Local Fast Segment Rerouting on Hypercubes

Klaus-Tycho Foerster
University of Vienna, Vienna, Austria
klaus-tycho.foerster@univie.ac.at
https://orcid.org/0000-0003-4635-4480

Mahmoud Parham[1]
University of Vienna, Vienna, Austria
mahmoud.parham@univie.ac.at
https://orcid.org/0000-0002-6211-077X

Stefan Schmid
University of Vienna, Vienna, Austria
stefan_schmid@univie.ac.at
https://orcid.org/0000-0002-7798-1711

Tao Wen
University of Electronic Science and Technology of China, Chengdu, China
winterwentao@gmail.com
https://orcid.org/0000-0002-0772-5296

Abstract

Fast rerouting is an essential mechanism in any dependable communication network, allowing to quickly, i.e., *locally*, recover from network failures, without invoking the control plane. However, while locality ensures a fast reaction, the absence of global information also renders the design of highly resilient fast rerouting algorithms more challenging. In this paper, we study algorithms for fast rerouting in emerging Segment Routing (SR) networks, where intermediate destinations can be added to packets by nodes along the path. Our main contribution is a maximally resilient polynomial-time fast rerouting algorithm for SR networks based on a hypercube topology. Our algorithm is attractive as it preserves the original paths (and hence waypoints traversed along the way), and does not require packets to carry failure information. We complement our results with an integer linear program formulation for general graphs and exploratory simulation results.

2012 ACM Subject Classification Networks → Routing protocols, Networks → Network reliability, Theory of computation → Design and analysis of algorithms

Keywords and phrases segment routing, local fast failover, link failures

Digital Object Identifier 10.4230/LIPIcs.OPODIS.2018.13

Acknowledgements We would like to thank David Lebrun for helpful discussions in the early stage of this paper.

1 Introduction

1.1 Motivation and Challenges

The need for a more reliable network performance and quickly growing traffic volumes led, starting from the late 1990s [19], to the development of more advanced approaches to control the routes along which traffic is delivered. Multipath-Label Switching (MPLS) was one of

[1] Contact and main author.

22nd International Conference on Principles of Distributed Systems (OPODIS 2018).
Editors: Jiannong Cao, Faith Ellen, Luis Rodrigues, and Bernardo Ferreira; Article No. 13; pp. 13:1–13:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

the first and most widely deployed alternatives to traditional weight and destination based routing (such as OSPF), enabling a per-flow traffic engineering. Recently, *Segment Routing (SR)* [38, 20] has emerged as a scalable alternative to MPLS networks: SR networks do not require any resource reservations nor states on all the routers part of the route (the *virtual circuit*). SR networks are also attractive for their simple deployment; in contrast to, e.g., Software-Defined Network (SDN) and OpenFlow-based solutions, they rely on existing protocols such as IPv6 [62].

We in this paper investigate how to enhance SR networks with *(local) fast rerouting* algorithms, to react to failures *without the need to invoke the control plane*. The re-computation (and distribution) of routes after failures via the control plane is notoriously slow [26] and known to harm performance [44]. Also link-reversal algorithms [27] tolerating multiple failures have a quadratic convergence time [7], besides requiring dynamic routing tables. This is problematic as certain applications, e.g., in datacenters, are known to require a latency of less than 100 ms [67]; voice traffic [33] and interactive services [35] already degrade after 60 ms of delay. Not surprisingly, reliability is also one of the foremost challenges for network carriers nowadays [66], and in the context of power systems (e.g., smart grids), an almost entirely lossless network is expected [58]. Accordingly, most modern communication networks (including IP, MPLS, OpenFlow networks) feature fast rerouting primitives to support networks to recover quickly from failures.

Designing a fast rerouting algorithm however is non-trivial, as reactions need to be *(statically) pre-defined* and can only depend on the *local* failures, but not on "future" failures, *downstream*. As link failures, also multiple ones, are common in networks [46], e.g., due to shared link risk groups or virtualization, it is crucial to pre-define the conditional local failover rules such that connectivity is preserved (i.e., forwarding loops and blackholes avoided) under *any* possible additional failures. In fact, in many networks, including SR networks, algorithms cannot even depend on already encountered failures *upstream*, as it requires mechanisms to carry and process such information in the packet header; such "failure-carrying packets" [22, 37] require additional and complex forwarding rules. Further challenges are introduced by policy-related constraints on the paths along which packets are rerouted in case of failures. In particular, failover paths may not be allowed to "skip" nodes, but rather should reroute around failed links individually: communication networks include an increasing number of middleboxes and network functions, so-called *waypoints* [2], which must be traversed for security and performance reasons. Without precautions, in case of a link failure, the backup path could omit these waypoints.

Ideally, a local fast rerouting algorithm preserves connectivity "whenever this is still possible", i.e., as long as the underlying network is still *physically* connected. In other words, in a k-(link-)connected network, we would like the rerouting algorithm to tolerate $k-1$ link failures. We will refer to this strong notion of robustness as *maximal robustness* in the following.

1.2 Example

Figure 1 illustrates an example for the problem considered in this paper: how to efficiently circumvent multiple link failures using SR local fast failover mechanisms, such that the original route will be preserved as part of the packets' new route, hence also ensuring waypoint traversal. In this example, while the backup path in dotted green reaches the destination, the middlebox w is not visited. Ideally, we want to circumvent the failed link and then continue on the original route (as depicted in the red dashed walk).

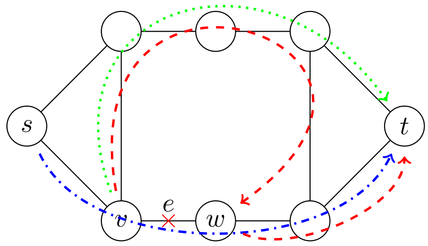


Figure 1 Illustration of two different fast failover mechanisms, upon hitting a link failure. The default path is depicted in dash-dotted blue. In the green dotted path, the destination t is reached, but the waypoint w is not traversed. The red dashed walk circumvents the link failure, traverses w, and then reaches t.

In case of only a single link $e = (v, w)$ failing, one can exploit that both nodes v, w have a globally correct view of the network link states. For example, if a packet hits the failed link e at v, the node v can provide an alternative path to w, after which the packet resumes its original path, as shown in dashed red in Figure 1. To this end, each node only needs to be provisioned with one alternative path for each of its incident links.

In Segment Routing, similar to MPLS, each packet contains a label stack, consisting of nodes or links. However, these labels just represent the next waypoint to be reached, the route ("segment") which depends on the underlying routing functions (e.g., shortest path). Once the top item is reached, the corresponding label is popped and the next item on the stack is parsed. As such, the next label does not need to be in the vicinity of the current node, it can be anywhere in the network. For the case of a single link $e = (v, w)$ failure, it has been shown that pushing two items on the label stack always suffices [25], if the network is still connected and a shortest alternative path is chosen.

While SR enables waypoint traversal even after a single link failure [25], dealing with multiple link failures in SR is still not well understood. As observed in [22], the option of choosing the shortest alternative path already fails under two link failures, see Figure 2; when e_1 fails (and the dash-dotted blue path as well), the packet will be sent along e.g. e_2, but upon the failure of e_2, the packet is sent along e_1—a forwarding loop (shown in dotted red). In this example, we can easily fix the reachability issues: a failure of e_2 causes rerouting along e_3 (in dashed green, not along e_1), and failure of e_3 causes rerouting along e_1. In other words, e_1 depends on e_2, which depends on e_3, which in turn depends on e_1. As this circular dependency chain has a length of three, two failures of $\{e_1, e_2, e_3\}$ cannot induce a forwarding loop when routing to w. We will later formalize and extend these ideas, generating dependency chains of length $\geq k$ for k-dimensional hypercubes.

