TI-MFA: Keep Calm and Reroute Segments Fast

Klaus-Tycho Foerster[1] Mahmoud Parham[1] Marco Chiesa[2] Stefan Schmid[1]
[1] University of Vienna, Austria [2] KTH Royal Institute of Technology, Sweden

***Abstract*—Segment Routing (SR) promises to provide scalable and fine-grained traffic engineering. However, little is known today on how to implement resilient routing in SR, i.e., routes which tolerate one or even multiple failures. This paper initiates the theoretical study of static fast failover mechanisms which do not depend on reconvergence and hence support a very fast reaction to failures. We introduce formal models and identify fundamental tradeoffs on what can and cannot be achieved in terms of static resilient routing. In particular, we identify an inherent price in terms of performance if routing paths need to be resilient, *even in the absence of failures*. Our main contribution is a first algorithm which is resilient even to multiple failures and which comes with provable resiliency and performance guarantees. We complement our formal analysis with simulations on real topologies, which show the benefits of our approach over existing algorithms.**

I. Introduction

Segment Routing (SR) [1], [2], [3] emerged to address operational issues of MPLS-based traffic engineering solutions. SR provides a fine-grained flow management and traffic steering, allowing to leverage a potentially much higher path diversity than shortest path control planes such as OSPF [4]. At the same time, SR overcomes the scalability issues of MPLS networks as it does not require any resource reservations and states on all the routers part of the virtual circuit. SR is also attractive as, by relying on existing routing protocols, it is easier to deploy than OpenFlow-based solutions.

While the benefits of segment routing in terms of traffic engineering and scalability have been articulated well in the literature, less is known today about how to provide resilience to link and node failures. In particular, we in this paper are interested in *Fast Reroute (FRR)* approaches, which handle failures quickly and without invoking the control plane or having to wait for shortest path reconvergence, using statically predefined failover rules. We are especially interested in highly robust FRR schemes which even tolerate *multiple* link failures as they are more likely to arise in larger networks (e.g., data center [5], backbone [6] or enterprise [7] networks) as well as due to shared risk link groups, and for these reasons also constitute a main concern of network carriers [8].

A. SR in a Nutshell

In a nutshell, segment routing is reminiscent of MPLS, in that it also relies on *label stacks* in the packet header: the label at the top of the stack defines the next *node* or *link*. However, unlike MPLS:

1) Segment routing leverages a *source routing* paradigm, in which entire paths are pushed at the network edge (resp. tunnel ingress). Thus, unlike in MPLS networks, there is hence no need for lookup tables to replace (e.g., swap) labels at each traversing node.
2) The next node (the node on the top of the stack) does not have to be directly adjacent to the current node. In this case, to transport packets to the next node, segment routing relies on *shortest path routing* (a "segment"). If the next element is a link, that link has to be directly adjacent to the current node.

The definition of intermediate points in the label stack allows to increase the path diversity beyond shortest paths. However, intermediate points can also be used for Fast Rerouting (FRR): if a link along the current shortest path from s to t failed, say at some node u, an alternative intermediate destination (or waypoint) w can be defined to reroute around the failure. In this case, node v receives a label stack "t" (only the destination node t is on the stack), pushes the intermediate destination w (a waypoint), and hence sends out a packet with label stack "wt". Once the packet reaches node w, this node is popped from the stack and the packet (with stack "t") routed to t. If additional failures occur, additional intermediate destinations can be defined recursively.

B. The Challenge

The key challenge in the design of fast rerouting schemes is that failover rules need to be proactively and statically *pre*-configured: there is no time to recompute paths or communicate information about failures up- or downstream. This also implies that FRR rules can only depend on *local* knowledge and in particular, are *oblivious* to possibly additional failures occurring downstream. Without such global knowledge about failures, however, if no provisions are made by the algorithm defining the failover rules, the resulting routes may become inconsistent (e.g., form a loop). For example, if due to a failed link e, a node v reroutes the traffic from s to t along an intermediate segment to w, it can be undesirable that an additional failure of a link e' in the segment from v to w will steer the traffic back to e again. Moreover, besides ensuring connectivity even under multiple link failures, the FRR mechanism should be *efficient*, and in particular, only require a small number of additional failover segments and a small amount of extra header space.

C. Our Contributions and Paper Organization

This paper initiates the systematic study of static fast failover in segment routing. We present a number of insights on what can and cannot be achieved in fast failover in SR, as well as on potential tradeoffs. In particular:

§II We show that existing solutions for SR fast failover, based on TI-LFA [9], do not work in the presence of two or more failures. Furthermore, any TI-LFA extension for k failures must provision for $2k + 1$ stack segments.

