, a random permutation scheme [3] (*OPODIS*) and a random destination-based scheme [3] (*DEST*) for a network of 183 nodes. The x-axis represents the number of random link failures, the y-axis the maximum link load. The dots represent the mean of the maximum link load over 100 experiments, the bars indicate the standard deviation.

The works closest to ours are by Feigenbaum et al. [10], Chiesa et al. [6], Stephens et al. [24], [25], and Borokhovich et al. [3]. Feigenbaum et al. [10] 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 [5], 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. [24], [25] 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 paper study 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, ignoring 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.

In this respect, the closest work to ours is [3]. The authors focus on the All-to-One model only and present a deterministic failover scheme which achieves an optimal tradeoff between resilience and network load for up to a logarithmic number of failures. For a larger number of failures, a randomized but non-constructive mechanism is proposed. Moreover, the authors prove the following two statements: *i)* 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, and *ii)* for any local failover scheme tolerating φ link failures $(0 < \varphi < n)$ without disconnecting any source-destination pair, there exists a failure scenario which results in a link load of at least $\widehat{\lambda} \geq \sqrt{\varphi}$, although the minimum edge cut (mincut) of the network is still at least $n - \varphi - 1$. In this paper, we provide a deterministic approach that matches these lower bounds asymptotically, thus solving this open problem. We also show in simulations, that we outperform their approach in all considered scenarios while additionally providing deterministic guarantees. More importantly, our results apply to general traffic matrices, beyond the All-to-One model.

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. [18], 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 model is motivated by 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.

The routing algorithms presented in this paper are *oblivious*: failover paths are independent of other packets. Accordingly, we believe that our work provides an interesting new perspective on oblivious routing: a field which to the best of our knowledge has so far mostly been studied for scenarios without failures. Oblivious routing schemes are attractive for their simplicity, but it is also known that such schemes come at an optimality price, if only a single path for every source-destination pair can be chosen: the famous Borodin-Hopcroft lower bound [2] states that for permutation routing, given a maximal node degree d, there is a permutation in which a node is traversed by at least $\sqrt{n/d}$ paths. While this lower bound also applies to our setting, due to the limited number of failures we can tolerate and the high remaining connectivity, it only provides weak bounds. On the positive side, it is known that if multiple paths are allowed, e.g., using Valiant's trick [29], lower loads can be achieved. We refer the reader to [2], [17], [22], [29] for more details. Interestingly, we show in this work that oblivious routing strategies can actually be asympotically optimal for fast failover routing.

VII. Conclusion

A highly available connectivity is the prerequisite for any dependable network-based application and service. Indeed, according to the CAP theorem, if and only if network connectivity is ensured, it is possible to provide both availability and consistency in a distributed system.

In order to guarantee connectivity, this paper leveraged an intriguing connection between local failover mechanisms and combinatorial block designs. In particular, we developed a 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 derived in prior work. 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.

Regarding the impact on flow table rules, one nice aspect of our approach is that the required number of failover rules is low: they only depend linearly on the number of failed links incident to the switch (whereas in principle one could imagine a scenario where there is a different failover strategy for each subset of failed ports). 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).

Nevertheless, our work leaves open several interesting directions for future research. For example, there remain many interesting opportunities to improve our approach in practice, and tailor it to specific use cases. For instance, we have so far treated all flows equally. However, in practice, it may make sense to use our rigorous routing scheme only for high-priority and critical flows, while low-priority flows could be routed in a best effort manner. Such a prioritization and the study of its tradeoffs is interesting but orthogonal to our work. At the same time, it holds that additional knowledge about the bandwidth and traffic matrix, also creates additional optimization opportunities, which we aim to explore in the future.

Acknowledgments. Research in part supported by the Danish Villum project *ReNet*.

References

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

[2] A. Borodin and J. E. Hopcroft. Routing, merging and sorting on parallel models of computation. In *Proc. ACM Symposium on Theory of Computing (STOC)*, pages 338–344, 1982.

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

[4] 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.

[5] M. Chiesa, A. V. Gurtov, A. Madry, S. Mitrovic, I. Nikolaevskiy, M. Schapira, and S. Shenker. On the resiliency of randomized routing against multiple edge failures. In *Proc. 43rd International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 134:1–134:15, 2016.

[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*, pages 2148–2156. IEEE, 2014.

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

[10] J. F. et al. Ba: On the resilience of routing tables. In *Proc. ACM Symposium on Principles of Distributed Computing (PODC)*, pages 237–238, 2012.

[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] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat. B4: Experience with a Globally-Deployed Software Defined WAN. In *Proc. ACM SIGCOMM*, 2013.

[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] B. M. Maggs, F. M. auf der Heide, B. Vocking, and M. Westermann. Exploiting locality for data management in systems of limited bandwidth. In *Proc. Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 284–293, 1997.

[18] G. Malewicz, A. Russell, and A. A. Shvartsman. Distributed scheduling for disconnected cooperation. *Distributed Computing*, 18(6):409–420, 2005.

[19] 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.

[20] 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.

[21] 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*, pages 27–38, 2013.

[22] H. Racke. Minimizing congestion in general networks. In *Proc. Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 43–52, 2002.

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

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

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

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

[27] 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.

[28] A. Vahdat, D. Clark, and J. Rexford. A purpose-built global network: Google's move to sdn. *Commun. ACM*, 59(3):46–54, Feb. 2016.

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

[30] M. Walraed-Sullivan, A. Vahdat, and K. Marzullo. Aspen trees: balancing data center fault tolerance, scalability and cost. In *Proc. ACM CONEXT*, pages 85–96, 2013.

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

[32] 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.