1.3 Contributions

We initiate the study of fast reroute algorithms for emerging Segment Routing networks which are 1) resilient to a maximum number of failures (i.e., are *maximally robust*), 2) respect the path traversal of the original route, and 3) are compatible to current technologies in that they do not require packets to carry failure information: routing tables are static and forwarding just depends on the packet's top-of-the-stack destination label and the incident link failures.

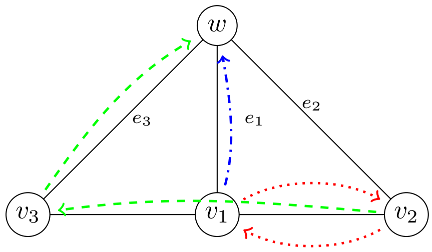


Figure 2 Example illustrating how local fast failover methods for a single link failure can loop under two link failures, as shown in [22]. When the dash-dotted blue default route between v_1 and w fails, v_2 can be pushed as a segment, to in turn reroute along e_2. However, when v_2 uses e_1 via v_1 as a failover for e_2, then failing both e_1 and e_2 leads to a permanent forwarding loop, as depicted in dotted red. In order to route successfully under both e_1, e_2 failing, v_2 has to push segments v_1, v_3 to route along e_3, as depicted in dashed green.

Our main result is an efficient algorithm which provably provides all these properties on hypercube networks, as they are commonly used in datacenters (see e.g., [48]). Furthermore, we formulate the underlying optimization problem as an integer linear program for general graphs, and provide first exploratory insights on the practical performance of segment routing under multiple link failures.

1.4 Organization

The remainder of this paper is organized as follows. We first introduce necessary model preliminaries in Section 2, followed by our main result in Section 3, where we provide a maximally robust SR failover scheme for k-dimensional hypercubes. We cover related work in Section 4 and conclude our study in Section 5, where we also provide further insights which we believe to be useful for future work, in the form of an integer linear program formulation for general graphs and a brief investigation regarding testbed experiments.

2 Model

In this section, we start by providing model and notation preliminaries. We will consider undirected graphs $G = (V, E)$, where the links may be indexed according to some (possibly arbitrary) ordering, with $\ell_i \in E$ denoting the ith link. All routing rules have to be pre-computed and may not be changed during the runtime (e.g., after failures). We will only allow routing rules that match on 1) the packet's next destination (i.e., the top of the label stack)[2], and the 2) incident link failures.[3] When a packet hits a failed link $\ell = (u, v)$ at some node u, the current node u may push a set of pre-computed labels on top of the current label stack, in order to create a so-called backup path to v (which can also be traversed in reverse from v to u).

▶ **Definition 1.** A *backup path* for a link ℓ is a simple path (not containing ℓ) that connects the endpoint of the link ℓ. Let $\mathcal{P}$ be the set of all backup paths in a graph. An injective function $BP : E \to \mathcal{P}$ that maps one backup path to each link is a *backup path scheme*.

---

[2] In practice, one could also imagine matching on other header fields, such as the packet's source, and also the incoming port. However, our algorithms do not require these additional inputs.
[3] In other words, only the endpoints u, v of the failed link $(u, v) = \ell \in E$ are aware of the failure.

When the packet reaches the current top label, the respective label is popped and the underlying label is set as top label. As such, via backup paths, the incoming packets that normally travel through ℓ are rerouted around the link to the respective endpoint, circumventing the failure. Hence, our model preserves the intermediate visits (i.e., all possible waypoints) and their order in a subset of the traversed route, possibly introducing repeated visits [1]. In the following, we will investigate backup path schemes that guarantee packet delivery even under multiple failures. To this end, we need to ensure that the backup paths do not contain infinite forwarding loops, for their specified maximum number of failures. More formally:

▶ **Definition 2.** A backup path scheme $BP(.)$ is called f*-resilient* if and only if there does not exist a subset of links $L \subseteq E, |L| \leq f$ such that for some ordering $\sigma : \{0, \dots, |L|-1\} \rightarrow \{0, \dots, |E|-1\}$, $\forall j < |L| : \ell_{\sigma(j+1 \pmod{|L|})} \in BP(\ell_{\sigma(j)})$. We refer to the inclusion relation ($\in$) as *dependency* from $\ell_{\sigma(j)}$ to $\ell_{\sigma(j+1 \pmod{|L|})}$. Equivalently, $BP(.)$ is f-resilient if and only if any *cycle of dependencies* is longer than f.

In the next section, we will show how to efficiently generate a $(k-1)$-resilient backup path scheme for k-dimensional hypercubes. As k-dimensional hypercubes are k-link-connected, our scheme has ideal robustness.

3 Efficient Resilient Segment Routing on k-Dimensional Hypercubes

This section presents a fast and maximally robust rerouting algorithm on hypercubes, one of the most important and well-studied network topologies [53, 64]. The regular structure of hypercubes makes them an ideal fit for e.g., parallel interconnection architectures [52] or datacenter [30].

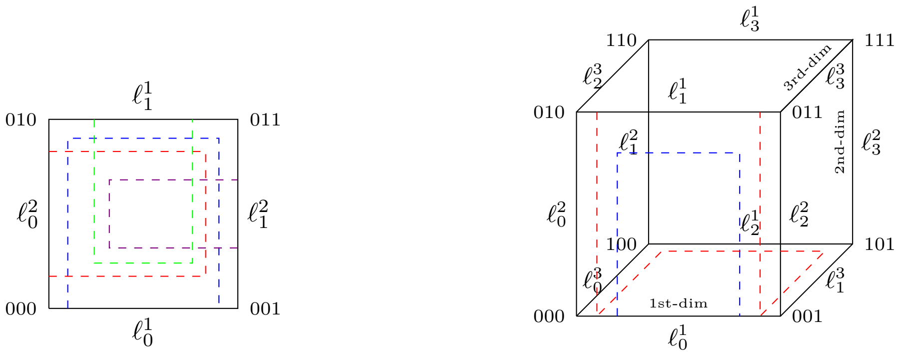
Our study on k-dimensional hypercubes is structured as follows: We first provide an intuition and overview of the $(k-1)$-resilient scheme in Section 3.1, providing a formal definition of all backup paths in Term 1. Next, in Section 3.2, we introduce some useful technical preliminaries for the correctness proof of our scheme, which is presented in Section 3.3.

3.1 Overview of the Fast Local Failover Scheme

We label the nodes in a k-dimensional hypercube (k*-cube*) with tuples $(b_k, b_{k-1}, \dots, b_1)$, $\forall i \in [k] : b_i \in \{0, 1\}$, such that the *origin* node has the label $\{0\}^k$. A hypercube link is denoted by an ordered pair of binary node labels (a, b) s.t. $a, b \in \{0, 1\}^k, a < b$, where the two labels differ in one bit. Additionally, a link is said to be in dimension $d, d \in [k]$, if and only if a and b differ only at their dth bit. We refer to them as d*-dim* links. For convenience, we treat a hypercube as a set of links grouped by their dimension, within each dimension sorted according to the following bitwise comparison. For $x \in \{0, 1\}^k$, let $x^{>>s} := x >> s$, where $>>$ is right circular shift. Let ℓ_i^d denote the ith link in dimension d, see Figure 3a. For $\ell_p^d = (a, b)$ and $\ell_q^d = (c, d)$, we have $p < q$ if and only if $a^{>>d} < c^{>>d}$. Lastly, we denote a k-cube by $C_k := \cup_{d,i} \ell_i^d, d \in [k], 0 \leq i < 2^{k-1}$.

The idea is to allocate backup paths in k iterations, one for each subset of links in the same dimension, such that the induced dependencies over same-dimension links form cycles of length at least k. However, since there are additional dependency cycles induced by links in different dimensions, we devise a scheme that does not induce any dependency cycle shorter than k (hence $(k-1)$-resiliency follows).