§III We show that at least in principle, existing fast rerouting techniques based on disjoint arborescences known from the literature considering non-SR networks [10], [11], [12], can be emulated in segment routing as well. However, this emulation comes at a high overhead.

§IV We identify an interesting and fundamental tradeoff between the efficiency and robustness of failover. In particular, we show that any failover scheme for SR, without extensions, which tolerates at least *two* failures, can be forced to use very costly routes even in the presence of a *single* failure.

§V We present an efficient algorithm which preserves reachability even under multiple link failures. Our algorithm, called TI-MFA, is an extension of TI-LFA, and comes with provable correctness guarantees. It is also evaluated in simulations using Rocketfuel topologies; the latter demonstrate TI-MFA's benefits over existing approaches. We furthermore study the effect of *flushing* the label stack in case of a failure, i.e., removing intermediate destinations to optimize the following failover path. Maybe surprisingly, the failure rate of TI-LFA nearly doubles in this setting in our simulations.

In the remainder of §I, we discuss the novelty and limitations of our contributions, along with further related work. We lastly conclude in §VI, also outlining possibilities for future work.

D. Novelty

Fast rerouting in SR differs from fast rerouting in other contexts (such as MPLS and OpenFlow) [11], [12], [13], [14], [15], [16], [17] in that a segment relies on shortest path routing and is subject to traditional control plane mechanisms such as OSPF. As we will show in this paper, the constraint that segment paths need to be shortest makes the underlying algorithmic problem very different. Having that said, and as we also show that in this paper, one can in principle *enforce* certain failover routes and emulate mechanisms such as arborescence routing [11]. This however does not only go against the fundamental principles of SR, but also comes at a higher overhead (in terms of the number of segments that need to be pushed). Finally, to the best of our knowledge, TI-LFA [9] is the only existing proposal to provide node and link protection in SR. In particular, Francois et al.'s approach builds upon IP-FRR concepts such as *Remote LFAs (RLFA)* and remote *LFAs with Directed Forwarding (DLFA)*.

However, as we will show in this paper, these approaches fail in the presence of more than link failure. We thus rely on a concept from [18], where previously hit link failures are inserted into the packet header. The conceptual idea is that then a packet cannot hit the same failed link twice, we provide more details of their work in Section V-A. The concepts in [18] differ from our ideas in the sense that we employ segment routing, whereas [18] either pushes the next hops one at a time or inserts complete paths into the packet header.

Finally, we note that our model is very different from models which invoke the control plane, wait for reconvergence or allow to signal failures upstream: in this case, improved failover schemes can be computed, e.g., using a linear programming model and also accounting for the restoration of bandwidth [19]; however, such approaches come with a communication overhead and delay.

E. Limitations

As our paper is a first exploration of the *fast* RR paradigm for segment routing beyond a single failure, we restrict our model for now to not include *later*, relatively *slow* convergence effects. This allows us take an unrestricted view on the immediate effects of link failures. In a practical setting, it can also be modeled by turning off reconvergence protocols, as our algorithms guarantee delivery if the network remains connected.

F. Other Related Work

There exists much interesting literature on the architectures and use cases for segment routing, and we refer the reader to the IETF documents [2], [3] as well as the works by Filsfils et al. [20] and David Lebrun [1] for a good overview. Existing algorithmic work on SR mostly revolves around flow control, traffic engineering and network utilization [21], [22], [23], [24], but other use cases are considered such as network monitoring [25]. Optimization problems typically include the minimization of the number of segments required to compute segmented paths [26]. Salsano et al. [27] propose methods to leverage SR in a network without requiring extensions to routing protocols, and Hartert et al. [28] propose a framework to express and implement network requirements in SR.

II. Limitation of Existing Approaches

In order to acquaint ourselves with the problem and in order to show the challenges and limitations of existing approaches, we first revisit the TI-LFA approach [9].

Here and throughout this paper, we will only consider the commonly studied case of symmetric networks. In other words, if P is the shortest path from v to w, then traversing P in the reverse direction is also the shortest path from w to v. Furthermore, we assume all link weights to be positive and that subpaths of a shortest path are shortest paths as well.

