Robust LFA Protection for Software-Defined Networks (RoLPS)

Daniel Merling, Steffen Lindner, and Michael Menth
Chair of Communication Networks, University of Tuebingen, Germany
{daniel.merling, steffen.lindner, menth}@uni-tuebingen.de

***Abstract*—In software-defined networks, forwarding entries on switches are configured by a controller. In case of an unreachable next-hop, traffic is dropped until forwarding entries are updated, which takes significant time. Therefore, fast reroute (FRR) mechanisms are needed to forward affected traffic over alternate paths in the meantime. Loop-free alternates (LFAs) and remote LFAs (rLFAs) have been proposed for FRR in IP networks. However, they cannot protect traffic for all destinations and some LFAs may create loops under challenging conditions.**

This paper proposes robust LFA protection for software-defined networks (RoLPS). RoLPS augments the coverage of (r)LFAs with novel explicit LFAs (eLFAs). RoLPS ranks available LFAs according to protection quality and complexity for selection of the best available LFA. Furthermore, we introduce advanced loop detection (ALD) so that RoLPS stops loops caused by LFAs. We evaluate RoLPS-based protection variants on a large set of representative networks with unit and non-unit link costs. We study their protection coverage, additional forwarding entries, and path extensions for rerouted traffic, and compare them with MPLS facility backup. Results show that RoLPS can protect traffic against all single link or node failures, and against most double failures while inducing only little overhead. We implement FRR on the P4-programmable switch ASIC Tofino and provide a control plane logic based on RoLPS. Measurement results show that the prototype achieves a throughput of 100 Gb/s, reroutes traffic within less than a millisecond, and reliably detects and drops looping traffic.

***Index Terms*—Software-Defined Networking, P4, Loop-Free Alternates, Resilience, Link Protection, Node Protection, Scalability,**

I. Introduction

Software-defined networking (SDN) separates data plane and control plane of forwarding nodes. A controller computes and installs forwarding rules on data plane devices to instruct them how to process data packets. Packet forwarding is impaired when a next-hop becomes unreachable due to a failure, i.e., a failed link or a failed node. Without controller interaction, switches drop affected packets. However, notification of the controller, recomputation of forwarding rules, and their installation on data plane devices takes a considerable amount of time. This outage time is too long, in particular for the transport of realtime traffic.

In IP networks fast reroute (FRR) mechanisms are used to quickly reroute packets via pre-computed backup paths while forwarding entries are recomputed. FRR would also be helpful in SDN to forward traffic with unreachable next-hops without controller interaction via alternate paths. However, SDN forwarding devices often have limited forwarding tables so that adding many forwarding entries for FRR purposes may be problematic. Loop-free alternates (LFAs) are a well-known FRR method for IP networks that requires no additional forwarding entries so that we consider them in this work. LFAs constitute alternative next-hops that successfully forward traffic towards the destination when the default next-hop is unreachable. The authors of [1] proposed to use LFAs to protect traffic without controller interaction in SDN-based networks. However, LFAs suffer from two major shortcomings. First, they cannot protect traffic for all destinations against single link failures (SLF) and single node failures (SNF). Second, some LFAs may cause rerouting loops in case of node failures or multiple failures.

In previous work [2] we improved the usage of LFAs in software-defined networks. We introduced explicit LFAs (eLFAs) based on explicit tunnels to protect destinations that cannot be protected by other LFAs. We proposed advanced loop detection (ALD) to detect and stop loops, which prevents severe overload that may happen with LFAs in failure cases. We described loop avoidance (LA), which leverages ALD, ranks available LFAs according to their protection quality and overhead, and chooses the best one. Furthermore, we showed how LA can be implemented in OpenFlow. Finally, a simulation-based evaluation showed that LA can protect all traffic in SDN networks against SLF and SNF and with less overhead compared to other FRR methods.

his paper is an extension of [2] with the following advances. (1) We augment eLFAs with explicit multipoint-to-point rerouting tunnels. This significantly decreases the required number of additional forwarding entries for explicit tunnels. (2) We modify ALD so that it can detect and stop loops faster while being implementable on P4 devices. (3) We update the simulative evaluations according to the new mechanisms. (4) We include topology-independent LFAs (TI-LFAs) [3] in the simulative evaluations because they are conceptually similar to eLFAs. (5) We improved the overall presentation, including a renaming of LA into RoLPS as the name LA did not capture the entire concept. (6) We implement a prototype of RoLPS on the P4-programmable switching ASIC Tofino featuring LFAs, rLFA, eLFA, and ALD, and a RoLPS-based SDN controller, and thereby, show its technical feasibility. (7) We demonstrate that the technical solution performs well by showing that the prototype operates at 100 Gb/s, reroutes traffic within less than

The authors acknowledge the funding by the Deutsche Forschungsgemeinschaft (DFG) under grant ME2727/1-2. The authors alone are responsible for the content of the paper.

a millisecond, and reliably detects and drops looping traffic.