Due to gray coding, starting from any link $\ell_i^d = (a, b)$, by traversing the (unique) pair of incident d'-dim links, we reach the link $N_{d'}(\ell_i^d) = (a', b')$ such that $a' = (2^{d'-1})_2 \oplus a$



(a) 1-resilient scheme for the 2-cube, backup paths shown with dashed lines.

(b) Link and node labels for the 3-cube and sample backup paths shown with dashed lines.

$BP(\ell_0^1) = \{\ell_0^2, \ell_1^2, \ell_1^1\}$	$BP(\ell_0^2) = \{\ell_0^3, \ell_1^3, \ell_1^2\}$	$BP(\ell_0^3) = \{\ell_0^1, \ell_1^1, \ell_1^3\}$
$BP(\ell_1^1) = \{\ell_0^2, \ell_1^2, \ell_0^3, \ell_1^3, \ell_3^1\}$	$BP(\ell_1^2) = \{\ell_0^3, \ell_1^3, \ell_0^1, \ell_1^1, \ell_2^2\}$	$BP(\ell_1^3) = \{\ell_0^1, \ell_1^1, \ell_0^2, \ell_1^2, \ell_2^3\}$
$BP(\ell_2^1) = \{\ell_2^2, \ell_3^2, \ell_3^1\}$	$BP(\ell_2^2) = \{\ell_2^3, \ell_3^3, \ell_3^2\}$	$BP(\ell_2^3) = \{\ell_2^1, \ell_3^1, \ell_3^3\}$
$BP(\ell_3^1) = \{\ell_2^2, \ell_3^2, \ell_0^3, \ell_1^3, \ell_0^1\}$	$BP(\ell_3^2) = \{\ell_2^3, \ell_3^3, \ell_0^1, \ell_1^1, \ell_0^2\}$	$BP(\ell_3^3) = \{\ell_2^1, \ell_3^1, \ell_0^2, \ell_1^2, \ell_0^3\}$

(c) List of all backup paths for the 2-resilient scheme on the 3-cube.

Figure 3 Illustration of $BP(.)$ on 2 and 3 dimensional cubes.

and $b' = (2^{d'-1})_2 \oplus b$. Let $\left(L_0^{d'}[\ell_i^d], L_1^{d'}[\ell_i^d]\right)$ denote the (unique) pair of incident d'-dim links, i.e. $L_0^{d'}[\ell_i^d] = (a, a')$ and $L_1^{d'}[\ell_i^d] = (b, b')$. The subscripts 0 and 1 indicate the value at the dth bit position of the links in the pair. Due to symmetry, $N_{d'}(N_{d'}(\ell_i^d)) = \ell_i^d$ and $L_b^{d'}[\ell_i^d] = L_b^{d'}[N_{d'}(\ell_i^d)], b \in \{0, 1\}$.

We formulate the backup path of a d-dim link as a set consisting of one d-dim link and pairs of links. These pairs constitute a joint path, i.e., two paths over the endpoints of detoured d-dim links. We refer to this joint path as a backup path and we always traverse it towards the included d-dim link. However in reality, a packet traverses the two paths in opposite directions, towards and away from the respective d-dim link.

For instance, the backup path of the first 1-dim link (i.e. ℓ_0^1) includes the 1-dim link reached via the incident pair of 2-dim links, and the pair itself (see Figure 3b): $BP(\ell_0^1) = \{L_0^2[\ell_0^1], L_1^2[\ell_0^1], N_2(\ell_0^1)\} = \{\ell_0^2, \ell_1^2, \ell_1^1\}$ (see Figure 3c). For the second 1-dim link we use the same pair, but we have to detour ℓ_0^1 in order to avoid conflict:

$$BP(\ell_1^1) = \{L_1^2[\ell_1^1], L_1^2[\ell_1^1], L_0^3[\ell_0^1], L_1^3[\ell_0^1], N_3(\ell_0^1)\} = \{\ell_0^2, \ell_1^2, \ell_0^3, \ell_1^3, \ell_3^1\}.$$

In general, the backup path of ℓ_i^d begins with the pair $\left(L_0^{d+1}[\ell_i^d], L_1^{d+1}[\ell_i^d]\right)$. If the first d-dim link, i.e. $N_{d+1}(\ell_i^d)$, is conflicting, then one continues by detouring this link via the pair of $(d+2)$-dim links and detours further d-dim links, until one reaches a d-dim link that is not conflicting, then traverses this link. Moreover, the jth detour is performed via the pair of $(d+j)$-dim links. Hence the pairs are traversed in the ascending order of consecutive dimensions. We denote the closure form of $N_d(.)$ w.r.t. this ordering as

$$N^{(j)}(\ell_i^d) := N_{d+j}(N_{d+j-1}(\dots N_{d+1}(\ell_i^d)\dots), 1 \le j < k.$$

We can now describe our backup path scheme formally, we refer to Figure 3c for an example listing all generated backup paths on the 3-dimensional hypercube. For each dimension $d \in [k]$ and every $0 \leq i < 2^{k-1}$, the backup path of ℓ_i^d is

$$\begin{aligned} BP(\ell_i^d) =\Big\{ & L_0^{d+1}[\ell_i^d], L_1^{d+1}[\ell_i^d], \\ & L_0^{d+2}[N_{d+1}(\ell_i^d)], L_1^{d+2}[N_{d+1}(\ell_i^d)], \\ & \ldots, \\ & L_0^{d+r}[N^{(r-1)}(\ell_i^d)], L_1^{d+r}[N^{(r-1)}(\ell_i^d)], \\ & N^{(r)}(\ell_i^d)) = \ell_{i'}^d \Big\}. \end{aligned} \tag{1}$$

The path detours $r-1$ links, where r is the number of link pairs necessary to have, in order to reach the non-conflicting link $\ell_{i'}^d$ with smallest index. Therefore the path length is $2r+1$. We will later argue that $r \leq R := \lceil \log k \rceil$.

Alternatively to the explicit formulation in (1), $\ell_{i'}^d$ can be obtained directly using bitwise operations. Assume $\ell_i^d = (a, b)$ and $\ell_{i'}^d = (a', b')$. By comparing a' to a (b' to b), we can see that only the r bits to the left of dth bit are affected, i.e., the bits $d+1$ to $d+R \pmod k$. For $x \in \{0,1\}^k$ and $s := R - (k-d)$, we define the increment function that determines the successor link as $inc_{s,d}(x) := (x^{>>s} + (2^d)_2^{>>s})^{>>-s}$. Here the $+$ ignores the carry flag out of the leftmost position. Therefore, $a' = inc_{s,d}(a)$ and $b' = inc_{s,d}(b)$. It is clear that the overall computation takes polynomial time.

In the next section, we will state some necessary observations regarding our hypercube construction, which we will employ for the correctness proof of our scheme in Section 3.3.

3.2 Proof Preliminaries

According to our backup path formulation (1), the backup path of a d-dim link passes through a d-dim link reached via links in higher dimensions, which are presented in pairs in (1). The backup path possibly detours some other d-dim links along its way. The pairs and the involved d-dim links together resemble a chain-like structure which facilitates describing some properties in this section. We now describe these structures formally.

▶ **Definition 3.** Given a *sequence of dimensions* $S_d := (d_i)_{i=0}, d_i \in [k] \setminus \{d\}$, a *chain* of d-dim links, starting from $\ell_{i_0}^d$, denoted by $C(\ell_{i_0}^d)$, consists of a subset of d-dim links and pairs of d_i-dim links, $d_i \in S_d$. The pairs form two walks over the endpoints of the contained d-dim links. The two parallel walks jointly *traverse* the chain. We denote the chain by $C_{S_d}(\ell_{i_0}^d) := \{\ldots, \ell_{i_j}^d, (L_0^{d_j}[\ell_{i_j}^d], L_1^{d_j}[\ell_{i_j}^d]), \ell_{i_{j+1}}^d, \ldots\}, j \geq 0, \ell_{i_{j+1}}^d = N_{d_j}(\ell_{i_j}^d)$. Moreover, if $\exists \ell_{i_{j'}} \in C(\ell_{i_0}^d) : \ell_{i_{j'+1}}^d = \ell_{i_0}$, then it is a *closed* chain denoted by C_{S_d}.