This symmetry allows TI-LFA to adequately push segments when a packet hits a failed link $e = (v, w)$ at node v: Let v_e-space be the set of all nodes reachable from v via shortest paths in G, where the link e will not be used, i.e., all nodes where the current routing will not hit the failed link. We can define t_e-space analogously for the destination t.[1]

The symmetry property now yields that the v_e- and the t_e-space overlap or at least have adjacent nodes, if e does not disconnect the graph.[2] For a proof intuition, imagine some node u would be contained in neither space: 1) if the shortest paths to u from v and t traverse e in different directions, then one of them could take a shortcut and not use e, and 2) if both shortest paths traverse e in the same direction, then the shortest path from v to t would not use e.

[1]Usually, the terms P- and Q-space are used in the literature, but we introduce node/link-specific notation here for ease of reference.

[2]For an in-depth tutorial, we refer to the topic *Topology Independent LFA (TI-LFA)* at http://www.segment-routing.net/tutorials/.

Hence, v can push pre-computed segments on top of the label stack: 1) if the v_e- and the t_e-space overlap, then a joint node that minimizes the routing distance, and 2) in case of mere adjacency, a border node along with a label of a link that connects both spaces, again minimizing total distance. As such, pushing two labels always suffices for a single failure.

A. TI-LFA loops indefinitely for two link failures

For a simple example why TI-LFA is no longer guaranteed to be correct, we refer to Fig. 1. There are three paths to the destination t, a left one from v_ℓ via e_ℓ, a middle one from v_m via e_m, and a right one from v_r via e_r. Assume e_m fails, leading w.l.o.g. to v_m pushing the label for v_r on the stack. If now e_r fails as well, then v_r pushes the label for v_m on the stack, i.e., the packet loops indefinitely between v_m and v_r, even though there is still a path via e_ℓ: the problem is that TI-LFA does not differentiate regarding where the packet is coming from, which we will tackle in Section III with respect to the incoming port and in Section V by storing the last hit failure(s) in the packet header.

Observation 1. *Even if the network is still connected after two link failures, TI-LFA can loop indefinitely.*

In order to guarantee that the packet still reaches its destination, we need to implement a local (pre-computed) mechanism that enforces that the third path via e_ℓ will be eventually taken, via segment routing.

However, this requires an increase in the number of labels pushed, for which we extend Fig. 1: for the case of two failures, replace the link between v_m and v_r with the construction shown in Fig. 2, analogously for the link between v_ℓ and v_m. Then, four segments are needed to reach t from v_r when the links e_r, e_m are failed, already in the case when v_r is aware that both these links are unavailable. With less segments, the packet can't circumvent the failed links. This idea can be extended to k link failures, each requiring two segments, by extending the network from Fig. 1 appropriately.

Observation 2. *Even if all k link failures in the network are known, at least $2k$ further segments must be provisioned s.t. the packet can reach the destination without further additions to the label stack on-route.*

We quickly summarize our findings so far: 1) TI-LFA can loop already with two link failures, as knowledge of prior actions is not used/available, 2) unless further segments are to be pushed on-route, a label stack proportional to the link failure number is required.

In the following, we now tackle these two observations. In Section III we use the incoming port of the packet (the in-port) as "prior knowledge" and perform ongoing additions to the label stack, by emulating known fast local reroute techniques. While this seems efficient at first, this approach suffers from a high overhead in the path length and essentially ignores the benefits of segment routing. Furthermore, we are not aware of *local* fast reroute techniques that can guarantee reachability beyond a single failure. As thus, in Section V, we take the approach of storing past link failures in the packet header

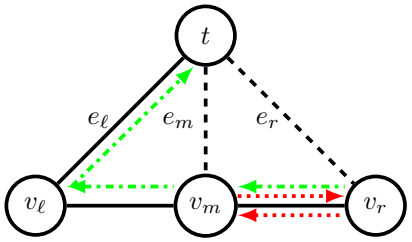

Fig. 1. Illustration of a network where TI-LFA loops permanently with two link failures. Consider a packet at node v_m headed for t, and fail the links e_m, e_r, both drawn dashed. We can assume w.l.o.g. that TI-LFA picks the backup path via e_r to reach t, as e_m is failed. However, when the the node v_r is reached, the link e_r cannot be used, and TI-LFA will send the packet back to v_m, to route to t via e_m. As thus, TI-LFA will loop permanently, drawn in dotted red. Our proposed algorithm TI-MFA will pick the dash-dotted green path when reaching v_r after being rerouted from the initial node v_m, successfully reaching the destination t.