The paper is structured as follows. In Section II we discuss related work. Then, we review state of the art for LFAs in Section III. Section IV introduces eLFAs and ALD for improved protection of the SDN data plane, and a RoLPS-based control plane logic for that features and existing LFAs. Section V describes the simulative evaluation methodology and discusses performance results based on comprehensive study. We present the implementation of a P4-based hardware prototype in Section VI. We evaluate its performance in Section VII by measurements. Finally, we conclude the paper in Section VIII. A table of acronyms and a glossary are provided at the end of the paper to facilitate the reading.

II. Related Work

In this section we describe related work. First, we discuss legacy FRR mechanisms to position LFAs. Then, we review FRR for SDN.

A. *FRR in Legacy Networks*

Rai et al. [4], Raj et al. [5], and Papan et al. [6] present surveys that provide a wide overview of FRR in legacy networks. Hutchinson et al. [7] discuss the architecture and design of resilient network systems, i.e., specifying and realizing appropriate components. They review state-of-the-art contributions and identify future research issues.

1) MPLS Networks: For MPLS [8] two major FRR mechanisms have been proposed [9]. One-to-one backup reroutes packets on preconfigured paths that avoid the failure. Facility backup tunnels the packets locally around the failure to the next-hop for link protection, or to the next-next-hop for node protection. Only recently, the authors of [10] propose a loop detection mechanism for MPLS. It is based on special MPLS labels that are pushed on the MPLS header stack when a packet is rerouted. This allows nodes to detect whether a packet has already been rerouted.

2) IP Networks: Not-via addresses [11] protect both IP and MPLS networks. The routing table of a node contains one additional forwarding entry for every outgoing link. When the default next-hop is unreachable, those additional entries are used to deviate the packet from its shortest path through a tunnel around the failure. This causes a similar path layout as MPLS facility backup [12]. Failure insensitive routing (FIR) [13] leverages interface-specific routing tables to encode failure information. Depending on the ingress interface, packets are rerouted on precomputed backup paths around the failure. Multiple routing configurations (MRCs) [14] implement multiple disjoint routing topologies so that always at least one topology provides a working path towards the destination despite the failure. For each topology, an entire set of forwarding entries is required which at least doubles the amount of forwarding entries. Maximally redundant trees (MRTs) [15] leverage a similar approach. A red and a blue set of backup forwarding entries are computed so that at least one set delivers the packet in case of a failure. However, MRTs triple the number of forwarding entries in the network and may lead to extensive backup paths [16]. LFAs can be combined with MRTs to reduce backup path length and link load [17]. Independent directed acyclic graphs (IDAGs) [18] compute only two sets of maximally disjoint forwarding entries, i.e., doubling the amount of forwarding entries so that one is working in case of a failure. The authors of [19] encode failure information in the packet header. Nodes leverage this information to identify the failure and reroute packets on disjoint paths around it.

3) LFA-Based Protection: LFAs [20] with either link or node protection locally reroute packets around the failure on shortest paths. Therefore, they do not require additional forwarding entries but cannot protect all destinations. Csikor et al. [21], [22] increase the number of protected destinations by optimizing link costs. rLFAs [23]–[25] augment LFAs to increase the number of protected destinations by rerouting packets to remote nodes through shortest path tunnels. They do not need additional forwarding entries but still cannot protect all destinations. The performance of both LFAs and rLFAs can be enhanced by adding links to the network [26]. In [27], the authors present a self-configuring extension for LFAs based on probes. It installs alternative hops in other nodes to prevent rerouting loops. Topology-independent LFAs (TI-LFAs) [3] leverage segment routing (SR) [28] to protect against failures. SR is based on forwarding instructions in the packet header which may be stacked. TI-LFAs leverage SR to implement explicit tunnels to remote nodes. As eLFAs leverage explicit tunnels, too, they can be viewed as a very specific but rather untypical form of TI-LFAs.

B. *FRR Protection in SDN*

We discuss FRR in the context of SDN. We first address general FRR approaches for SDN and then we discuss related work for FRR in OpenFlow- and P4-based networks.

1) FRR in SDN: There have been many proposals to make the SDN control plane more resilient [29]. However, there are only very few efforts to protect traffic in the data plane. If the controller is notified about the failure, it may update its topology, and recompute and install updated forwarding entries. Sharma et al. [30] measure that recomputation takes about 80-100 ms. However, the authors clarify that this number highly depends on the number of affected flows, path lengths, and traffic bursts in the control network. In particular, it is likely that the time for rerouting is significantly higher in larger networks. Da Silva et al. [31] and Chiesa et al. [32] present surveys that give overviews of FRR in SDN with significantly faster protection than recomputation of forwarding entries.