We can directly obtain the following property.

▶ **Property 4.** *Starting from any link ℓ_i^d, by traversing a chain $C_{S_d}(\ell_i^d)$, assume we arrive back at the same link. Then it must be the case that S_d contains every dimension an even number of times.*

▶ **Definition 5.** A link $(a, b), a < b$ is traversed in *uphill* direction when it is from a. The opposite is a *downhill* direction.

Based off this definition, we can categorize the traversal directions.

▶ **Property 6.** *Consider a closed chain containing the pair $(L_0^{d'}[\ell_i^d], L_1^{d'}[\ell_i^d])$ traversed between the links ℓ_i^d and $\ell_j^d = N_{d'}(\ell_i^d)$. If $j > i$ then the direction from ℓ_i^d to ℓ_j^d is uphill, otherwise downhill.*

▶ **Property 7.** *By Properties 4 and 6, in a closed chain, the number of traversals in every dimension is even, half of which is in downhill (uphill) direction.*

Intuitively, uphill and downhill traversals cancel each other which consequently turns the joint walks into joint closed walks over the endpoints.

We next study the interaction between chains. Let $S_d, S_{d'}, d' \neq d$ be two sequences of dimensions. We say the chain $C_{S'_d}$ *crosses* the chain C_{S_d} if $\exists P := (L^0_d, L^1_d) \in C_{S_{d'}} : P \cap C_{S_d} \neq \emptyset$. That is, $C_{S'_d}$ traverses a pair of d-dim links, at least one of which belongs to C_{S_d}.

▶ **Definition 8.** A *mixed* chain is the concatenation of multiple chains (over several dimensions) that cross each other consecutively. In other words, a mixed chain consists of chains of links in at least two dimensions. Formally, for given dimensions $d, d', d'' \in [k]$ and sequences S_d and $S_{d'}$, assume the chain $C_{S_{d'}} = \{\dots, \ell^{d'}_x, (L^0_d[\ell^{d'}_x] = \ell^d_y, L^1_d[\ell^{d'}_x]), \dots\}$ crosses $C_{S_d} = \{\dots, \ell^d_y, (L^0_{d''}[\ell^d_y], L^1_{d''}[\ell^d_y]), \dots\}$. We concatenate these chains into a mixed chain as $\{\dots, \ell^{d'}_x, \ell^d_y, (L^0_{d''}[\ell^d_y], L^1_{d''}[\ell^d_y]), \dots\}$.

The observations in the Properties (4), (6), and (7) hold for mixed chains as well. This is because the mentioned properties do not depend on the dimension of the links being chained, but only on dimensions that are actually traversed. However, traversing a chain of d-dim links does not always imply that dimension d is traversed. Consider three chains of d, d', and d''-dim links that cross each other consecutively. E.g., the chain $C_{S_d} = \{\dots, \ell^d_{j_L}, \dots, \ell^d_{j_R}, (L^0_{d''} = \ell^{d''}_{j_R+1}, L^1_{d''}), \dots\}$ that is crossed by the chain $C_{S_{d'}} = \{\dots, \ell^{d'}_{j_L-1}, (L^0_d = \ell^d_{j_L}, L^1_d), \dots\}$ at the link $\ell^d_{i_{j_L}}$. Also, C_{S_d} crosses the chain $C_{S_{d''}} = \{\dots, \ell^{d''}_{i_{j_{R+1}}}, \dots\}$ at the link $\ell^{d''}_{i_{j_{R+1}}}$. We examine whether dimension d is traversed by comparing the dth bit of the last link before the first cross to C_{S_d}, i.e. $\ell^{d'}_{i_{j_L-1}} = (a_0, b_0)$, to the dth bit of the first link after the second cross (by C_{S_d}), i.e. $\ell^{d''}_{i_{j_{R+1}}} = (a_1, b_1)$. Dimension d is traversed if and only if the two bits hold different values. That is, $(a_0 \wedge a_1) \wedge (2^{d-1})_2 = 0$.

A backup path $BP(\ell^d_i) = \{(L^0_{d+1}, L^1_{d+1}), \dots, N^*(\ell^d_i)\}$ can be represented as a chain $C(\ell^d_i) := \{\ell^d_i, \{L^0_{d+1}, L^1_{d+1}\}, N^{(1)}(\ell^d_i), \dots, N^{(*)}(\ell^d_i)\}$. By Definition 2, there is a dependency from ℓ^d_i to every other link $\ell^{d'}_{i'} \in BP(\ell^d_i)$. Let $MC(\ell^d_i, \ell^{d'}_{i'}) \subseteq C(\ell^d_i) \cup \{\ell^{d'}_{i'}\}$ denote the mixed chain up to and including $\ell^{d'}_{i'}$. Consider the set of backup paths of some subset of links $\{\ell^{d_0}_{i_0}, \ell^{d_1}_{i_1}, \dots, \ell^{d_{x-1}}_{i_{x-1}}\}$ that induce a cycle of dependencies. Each dependency corresponds to a mixed chain, concatenating them sequentially, yields the closed mixed chain $\mathcal{MC} := MC(\ell^{d_0}_{i_0}, \ell^{d_1}_{i_1}) \cup MC(\ell^{d_1}_{i_1}, \ell^{d_2}_{i_2}) \cup \dots \cup MC(\ell^{d_{x-1}}_{i_{x-1}}, \ell^{d_0}_{i_0})$. Recall that in $BP(.)$, pairs connecting consecutive same-dimension links are traversed in the ascending order of dimensions. Therefore, the sequence of dimensions traversed by $\mathcal{MC}$ is specified by $(\tilde{d}_i)_{i=0}$, where $\tilde{d}_0 = 0$, and either $\tilde{d}_{j+1} = \tilde{d}_j$ or $\tilde{d}_{j+1} = \tilde{d}_j + 1 \pmod{k}$. From now on, we assume only the closed chains restricted to the sequence of dimensions $\tilde{d}_i$.

3.3 Correctness

In the following, we address the correctness of our backup path scheme, i.e., resilience to up to $k-1$ link failures in k-dimensional hypercubes. To this end, we need one additional result:

▶ **Claim 9.** *In any backup path $p := BP(\ell^d_i)$ at most one pair of links is traversed in uphill direction.*

Proof. If p does not detour any link then the only pair of links, i.e. $(L^0_{d+1}(\ell^d_i)), L^1_{d+1}(\ell^d_i)))$, is traversed either in uphill or downhill direction, which trivially satisfies the claim. If p detours some link ℓ^d_j, then $j < i$ (by construction). By Property 6, the pair of links preceding

ℓ_j^d is traversed in downhill direction. Since p does not detour the last d-dim link, only the last pair (preceding the last link) is possibly traversed in uphill direction. ◀

We can now prove our main result:

▶ **Theorem 10.** *The scheme $BP(.)$ listed in Term (1) is $(k-1)$-resilient.*

Proof. In order to show that the scheme is $(k-1)$-resilient, we argue that any cycle of dependencies consists of at least k links. We first show that for every $d \in [k]$, any cycle of dependencies over d-dim links is of length at least k. The backup path of every link ℓ_i^d uses only one d-dim link $\ell_{i'}^d, i' = i+1 \pmod R$. Hence, the set of d-dim links are dependent sequentially. Therefore, having $R = \lceil \log k \rceil$ is sufficient to ensure any cycle of dependencies induced by d-dim links is of length $2^R \geq k$.