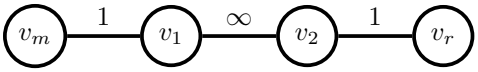

Fig. 2. Construction to increase the minimum number of pushed labels in Fig. 1. To this end, we replace the link between v_m and v_r with a path of weights $1, \infty, 1$, the remaining network from Fig. 1 is omitted here for ease of viewing. When segments are pushed at v_r to reach v_m, without going across the failed link e_r, one segment does not suffice: when being at v_2, the shortest path to v_m is via the failed link e_r (not drawn here). Hence, a second segment is needed, the link between v_2 and v_1. When an analogous construction is applied to the link between v_ℓ and v_m, two further segments are needed to circumvent e_m. Hence, to reach t from v_r without pushing further segments along the way, v_r needs to push four segments.

and push more segments at once, which leads to provable reachability of the destination, as long as the network remains connected under link failures. Our simulations show that for two link failures and flushing, we have an extra label stack size of at most 4 ($+t$), which matches Observation 2 for $k = 2$.

III. A Robust But Inefficient Solution

Literature on static resilient routing for non-SR networks provides powerful techniques to preserve connectivity even under many link failures [10], [11], [12], [29] . In the following, we show that such an approach can be employed also with SR.

The deterministic techniques of Chiesa et al. in [10], [11], [12] rely on building pre-computed routing rules that match on 1) the destination, 2) incident failures, and (unlike classical segment routing) 3) the incoming port of the packet. These works consider k-link-connected networks and aim at reaching resiliency up to $k-1$ failures[3]. Notwithstanding, they do not guarantee reachability of the destination t if the only invariant is that the network remains connected.

We can emulate these techniques if we incorporate a match on the incoming port or on the top element of the stack before popping it. However, the main obstacle is that the techniques of Chiesa et al. do not route along shortest paths, but rather enforce single-hop routing mechanisms, guiding the packet to the destination one step at a time, possibly along long detours in comparison to the global shortest paths.[4]

[3]At least a resiliency of $\lfloor k/2 \rfloor$ is guaranteed, with various graph families also reaching $k-1$, we refer to [10], [11], [12] for details.

[4]The main obstacle to achieving short paths in this context is that manually adding short failover paths can heavily impact resiliency, requiring specialized schemes, such as in [30] for hypercubes, torus, and Clos topologies.

We overcome this obstacle by putting ("forcing") a link on top of the label stack each time the packet arrives at a node.

Now, we can install rules on the switches emulating the techniques by Chiesa et al.: 1) at most one extra label will be on the top of the destination (a link), which is popped when reaching the next node, leaving the destination on top of the label stack to match, 2) incident failures may inherently be matched upon in SR, and 3) the in-port as described above.

However, this emulation approach comes with a high overhead and essentially ignores the main features of SR networks by forcing each link along the route individually.

IV. The Price of Resiliency

We showed that robustness can be achieved using segment routing, but that our emulation approach removes many inherent efficiency aspects. Unfortunately, as we show in the following, there is an inherence price of local resiliency.

Consider the example in Fig. 1, but assign the following link weights: 1 for e_m and e_r, ∞ for e_ℓ.

If, similar to TI-LFA, we protect only against one link failure amongst e_ℓ, e_m, e_r towards the destination t from v_r, v_m, v_ℓ, such as TI-LFA, the failover path can always be chosen to have a short length, respectively small link weights.

However, as soon as we protect against two link failures, this resiliency comes at a price. As the label-pushing at the second failure only takes the 1) incident failure and 2) the destination t[5] into account, the node is unaware of the first failure – and rerouting decisions are only performed when a failure is hit. Hence, either the failover for e_m or e_r must reroute via the expensive e_ℓ, or risk looping indefinitely, similar to the example in Section II-A. Note that the example from Fig. 1 can easily be extended to a 3-link-connected graph.

This highlights the price of resiliency against two failures in local schemes: even if only one failure occurs, an inefficient failover path has to be taken for some network topologies.

V. Efficient Resilient Segment Routing

In order to overcome the identified tradeoff, we make the following observation: while static failover schemes are inherently oblivious to failures downstream of the path, at least we could remember the already encountered failures.

Note that TI-LFA implicitly assume knowledge of all failures: as the rerouting decision is performed at a node incident to the single failure, all other links are assumed to be available. Hence, if there is still a path to the destination, packets will reach it. Nonetheless, as we showed before, if the rerouting node is not aware of the second failure, TI-LFA loops indefinitely.

As in [18] we thus propose to store already hit failures in the packet header, for two reasons:

1) The header space usage scales linearly with the number of hit failures, where a single failure uses similar space as pushing an intermediate destination on the label stack.
2) Rerouting via intermediate segments is performed locally, based of pre-computed table entries.