2) OpenFlow-Based FRR: FRR capabilities have been introduced in OpenFlow with Version 1.1. The authors of [33] provide a BFD-based protection scheme for earlier OpenFlow versions than 1.1. It is based on a bidirectional forwarding detection (BFD) where nodes periodically exchange information about their reachability. Van Adrichem et al. [34] measure that failure detection takes about 3-30 ms on the software-based Open vSwitch depending on the configuration of the BFD. SlickFlow [35] encodes primary and backup paths in the packet header to reroute packets when an unavailable egress port is selected. SPIDER [36] leverages additional state in the OpenFlow pipeline. Packet labels carry reroute and connectivity information. Braun et al. [1] propose loop detection for LFAs (LD-LFA) which increases the number of protected destinations but may erroneously drop packets. The authors of [37] use labels in the packet header that carry failure information to trigger rerouting in other nodes. Cevher et al. [38] implement MRCs in OpenFlow. The authors of [39] implement multi-topology routing which uses virtual topologies to provide redundancies in routing tables. If a failure is detected, packet forwarding is switched to a topology which is not affected by the failure. BOND [40] optimizes memory management for backup rules and leverages global hash tables to accelerate failure recovery.

3) P4-based FRR: P4 does not provide native FRR capabilities. Therefore, the hardest challenge is to provide the data plane devices with information about which neighbors are reachable, i.e., which port is up or down.

Sedar et al. [41] propose to use registers to store information about which egress port is up or down. Depending on the port status registers, primary or backup forwarding actions are triggered. However, the authors depend on a local agent to populate the registers. Shared Queue Ring (SQR) [42] caches recent traffic in a delayed queue. If a link failure is detected, the cached traffic is sent over alternative paths. Lindner et al. [43] implement 1+1 protection in P4 which replicates traffic, includes sequence numbers, and sends it over disjoint paths. The joint head end of those paths deduplicates the traffic. Hirata et al. [44] implement a FRR scheme in P4 which is similar to MRCs. Multiple routing topologies with disjoint paths are deployed. A field in the packet header identifies the topology which should be used for forwarding. D2R [45] is a resilience mechanism which works entirely in the data plane. When a failure is detected, the data plane itself, i.e., the failure-detecting switch, recomputes a new path to the destination. A primitive for reconfigurable fast reroute (PURR) [46] stores additional egress ports for each destination. During packet processing, the first working egress port is selected for forwarding.

III. LFAs: State of the Art

We review LFAs and remote LFAs (rLFAs) and give an overview of previous work regarding loop detection for LFAs. Finally, we explain topology-independent LFAs (TI-LFAs).

A. LFAs and rLFAs

In this subsection we introduce the concept of LFAs and rLFAs. Then, we discuss three important properties of LFAs. First, we differentiate protection levels for LFAs, i.e., link protection and node protection. Second, we explain the influence of links cost on LFA-based protection. Third, we point out that LFAs may generate loops under some conditions.

1) Concept: LFAs [20] have been proposed in the context of FRR for IP networks to quickly protect traffic against the failure of links and nodes while primary forwarding entries are recomputed.

A point of local repair (PLR) denotes a node that detects an unreachable next-hop and reroutes affected traffic to some other neighbor. However, some neighbors would send the traffic back to the PLR, which creates a loop. The other neighbors can forward the traffic without creating a loop and are called loop-free alternates (LFAs). They are used by a PLR to reroute traffic in case of a failure.

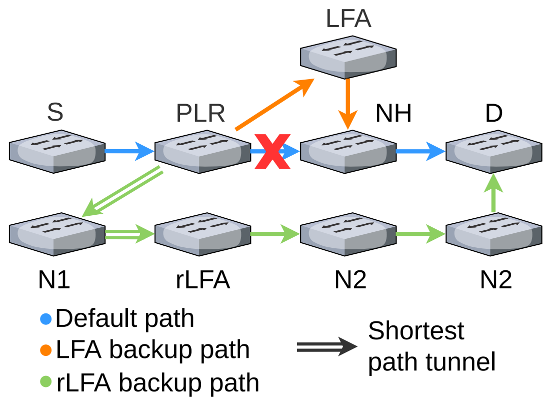


Figure 1: In case of a failure, a PLR may reroute a packet to an LFA or tunnel it via a shortest path to a rLFA. The (r)LFA then forwards the packet via a shortest path to its destination.