It remains to analyze the dependency cycles that consist of links in multiple dimensions. By Definition 8 and the construction of the $\mathcal{MC}$, such cycles correspond to mixed chains in the k-cube, each having the following properties:

1. Due to the non-descending sequence $\tilde{d}_i$ and by Property 4, $\mathcal{MC}$ traverses the sequence of dimensions $1, \ldots k$ an even number of times, therefore there are at least $2k$ traversals.
2. By Property 7, at least k of the traversals are in uphill direction.
3. By Property 9, a backup path takes at most one uphill. Meaning, each dependency contributes at most one uphill traversal to the mixed chain.

Combining (1), (2), and (3), implies that there must be at least k dependencies in the assumed cycle of dependencies, which concludes our claim. ◀

4 Related Work

Most modern communication networks support some form of resilient routing, and the topic has already received much interest in the literature. There exists much literature on single [16, 47, 65, 68], double [12, 49], and more [15] failure scenarios, the latter being motivated by, e.g., shared risk link groups [57], attacks [61], or simply node failures which affect all incident links [3, 15, 28, 56]. The spectrum of solutions is broad as well, with some solutions providing only heuristic guarantees [12, 49], some schemes exploiting packet-header rewriting [8, 15] (which however is not always supported in existing networks) or packet-duplication [32] (which however comes with overheads). Furthermore, there is also work that aims at quickly optimizing network behavior after link failures have propagated, e.g., by pre-computing how to rescale traffic at ingress routers once these nodes are fault-aware [43]. However, such mechanisms do not provide protection for packets during convergence.

An interesting line of research studies mechanisms which do not require any additional information in the packet header, such as the works by Feigenbaum et al. [17], by Chiesa et al. [10, 11] (establishing an interesting connection to arc-disjoint graph covers), by Elhourani et al. [15], by Stephens et al. [59, 60], by Borokhovich et al. [6], by Pignolet et al. [51] (establishing an interesting connection to distributed computing problems without communication [45]), and by Foerster et al. [23]. However, these solutions do not require failover paths to traverse the nodes of the original path and do not account for the specific properties of the networks considered in this paper. The former is particularly motivated by the advent of (virtualized [18]) middleboxes [9], and is also known as *local protection scheme* in MPLS terminology [55].

Our work is situated in the context of MPLS and Segment Routing (SR) networks where routing is based on stacks and more specifically, the top of the stack label [50]. While the design of resilient routing algorithms has received much attention already in the context of MPLS, see e.g., [31] and [55, 36] and references therein, existing research on SR networks mainly revolves around flow control, traffic engineering and network utilization [5, 63, 13, 42], or network monitoring [4], see the works by Filsfils et al. [21] and Lebrun et al. [14, 38, 41, 40] for a good overview. Optimization problems typically include the minimization of the number of segments required to compute segmented paths [29]. Salsano et al. [54] propose methods to leverage SR in a network without requiring extensions to routing protocols, and Hartert et al. [34] propose a framework to express and implement network requirements in SR. Only little is known today about fast rerouting in SR networks. In [22], it has been shown that existing solutions for SR fast failover, based on TI-LFA [25], do not work in the presence of two or more failures. However, [22] relies on failure-carrying packets, which is undesirable as discussed above and we overcome in the current paper. Finaly, we in this paper considered hypercubes, which have recently been studied for local fast failover algorithms in [11, 24] as well. While for a single link failure, the general approach of François et al. [25] can be used, we are not aware of any approaches that (conceptually) employ Segment Routing for local fast failover in hypercubes for multiple failures.

5 Conclusion and Future Work

This paper studied the design of algorithms for local fast failover in Segment Routing networks, subject to multiple link failures. Our main result is a maximally robust, $(k-1)$-resilient algorithm for k-dimensional hypercubes, which can be computed efficiently.

We see our work as a first step and believe that it opens several promising directions for future research. On the algorithmic side, it would be interesting to extend the study to algorithms for other graph classes, also providing a minimal number of segments or requiring a minimal number of forwarding rules. On the practical side, given that segment routing is ready to be deployed in IPv6 environments, it would be interesting to study experimental evaluations, which can in turn also refine our model. In the following, we provide some first directions.

5.1 Future Work I: Resilient Segment Routing on General Graphs

It will be interesting to study the complexity of fast rerouting on general graphs, and develop (approximation) algorithms accordingly. We conjecture that computing backup path schemes with maximal resiliency is NP-hard on general graphs. In non-polynomial time, a Mixed Integer Program (MIP) formulation can provide an optimal solution for general graphs. The following MIP considers the problem of generating a small number of required segments for the backup paths, and if the desired resiliency cannot be met, at least maximizes the number of protected links. We hope that our MIP formulation can aid the community in developing further backup path schemes, e.g., by using it as a baseline comparison to evaluate the quality of polynomial runtime algorithms for different graph classes beyond the hypercube.

More specifically, the MIP presented next will compute an f-resilient backup path allocation that is optimal in the number of protected links. For completeness purposes, we consider directed graphs $G = (V, E)$. As our MIP is also concerned with the number of labels for each backup path, we provide some additional preliminaries relevant to practical implementations. A backup path in general can be subdivided into path segments, each being a shortest path between its endpoints: such a path segment will only need one label on the stack, when the nodes employ shortest path routing. However, when the network utilizes link weights, some backup paths cannot be represented by node labels [25]: e.g., if a link on the backup path has infinite weight, while all other links have unit weight. For these corner cases, we need to allow single links as items on the label stack, which we denote as *tunnel* links: In the worst case, the whole backup path contains only tunnel links. Should a tunnel link physically fail, the corresponding label will be popped to prevent stuck packets (a failed link cannot be traversed), and the respective backup path will be traversed.

$$\textbf{Maximize} \sum_{\ell \in E} \mathcal{I}_\ell \tag{2}$$

$$\mathcal{SP}_\ell^z = \begin{cases} 1 & \ell \in SP(u,z) \\ 0 & else \end{cases} \qquad \forall \ell = (u,v) \in E, z \in V \tag{3}$$

$$\mathcal{D}_{\ell\ell'}, \mathcal{X}_{\ell\ell'}^v, \mathcal{T}_{\ell\ell'}, \mathcal{W}_\ell^v, \mathcal{I}_\ell, \in \{0,1\} \qquad \forall \ell, \ell' \in E, \forall v \in V \tag{4}$$

$$\mathcal{D}_{\ell\ell} = 0 \qquad \forall \ell \in E \tag{5}$$

$$\sum_{\ell_2=(v,*)} \mathcal{D}_{\ell_1\ell_2} - \sum_{\ell_2=(*,v)} \mathcal{D}_{\ell_1\ell_2} = \begin{cases} \mathcal{I}_{\ell_1} & v = s \\ -\mathcal{I}_{\ell_1} & v = t \\ 0 & \text{else} \end{cases} \qquad \forall \ell_1 = (s,t) \in E, v \in V \tag{6}$$

$$\mathcal{X}_{\ell_1\ell_2}^v \leq \mathcal{SP}_{\ell_2}^v, \sum_{\ell \in E, \ell \ni v} \mathcal{D}_{\ell_1\ell} \qquad \forall \ell_1, \ell_2 \in E, v \in V \tag{7}$$

$$\mathcal{D}_{\ell_1\ell_2} \leq \mathcal{SP}_{\ell_2}^t + \mathcal{T}_{\ell_1\ell_2} + \sum_{v \in V} \mathcal{X}_{\ell_1\ell_2}^v \qquad \forall \ell_1 = (s,t), \ell_2 \in E \tag{8}$$

$$d_{\ell_1\ell_2} \geq 0, d_{\ell_1\ell_1} = 0 \qquad \forall \ell_1, \ell_2 \in E \tag{9}$$

$$d_{\ell_1\ell_3} \leq d_{\ell_1\ell_2} + 1 + (1 - \mathcal{D}_{\ell_2\ell_3}) \times \infty \qquad \forall \ell_1, \ell_2, \ell_3 \in E \tag{10}$$

$$d_{\ell_1\ell_2} + d_{\ell_2\ell_1} \geq f + 1 \qquad \forall \ell_1, \ell_2 \in E \tag{11}$$

$$\mathcal{W}_{\ell_1}^v \geq \mathcal{X}_{\ell_1\ell_2}^v \qquad \forall \ell_1\ell_2 \in E, v \in V \tag{12}$$

$$\sum_{v \in V} \mathcal{W}_\ell^v + \sum_{\ell' \in E} \mathcal{T}_{\ell\ell'} \leq \text{LABELS} \qquad \forall \ell \in E \tag{13}$$