[5]As v_r will be popped from the label stack when reaching v_r. If the label v_r would also be taken into consideration, we can use a construction along the lines of Fig. 2 to ensure that the label stack at v_r only consists of t.

We note that all described computations can be performed ahead of time and stored in a static routing table, whose size scales with the number of failure combinations.

More specifically, our so-called *Topology Independent Multi-Failure Alternate* (*TI-MFA*) algorithm works as follows, described from the viewpoint of the node v where the packet hits another failed link:

1) Flush the label stack except for the destination t.
2) Based on all link failures stored in the packet header, determine the shortest path P to the destination t in the remaining network G'.
3) Add segments to the label stack of the packet as follows:
   - Index the nodes on P as $v = v_1, v_2, \ldots, v_x = t$. Compute the node v_i on P with the highest index s.t. the shortest path from v is identical in G' (with failures) and G (without failures) and set it as the top of the label stack. If this node is v, push the link $(v_1, v_2 = v_i)$ as the top of the label stack. For the second item on the label stack, start over with v_i as the starting node, etc., until $v_i = t$.

We next show TI-MFA to be correct in theory and efficient in practice, only using small stack sizes under two link failures.

A. Formal Correctness Analysis

We begin by first going a step back to explain the approach of Lakshminarayanan et al. [18], which comes in two flavors called FCP (Failure-Carrying Packets) and source-routing FCP:

- In FCP, a node decides on the next hop by considering the failures the packet hit so far and the packet destination t. Delivery in a connected network is guaranteed by the fact that a packet will not hit a link failure twice: else, an intermediate node would have ignored that this link failure was inserted into the packet header. In our context, FCP thus resembles the hop-by-hop forwarding in Section III, with the advantage that FCP can aim for short paths and stronger reachability, but on the other hand, the emulation idea in Section III did not add failures to the packet header.
- Source-routing FCP behaves analogously, but rather inserts the whole path to the destination into the packet header, consistent with the view based on the link failures stored in the packet header. Should the packet hit a new failure, it is added to the header, the old path is removed (as in our case), and a new path is inserted into the header.

The main difference between TI-MFA and the work of Lakshminarayanan et al. [18] is that in segment routing, only few intermediate waypoints are provided, whereas FCP implicitly provides the whole path: either on a hop-by-hop basis or by inserting the path into the packet header.

We now show the correctness of TI-MFA:

Theorem 1.
Let G be a network where k links fail, s.t. the remaining graph G' is connected. TI-MFA routes successfully to the destination.

Proof. Our proof will be via nested induction arguments over the number of failures hit. Clearly, if no failures are hit, the destination is reached. Similarly, if only one failure is hit at a node v, TI-MFA behaves identical to TI-LFA: the packet is

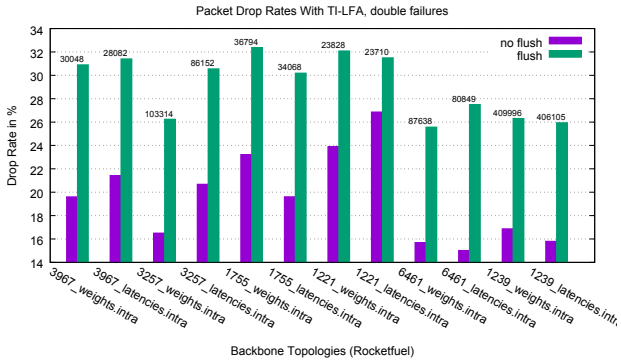

Fig. 3. Failure rates of TI-LFA on ASNs [31], using the provided weights or latencies. The numbers on top of the bars represent the total number of experiments per topology for each algorithm. Only cases where two link failures are encountered are considered. TI-MFA never failed in our simulations.

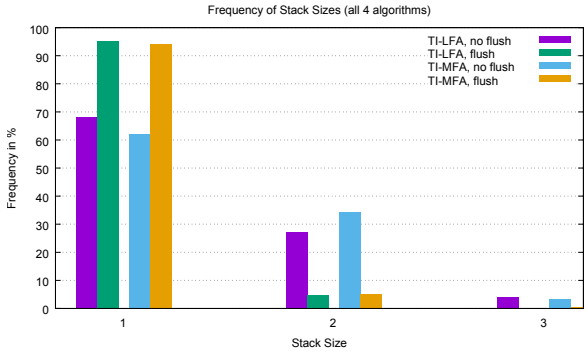

Fig. 4. Frequency of maximum stack sizes of $1, 2, 3$ $(+t)$ in the individual experiments. The (small) frequencies beyond $3+1$ are reported in the text.

routed along a failure-free backup path. Note that due to the pushed segments, the shortest-path segment routing is identical in G (without failures) and G' (with failures) from v.