Armed with the above preliminaries, we can now provide a general overview of the MIP. Let $SP(u,z)$ be the shortest path between u and z.[4] For every link $\ell = (s,t)$, we pre-compute constants $\mathcal{SP}_\ell^z$, each indicating whether the shortest path from s to z includes ℓ or not. With respect to the logical flow of the formulation, the MIP first computes a backup path $\mathcal{P}_\ell = \{\ell' \in E \mid \mathcal{D}_{\ell\ell'} = 1\}$ for every link $\ell \in E$. Then, for every link $\ell' \in \mathcal{P}_\ell$ whose shortest path to t does not take the link itself (i.e. $\mathcal{SP}_{\ell'}^t = 0$), the MIP either finds an intermediate node v such that $\mathcal{SP}_{\ell'}^v = 1$, or flags the link as a tunnel link (with $\mathcal{T}_{\ell_1\ell_2}$). As a result, every link of $\mathcal{P}_\ell$ either is a tunnel link or is on the shortest path to a next intermediate node, if not t (i.e. on a segment). This is imposed by the set of constraints (7) and (8). With constraints (9) to (11), we ensure an f-resilient backup path selection. Constraint (11) forbids any cyclic dependency of length $\leq f$. At the end, the MIP restricts the number of segments to the constant LABELS.

Next, we explain each set of constraints and variables more technically.

- (2): maximizing the number of protected links. The failure of any subset of up to f protected links can be tolerated.
- (3): are the pre-computed shortest path trees for all nodes.

[4] Should there be multiple options for shortest paths, we pick them in such a way that each subpath of a shortest path is again a shortest path.

- (4): each variable $\mathcal{D}_{\ell\ell'}$ is set to 1 if ℓ' is designated to the backup path of ℓ ($\mathcal{P}_\ell$), otherwise remains 0. Each variable $\mathcal{X}^v_{\ell\ell'}$ indicates whether 1) the node v is a waypoint on $\mathcal{P}_\ell$ and 2) ℓ' is on the shortest path from the tail of ℓ' to v, hence on the backup path. Similarly, $\mathcal{T}_{\ell\ell'}$ indicates whether ℓ' is a tunnel link on $\mathcal{P}_\ell$. Variables $\mathcal{W}^v_\ell$ is set to 1 when some node v is used as a waypoint for $\mathcal{P}_\ell$. Each variable $\mathcal{I}_\ell$ indicated whether ℓ is protected.
- (6): these constraints enforce the links specified by $\mathcal{D}_{\ell_1 *}$ to form a simple path connecting the endpoints of ℓ_1, not using ℓ_1 (due to (5)).
- (7), (8): a link $\ell_2 = (x, y)$ is allowed to be on the the backup path $\mathcal{P}_{\ell_1}, \ell_1 = (s, t)$ only if
  1. the link ℓ_2 is on the shortest path $SP(x, t)$ i.e. $SP^t_{\ell_2} = 1$;
  2. else, a node $v \in \mathcal{P}_{\ell_1}$ exists s.t. $SP(x, v)$ begins with ℓ_2 (when $\mathcal{X}^v_{\ell_1\ell_2} = 1$ in (7)),
  3. else, the variable $\mathcal{T}_{\ell_1\ell_2}$ is set to 1, which enforces the link ℓ_2 on $\mathcal{P}_{\ell_1}$ as a tunnel link.

  Therefore at least one of the cases must apply to the pair ℓ_1, ℓ_2 in order to have $\mathcal{D}_{\ell_1\ell_2} = 1$ feasible. Cases 2 and 3 correspond to adding new segments. Note that the case 3 can trivially hold for any link which would result in unrestricted number of segments. But latter constraints avoid this in favour of having fewer segments.
- (9),(10),(11): here we formulate the all-pairs shortest path sub-problem on the dependency graph induced by $\mathcal{D}_{**}$. Given a feasible assignment, the value of each d_{xy} is at most the length of the shortest path from x to y. The length of the shortest cycle of dependencies through each dependency arc (ℓ_1, ℓ_2) is constrained by (11).
- (12),(13): the flag $\mathcal{W}^v_\ell$ is set to 1 whenever the node $v \in \mathcal{P}_\ell$ is used as a waypoint for some ℓ' on $\mathcal{P}_\ell$. We restrict the total number of labels (thus, the number of segments) using the constant LABELS.

5.2 Future Work II: Testbeds for Fast Failover in Segment Routing

The most popular testing environment for Segment Routing is *Nanonet* [39], which provides an IPv6 data plane and is conceptually based off *Mininet*[5]. Nanonet allows to easily benchmark Segment Routing in different topologies, all contained in a virtualized enviroment.

To conduct a first feasibility study and evaluate the performance of Segment Routing under different failure scenarios, we deploy the example from Figure 2 as a topology, with an additional source node s connected to v_1, using w as the destination node. Each link has 1 ms delay and bidirectional 10 Mbit/s bandwidth. Without failures, the standard route is s—v_1–(e_1)–w.

If e_1 is unavailable, then v_1 will push v_2 as a segment label (and w will switch to e_2 for the return path), i.e., the packet path is s—v_1—v_2–(e_2)–w. When additionally e_2 is unavailable, then v_1 will push v_1, v_3 as segment labels (with w switching to e_3 for the return path), with the total packet path being s—v_1—v_2—v_1—v_3–(e_3)–w.

We use `iperf3` to generate IPv6 traffic to evaluate the TCP throughput between source and destination nodes, stopping the experiment after 20 seconds, providing ample time for TCP to stabilize. As Nanonet does not support failing links during runtime, we run the experiment three times, first without link failures, then deactivating e_1, and lastly deactivating e_1 and e_2. The results of all three experiments are plotted in Figure 4.

As can be seen, the throughput slightly deteriorates after one link failure, with an additional very small performance hit after the second link failure. We believe that the extent of the slowdown may be related to simulation constraints, as implementing Segment Routing takes additional computational overhead in the virtualized environment, but it would be

[5] `http://mininet.org/`

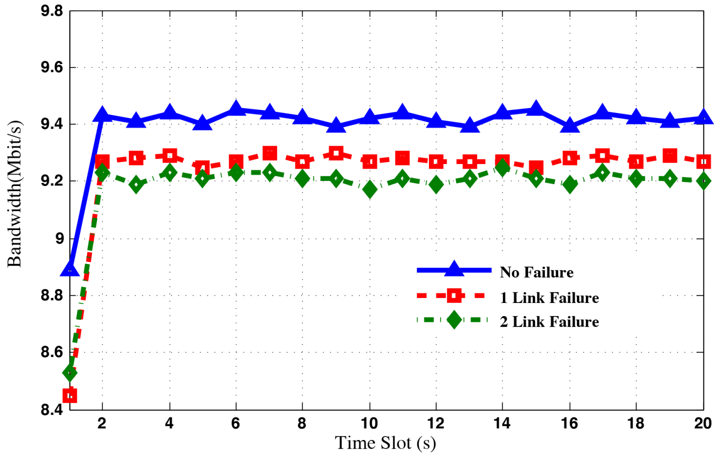


Figure 4 TCP throughput of `iperf3` under 0, 1, and 2 link failures in Nanonet, using an adapted version of the topology and Segment Routing rules from Figure 2.

interesting to investigate the performance impact in a real hardware testbed. Additionally, we believe it would be worthwhile to implement link failures during the simulation runtime in Nanonet, to efficiently estimate the possible performance changes that occur directly after the links went down. We plan to extend our current simulations in these directions.

References

1 Saeed Akhoondian Amiri, Klaus-Tycho Foerster, Riko Jacob, Mahmoud Parham, and Stefan Schmid. Waypoint Routing in Special Networks. In *Proc. IFIP Networking*, 2018.

2 Saeed Akhoondian Amiri, Klaus-Tycho Foerster, Riko Jacob, and Stefan Schmid. Charting the Algorithmic Complexity of Waypoint Routing. *CCR*, 48(1):42–48, 2018. `doi:10.1145/3211852.3211859`.

3 Alia K Atlas and Alex Zinin. Basic specification for IP fast-reroute: loop-free alternates. *IETF RFC 5286*, 2008.

4 François Aubry, David Lebrun, Stefano Vissicchio, Minh Thanh Khong, Yves Deville, and Olivier Bonaventure. Scmon: Leveraging segment routing to improve network monitoring. In *Proc. IEEE INFOCOM*, 2016.

5 Randeep Bhatia, Fang Hao, Murali Kodialam, and TV Lakshman. Optimized network traffic engineering using segment routing. In *IEEE INFOCOM*, 2015.

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

7 Costas Busch, Srikanth Surapaneni, and Srikanta Tirthapura. Analysis of link reversal routing algorithms for mobile ad hoc networks. In *Proc. ACM SPAA*. ACM, 2003. `doi:10.1145/777412.777446`.

8 Marco Canini, Petr Kuznetsov, Dan Levin, and Stefan Schmid. A Distributed and Robust SDN Control Plane for Transactional Network Updates. In *Proc. IEEE INFOCOM*, 2015.

9 B. Carpenter and S. Brim. Middleboxes: Taxonomy and Issues. RFC 3234, RFC Editor, February 2002. URL: `http://www.rfc-editor.org/rfc/rfc3234.txt`.

10 Marco Chiesa, Ilya Nikolaevskiy, Slobodan Mitrovic, Andrei V. Gurtov, Aleksander Madry, Michael Schapira, and Scott Shenker. On the Resiliency of Static Forwarding Tables. *IEEE/ACM Trans. Netw.*, 25(2):1133–1146, 2017. `doi:10.1109/TNET.2016.2619398`.

11 Marco Chiesa, Ilya Nikolaevskiy, Slobodan Mitrovic, Aurojit Panda, Andrei V. Gurtov, Aleksander Madry, Michael Schapira, and Scott Shenker. The quest for resilient (static) forwarding tables. In *Proc. IEEE INFOCOM*, 2016. doi:10.1109/INFOCOM.2016.7524552.

12 Hongsik Choi, Suresh Subramaniam, and Hyeong-Ah Choi. On double-link failure recovery in WDM optical networks. In *Proc. IEEE INFOCOM*, 2002. doi:10.1109/INFCOM.2002.1019327.

13 Luca Davoli, Luca Veltri, Pier Luigi Ventre, Giuseppe Siracusano, and Stefano Salsano. Traffic engineering with segment routing: SDN-based architectural design and open source implementation. In *Proc. EWSDN*, 2015.

14 Fabien Duchêne, David Lebrun, and Olivier Bonaventure. SRv6Pipes: enabling in-network bytestream functions. In *Proc. IFIP Networking*, 2018.

15 Theodore Elhourani, Abishek Gopalan, and Srinivasan Ramasubramanian. IP fast rerouting for multi-link failures. *IEEE/ACM Trans. Netw*, 24(5):3014–3025, 2016.

16 Gábor Enyedi, Gábor Rétvári, and Tibor Cinkler. A novel loop-free IP fast reroute algorithm. In *Meeting of the European Network of Universities and Companies in Information and Communication Engineering*, pages 111–119. Springer, 2007.

17 Joan Feigenbaum et al. BA: On the resilience of routing tables. In *Proc. ACM PODC*, 2012.

18 ETSI. Network Functions Virtualisation. In *White Paper*, 2013.

19 Nick Feamster, Jennifer Rexford, and Ellen Zegura. The road to SDN: an intellectual history of programmable networks. *ACM SIGCOMM CCR*, 44(2):87–98, 2014.

20 Clarence Filsfils, Pierre François, Stefano Previdi, Bruno Decraene, Stephane Litkowski, Martin Horneffer, Igor Milojevic, Rob Shakir, Saku Ytti, Wim Henderickx, Jeff Tantsura, Sriganesh Kini, and Edward Crabbe. Segment Routing Architecture. In *Segment Routing Use Cases, IETF Internet-Draft*, 2014.

21 Clarence Filsfils, Nagendra Kumar Nainar, Carlos Pignataro, Juan Camilo Cardona, and Pierre Francois. The segment routing architecture. In *IEEE GLOBECOM*, 2015.

22 Klaus-Tycho Foerster, Mahmoud Parham, Marco Chiesa, and Stefan Schmid. TI-MFA: keep calm and reroute segments fast. In *Global Internet Symposium (GI)*, 2018.

23 Klaus-Tycho Foerster, Yvonne-Anne Pignolet, Stefan Schmid, and Gilles Tredan. Local Fast Failover Routing With Low Stretch. *ACM SIGCOMM CCR*, 1:35–41, January 2018.

24 Klaus-Tycho Foerster, Yvonne Anne Pignolet, Stefan Schmid, and Gilles Trédan. Local Fast Failover Routing With Low Stretch. *CCR*, 48(1):35–41, 2018. doi:10.1145/3211852.3211858.

25 Pierre François, Clarence Filsfils, Ahmed Bashandy, and Bruno Decraene. Topology Independent Fast Reroute using Segment Routing. Internet-Draft draft-francois-segment-routing-ti-lfa-00, Internet Engineering Task Force, November 2013. URL: https://datatracker.ietf.org/doc/html/draft-francois-segment-routing-ti-lfa-00.

26 Pierre François, Clarence Filsfils, John Evans, and Olivier Bonaventure. Achieving subsecond IGP convergence in large IP networks. *CCR*, 35(3):35–44, 2005. doi:10.1145/1070873.1070877.

27 Eli M. Gafni and Dimitri P. Bertsekas. Distributed Algorithms for Generating Loop-Free Routes in Networks with Frequently Changing Topology. *IEEE Transactions on Communications*, 29(1):11–18, January 1981.

28 Phillipa Gill, Navendu Jain, and Nachiappan Nagappan. Understanding network failures in data centers: measurement, analysis, and implications. In *ACM SIGCOMM CCR*, volume 41, pages 350–361, 2011.

29 Alessio Giorgetti, Piero Castoldi, Filippo Cugini, Jeroen Nijhof, Francesco Lazzeri, and Gianmarco Bruno. Path encoding in segment routing. In *Proc. IEEE GLOBECOM*, 2015.

30 Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu. BCube: a high performance, server-centric network architecture for modular data centers. In *Proc. ACM SIGCOMM*, 2009.

31 Anupam Gupta, Amit Kumar, and Rajeev Rastogi. Traveling with a pez dispenser (or, routing issues in mpls). *SIAM Journal on Computing*, 34(2):453–474, 2005.

32 Prashanth Hande, Mung Chiang, Robert Calderbank, and Sundeep Rangan. Network pricing and rate allocation with content-provider participation. In *Proc. IEEE INFOCOM*, 2010.