We now assume the packet already hit $x-1$ failures and hits failure x on its path. We distinguish two cases:

- $x = k$: Then, due to construction (Item 3), the shortest-path segment routing is again identical in G' and G.
- $x > k$: Assume some link failure $x+1$ is hit next (the first x failures can no longer be hit, due to construction), else we will reach the destination. Then, we invoke our inductive construction again: as an invariant, already hit failures can no longer be hit, i.e., eventually we will either have hit all failures or reach the destination. However, once all failures are hit, this implies that the destination will be reached via the path P.

Note that the correctness of the above arguments relies on the assumption that the network G' is connected. □

We would like to note that for two link failures, we can show that the segment stack does not need to be flushed s.t. only the destination remains[6]: let P_1 be the path the packet takes as a backup, after hitting the link failure e_1, if no second link failure is hit. To hit a second failure e_2, the link e_2 must be on P_1. If the label stack is then empty except for t, the destination will be reached by the above proof arguments. Else, the next label stack item v_i is *on* P_1, behind e_2. As the rerouting is aware of both link failures e_1, e_2, v_i will be reached as the network remains connected. The remaining part of P_1 beyond v_i cannot contain the failed links e_1 or e_2.

Corollary 1. *If the number of link failures is at most $k = 2$ and the remaining network is connected, TI-MFA is correct even if the label stack is not flushed on a link failure hit.*

B. Simulations

We now evaluate both TI-MFA and TI-LFA in a practical setting. Our simulations are run on Rocketfuel [31] topologies, using both variants of provided link weights and latencies. Specifically, we enumerate all failure cases where TI-LFA and TI-MFA will hit two failed links. As packets following TI-LFA and TI-MFA have an identical path until the second link failure is hit, the instances generated by two link failures and a destination are the same for both algorithms.[7] Furthermore, we discard all disconnected instances.

Our program, written in C++, simulates the behavior of TI-LFA and TI-MFA hop-by-hop, recording 1) if the packet successfully reaches the destination, 2) the maximum label stack size used, and 3) the length of the path taken. An experiment is considered as failed, respectively unsuccessful, if the packet loops indefinitely, for which we record the past packet states (current node, hit failed links, top of the label stack) and check for repetitions in the packet state. We also run TI-LFA/MFA with and without label flushing in case a failure is hit, i.e., four different algorithms in total.

Success rate/reachability analysis. Both variants of TI-MFA reach the destination in all instances (as expected), whereas TI-LFA (no flush) loops indefinitely for roughly one fifth of the experiments, see Fig. 3. In total, over 5 million experiments were performed for all four algorithms.

Of particular interest is comparing the success rate for TI-LFA with (non-standard behavior) and without (standard) flushing. Interestingly, adding label stack flushing to TI-LFA actually increases the failure rate for our simulations in all cases, up to 32% (1221 weights) or a factor of two (6461 latencies). Hence, based on our simulations, we would advocate against adding such a feature to TI-LFA.

Maximum stack size analysis. From the algorithm description of TI-MFA, the size of the label stack is not bounded and could grow with the path length. We plot the distribution for both variants of TI-MFA and TI-LFA in Fig. 4, up to a stack size of 3, not including the destination t $(+1)$. We observe that 3 extra items on the stack suffice for nearly all cases. We list the remaining stack sizes explicitly: TI-LFA with flushing naturally never goes beyond a stack size of 2+1, whereas the successful TI-LFA instances without flushing have at most a stack size of 5+1 (0.025%), respectively 4+1: 0.27% and 3+1: 4%. On the other hand, TI-MFA with flushing never goes beyond a size of 4+1 (4+1: 0.03%, 3+1: 0.37%), and TI-MFA without flushing needs at most a stack size of 6+1 (6+1: 0.0003%, 5+1: 0.02%, 4+1: 0.26%, 3+1: 3.2%).

[6]We assume that if a link is on top of the label stack and this link fails, that popping this unavailable link does not count as flushing.

[7]For this reason, we do not simulate the behavior of TI-LFA/MFA if less than two failures are hit.