33 Ed Harrison, Adrian Farrel, and Ben Miller. Protection and restoration in MPLS networks. *Data Connection White Paper*, 2001.

34 Renaud Hartert, Stefano Vissicchio, Pierre Schaus, Olivier Bonaventure, Clarence Filsfils, Thomas Telkamp, and Pierre Francois. A declarative and expressive approach to control forwarding paths in carrier-grade networks. In *ACM SIGCOMM CCR*, volume 45, pages 15–28, 2015.

35 Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. Achieving high utilization with software-driven WAN. In *Proc. ACM SIGCOMM*, 2013.

36 Jesper Stenbjerg Jensen, Troels Beck Krogh, Jonas Sand Madsen, Stefan Schmid, Jiri Srba, and Marc Tom Thorgersen. P-Rex: Fast Verification of MPLS Networks with Multiple Link Failures. In *Proc. ACM CoNEXT*, 2018.

37 Karthik Lakshminarayanan, Matthew Caesar, Murali Rangan, Tom Anderson, Scott Shenker, and Ion Stoica. Achieving convergence-free routing using failure-carrying packets. In *Proceedings of the ACM SIGCOMM 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Kyoto, Japan, August 27-31, 2007*, pages 241–252. ACM, 2007. `doi:10.1145/1282380.1282408`.

38 David Lebrun. *Reaping the Benefits of IPv6 Segment Routing.* PhD thesis, UCLouvain / ICTEAM / EPL, October 2017.

39 David Lebrun. Virtual networks testing framework (Nanonet). `https://github.com/segment-routing/nanonet`, February 2017.

40 David Lebrun and Olivier Bonaventure. Implementing IPv6 Segment Routing in the Linux Kernel. In *Proc. ACM ANRW*, 2017.

41 David Lebrun, Mathieu Jadin, François Clad, Clarence Filsfils, and Olivier Bonaventure. Software Resolved Networks: Rethinking Enterprise Networks with IPv6 Segment Routing. In *Proc. ACM SOSR*, 2018.

42 Ming-Chieh Lee and Jang-Ping Sheu. An Efficient Routing Algorithm Based on Segment Routing in Software-defined Networking. *Comput. Netw.*, 103(C):44–55, July 2016.

43 Hongqiang Harry Liu, Srikanth Kandula, Ratul Mahajan, Ming Zhang, and David Gelernter. Traffic engineering with forward fault correction. In *Proc. ACM SIGCOMM*, 2014.

44 Junda Liu, Aurojit Panda, Ankit Singla, Brighten Godfrey, Michael Schapira, and Scott Shenker. Ensuring Connectivity via Data Plane Mechanisms. In *Proc. USENIX NSDI*, 2013.

45 Grzegorz Malewicz, Alexander Russell, and Alexander A. Shvartsman. Distributed scheduling for disconnected cooperation. *Distributed Computing*, 18(6):409–420, 2005.

46 Athina Markopoulou, Gianluca Iannaccone, Supratik Bhattacharyya, Chen-Nee Chuah, Yashar Ganjali, and Christophe Diot. Characterization of failures in an operational IP backbone network. *IEEE/ACM Trans. Netw.*, 16(4):749–762, 2008. `doi:10.1145/1453698.1453699`.

47 Srihari Nelakuditi, Sanghwan Lee, Yinzhe Yu, Zhi-Li Zhang, and Chen-Nee Chuah. Fast local rerouting for handling transient link failures. *IEEE/ACM Trans. Netw*, 15(2):359–372, 2007.

48 Mohammad Noormohammadpour and Cauligi S Raghavendra. Datacenter Traffic Control: Understanding Techniques and Tradeoffs. *IEEE Communications Surveys & Tutorials*, 20(2):1492–1525, 2017.

49 Eunseuk Oh, Hongsik Choi, and Jong-Seok Kim. Double-Link Failure Recovery in WDM Optical Torus Networks. In *Information Networking, Networking Technologies for Broadband and Mobile Networks, International Conference ICOIN*, 2004.

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

51 Yvonne-Anne Pignolet, Stefan Schmid, and Gilles Tredan. Load-Optimal Local Fast Rerouting for Dependable Networks. In *Proc. IEEE/IFIP DSN*, 2017.

52 Y. Saad and M. H. Schultz. Topological properties of hypercubes. *IEEE Transactions on Computers*, 37(7):867–872, July 1988.

53 Yousef Saad and Martin H. Schultz. Data Communication in Hypercubes. *J. Parallel Distrib. Comput.*, 6(1):115–135, 1989. `doi:10.1016/0743-7315(89)90045-2`.

54 Stefano Salsano, Luca Veltri, Luca Davoli, Pier Luigi Ventre, and Giuseppe Siracusano. PMSR—Poor Man's Segment Routing, a minimalistic approach to Segment Routing and a Traffic Engineering use case. In *Proc. IEEE/IFIP NOMS*, 2016.

55 Stefan Schmid and Jiri Srba. Polynomial-Time What-If Analysis for Prefix-Manipulating MPLS Networks. In *Proc. IEEE INFOCOM*, 2018.

56 Aman Shaikh, Chris Isett, Albert Greenberg, Matthew Roughan, and Joel Gottlieb. A case study of OSPF behavior in a large enterprise network. In *Proc. ACM SIGCOMM Workshop on Internet Measurment*, 2002.

57 Lu Shen, Xi Yang, and Byrav 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.

58 Abhinav Kumar Singh, Ravindra Singh, and Bikash C. Pal. Stability Analysis of Networked Control in Smart Grids. *IEEE Trans. Smart Grid*, 6(1):381–390, 2015. `doi:10.1109/TSG.2014.2314494`.

59 Brent Stephens, Alan L. Cox, and Scott Rixner. Plinko: Building Provably Resilient Forwarding Tables. In *Proc. 12th ACM HotNets*, 2013.

60 Brent Stephens, Alan L Cox, and Scott Rixner. Scalable Multi-Failure Fast Failover via Forwarding Table Compression. *SOSR. ACM*, 2016.

61 János Tapolcai, Balázs Vass, Zalán Heszberger, József Bıró, David Hay, Fernando A Kuipers, and Lajos Rónyai. A Tractable Stochastic Model of Correlated Link Failures Caused by Disasters. In *Proc. IEEE INFOCOM*, 2018.

62 Frederic Trate. Bringing Segment Routing and IPv6 together, August 2016. URL: `https://blogs.cisco.com/sp/bringing-segment-routing-and-ipv6-together`.

63 George Trimponias, Yan Xiao, Hong Xu, Xiaorui Wu, and Yanhui Geng. On Traffic Engineering with Segment Routing in SDN based WANs. *arXiv preprint arXiv:1703.05907*, 2017.

64 Emmanouel A. Varvarigos and Dimitri P. Bertsekas. Performance of hypercube routing schemes with or without buffering. *IEEE/ACM Trans. Netw.*, 2(3):299–311, 1994. `doi:10.1109/90.311628`.

65 Junling Wang and Srihari Nelakuditi. IP fast reroute with failure inferencing. In *Proc. SIGCOMM Workshop on Internet Network Management*, pages 268–273, 2007.

66 Ye Wang, Hao Wang, Ajay Mahimkar, Richard Alimi, Yin Zhang, Lili Qiu, and Yang Richard Yang. R3: resilient routing reconfiguration. In *Proc. ACM SIGCOMM*, 2010.

67 Christo Wilson, Hitesh Ballani, Thomas Karagiannis, and Antony I. T. Rowstron. Better never than late: meeting deadlines in datacenter networks. In *Proc. ACM SIGCOMM*, 2011.

68 Baobao Zhang, Jianping Wu, and Jun Bi. RPFP: IP fast reroute with providing complete protection and without using tunnels. In *Proc. IWQoS*, 2013.