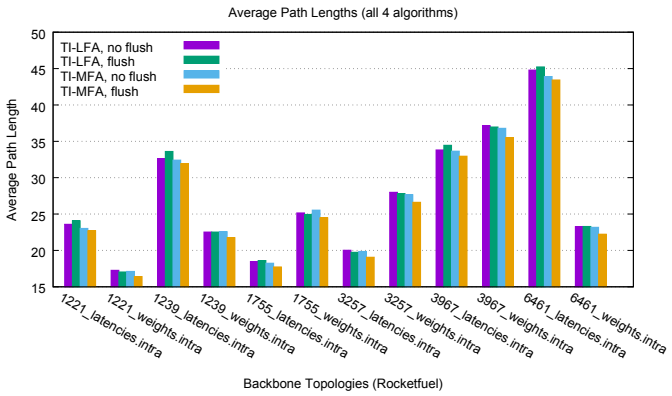

Fig. 5. Average path lengths of the algorithms in successful experiments.

We thus believe that TI-MFA is efficient in its stack size usage and conjecture that a stack size of $2k + 1$ suffices for k link failures, when using flushing.

Path length analysis. A natural question is if the 100% success rate of TI-MFA (in connected topologies) comes at the cost of increased path lengths, which we plot in Fig. 5 for all successful instances, for both latency and weight cost functions. As can be seen, our TI-MFA with flushing performs best in all topologies, whereas the (close) ranking of the other three algorithm variants depends on the specific topology.

VI. Conclusion

This paper studied algorithms for and limitations of resilient routing in SR networks subject to multiple link failures and without invoking the control plane. We believe that our work opens several interesting directions for future research. In particular, it remains to study the optimality of our algorithms, in terms of worst-case resilience but also overhead in terms of required number of segments. Another interesting direction is to study the resilience and other properties of a *given* configuration. In this respect, it is interesting to note that a recent result on MPLS networks [32] can also be applied to SR networks to conduct polynomial-time *what-if analyses*: it is possible to efficiently test whether a certain network configuraiton provides reachability and is policy-compliant even under multiple failures. However, the precise complexity of such tests remains a subject for future research.

Acknowledgments. We would like to thank David Lebrun for helpful discussions regarding segment routing.

References

[1] D. Lebrun, "Reaping the benefits of ipv6 segment routing," in *PhD Thesis (preliminary)*, 2017.
[2] C. Filsfils, P. Francois, S. Previdi, B. Decraene, S. Litkowski, M. Horneffer, I. Milojevic, R. Shakir, S. Ytti, W. Henderickx, J. Tantsura, S. Kini, and E. Crabbe, "Segment routing architecture," in *Segment Routing Use Cases, IETF Internet-Draft*, 2014.
[3] C. Filsfils, S. Previdi, B. Decraene, S. Litkowski, and R. Shakir, "Segment routing architecture," in *IETF Internet-Draft*, 2017.
[4] J. Moy, "OSPF Version 2." RFC 2328 (Standards Track), Apr. 1998.
[5] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: measurement, analysis, and implications," in *ACM SIGCOMM Computer Communication Review*, vol. 41, pp. 350–361, 2011.
[6] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C.-N. Chuah, and C. Diot, "Characterization of failures in an ip backbone," in *Proc. IEEE INFOCOM*, 2004.
[7] A. Shaikh, C. Isett, A. Greenberg, M. Roughan, and J. Gottlieb, "A case study of ospf behavior in a large enterprise network," in *Proc. ACM SIGCOMM Workshop on Internet Measurment*, 2002.
[8] Y. Wang, H. Wang, A. Mahimkar, R. Alimi, Y. Zhang, L. Qiu, and Y. R. Yang, "R3: resilient routing reconfiguration," in *ACM SIGCOMM Computer Communication Review*, vol. 40, pp. 291–302, 2010.
[9] P. Francois, C. Filsfils, A. Bashandy, and B. Decraene, "Topology Independent Fast Reroute using Segment Routing," Internet-Draft draft-francois-segment-routing-ti-lfa-00, Internet Engineering Task Force, Nov. 2013. https://tools.ietf.org/html/draft-francois-segment-routing-ti-lfa-00.
[10] M. Chiesa, A. Gurtov, A. Mdry, S. Mitrovi, I. Nikolaevkiy, A. Panda, M. Schapira, and S. Shenker, "Exploring the limits of static failover routing (v4)," *arXiv:1409.0034 [cs.NI]*, 2016.
[11] M. Chiesa, A. V. Gurtov, A. Madry, S. Mitrovic, I. Nikolaevskiy, M. Schapira, and S. Shenker, "On the resiliency of randomized routing against multiple edge failures," in *Proc. ICALP*, 2016.
[12] 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.
[13] T. Elhourani, A. Gopalan, and S. Ramasubramanian, "Ip fast rerouting for multi-link failures," *IEEE/ACM Trans. Netw*, vol. 24, no. 5, pp. 3014–3025, 2016.
[14] B. Stephens, A. L. Cox, and S. Rixner, "Plinko: Building provably resilient forwarding tables," in *Proc. 12th ACM HotNets*, 2013.
[15] B. Stephens, A. L. Cox, and S. Rixner, "Scalable multi-failure fast failover via forwarding table compression," *SOSR. ACM*, 2016.
[16] M. Borokhovich and S. Schmid, "How (not) to shoot in your foot with sdn local fast failover: A load-connectivity tradeoff," in *OPODIS*, 2013.
[17] Y.-A. Pignolet, S. Schmid, and G. Tredan, "Load-optimal local fast rerouting for dependable networks," in *Proc. IEEE/IFIP DSN*, 2017.
[18] K. Lakshminarayanan, M. Caesar, M. Rangan, T. Anderson, S. Shenker, and I. Stoica, "Achieving convergence-free routing using failure-carrying packets," in *Proc. ACM SIGCOMM*, pp. 241–252, 2007.
[19] F. Hao, M. Kodialam, and T. Lakshman, "Optimizing restoration with segment routing," in *Proc. INFOCOM*, 2016.
[20] C. Filsfils, N. K. Nainar, C. Pignataro, J. C. Cardona, and P. Francois, "The segment routing architecture," in *Global Communications Conference (GLOBECOM), 2015 IEEE*, pp. 1–6, IEEE, 2015.
[21] R. Bhatia, F. Hao, M. Kodialam, and T. Lakshman, "Optimized network traffic engineering using segment routing," in *IEEE INFOCOM*, 2015.
[22] G. Trimponias, Y. Xiao, H. Xu, X. Wu, and Y. Geng, "On traffic engineering with segment routing in sdn based wans," *arXiv preprint arXiv:1703.05907*, 2017.
[23] L. Davoli, L. Veltri, P. L. Ventre, G. Siracusano, and S. Salsano, "Traffic engineering with segment routing: Sdn-based architectural design and open source implementation," in *Proc. 4th European Workshop on Software Defined Networks (EWSDN)*, pp. 111–112, 2015.
[24] M.-C. Lee and J.-P. Sheu, "An efficient routing algorithm based on segment routing in software-defined networking," *Comput. Netw.*, vol. 103, pp. 44–55, July 2016.
[25] F. Aubry, D. Lebrun, S. Vissicchio, M. T. Khong, Y. Deville, and O. Bonaventure, "Scmon: Leveraging segment routing to improve network monitoring," in *Proc. IEEE INFOCOM*, 2016.
[26] A. Giorgetti, P. Castoldi, F. Cugini, J. Nijhof, F. Lazzeri, and G. Bruno, "Path encoding in segment routing," in *Proc. IEEE GLOBECOM*, 2015.
[27] S. Salsano, L. Veltri, L. Davoli, P. L. Ventre, and G. Siracusano, "Pmsrpoor man's segment routing, a minimalistic approach to segment routing and a traffic engineering use case," in *Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS)*, pp. 598–604, 2016.
[28] R. Hartert, S. Vissicchio, P. Schaus, O. Bonaventure, C. Filsfils, T. Telkamp, and P. Francois, "A declarative and expressive approach to control forwarding paths in carrier-grade networks," in *ACM SIGCOMM Communication Review*, vol. 45, pp. 15–28, 2015.
[29] M. Chiesa, I. Nikolaevskiy, S. Mitrovic, A. Gurtov, A. Madry, M. Schapira, and S. Shenker, "On the resiliency of static forwarding tables," *IEEE/ACM Trans. Netw*, vol. 25, no. 2, pp. 1133–1146, 2017.
[30] K.-T. Foerster, Y.-A. Pignolet, S. Schmid, and G. Tredan, "Local fast failover routing with low stretch," in *Proc. ACM SIGCOMM Computer Communication Review (CCR)*, 2018.
[31] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson, "Measuring ISP topologies with rocketfuel," *IEEE/ACM Trans. Netw.*, vol. 12, no. 1, pp. 2–16, 2004.
[32] S. Schmid and J. Srba, "Polynomial-time what-if analysis for prefix-manipulating mpls networks," in *Proc. IEEE INFOCOM*, 2018.