LFAs are illustrated in Figure 1. Traffic is forwarded on shortest paths. A packet is sent from sender *S* to destination *D*. The default path is via *PLR* and *NH*. When *PLR* cannot reach its next-hop *NH* due to a link failure, it cannot reroute the packet via neighbors *S* or *N1* as they forward traffic towards *D* to *PLR*, which creates a loop. However, *PLR* may reroute the packet via *LFA* which can forward the packet to *D*. Thus, the node *LFA* represents an LFA for *PLR* with respect to destination *D*.

We now assume that *NH* fails so that *LFA* has no working path towards *D*. If *PLR* reroutes the packet to *LFA*, *LFA* may use *PLR* as an LFA and return the packet. Thus, a loop occurs.

Remote LFAs (rLFAs) [23]–[25] have been introduced to protect more destinations than LFAs by sending packets through shortest path tunnels to remote nodes. In our example, the node *rLFA* is an rLFA for *PLR* with respect to destination *D*. If *NH* fails, *PLR* may tunnel the packet to *rLFA* which decapsulates the packet and sends it to *D* via a shortest path.

2) Protection Level: We already observed that some (r)LFAs protect only against link failures, others protect also against node failures. The first are classified as link-protecting (LP), the second as node protecting (NP). A link-protecting LFA (LP-LFA) forwards traffic to a destination via a path that avoids a PLR's failed link. A node-protecting LFA (NP-LFA) forwards traffic to a destination via a path that avoids a PLR's failed next-hop. Thus, NP-LFAs are also LP-LFAs, but not vice-versa. Therefore, a PLR can protect more destinations with LP-LFAs than with NP-LFAs. For some destinations, there may be no LP-LFA or NP-LFA at all. Then, rLFAs may help.

3) Influence of Link Cost: Networks are configured without link costs, i.e., unit link cost networks, or with link costs, i.e., non-unit link cost networks, e.g., for traffic-engineering. (r)LFAs have different protection properties in unit link cost networks than in non-unit link cost networks. The authors of

[1], [2] showed that for some destinations there is no LP-LFA or NP-LFA in both unit-link cost networks and non-unit link cost networks. Then, some destinations may be protected with rLFAs. Csikor et al. [25] proved that there is always an LP-rLFA for any destination in unit link cost networks. However, in [2] we showed that this is not the case in non-unit link cost networks. Furthermore, we showed that in both unit link cost networks and non-unit link cost networks there is not always an NP-rLFA for a destination [2] although there are more NP-rLFAs in unit link cost networks. Thus, in general, more destinations can be protected in unit link cost networks with (r)LFAs than in non-unit link cost networks.

4) LFA-Generated Loops: Forwarding loops in networks are problematic for two reasons. First, the traffic cannot reach its destination. Second, looping traffic consumes bandwidth, which may lead to packet loss for other traffic. However, looping traffic does not loop forever because the TTL field in the IP header limits the number of forwarding hops. As TTL=64 is a typical value, looping traffic can easily waste the 30-fold of the capacity it would normally occupy on a link. Therefore, routing loops are detrimental and should be avoided.

Depending on their protection level (r)LFAs may cause rerouting loops in specific failure scenarios. We distinguish and order four failure scenarios: single link failure (SLF) $<$ single node failure (SNF) $<$ double link failure (DLF) $<$ single link and single node failure (SLF+SNF).

LP-(r)LFAs do not cause rerouting loops for SLF but they may cause loops in other scenarios. NP-(r)LFAs prevent loops for both SLF and SNF [2], but fewer destinations can be protected by them. In case of multiple failures, even NP-(r)LFAs may generate loops. Some LP- or NP-(r)LFAs have the “downstream” property [12] and they avoid loops in case of multiple failures. However, only a few LFAs have that property so that only a few destinations can be protected by them. We do not consider them any further in this study.

B. Loop Detection for LFAs

The authors of [1] propose loop detection based on bit strings. They use it in combination with LFAs to protect more destinations by LFAs without suffering from loops. In addition, they suggest to protect destinations with LFAs with the highest possible protection level to maximize the coverage against link and node failures. They call this approach LD-LFA.

1) Loop Detection Based on Bit Strings: The loop detection in [1] requires a bit string in the packet header to indicate nodes that have rerouted the packet before. Each node in the network is associated with a bit position. If a packet is rerouted, the node activates it bit in the packet’s header. If a node receives a packet with its corresponding bit activated, the packet is dropped.

The authors suggest an implementation in OpenFlow but do not deliver a prototype. An advantage of this approach is that a packet can be rerouted by multiple nodes. A disadvantage is the missing scalability. Bit strings in packet headers should be small. In OpenFlow, MPLS labels may be reused for that purpose, but they are only 4 bytes long which is not enough to number all nodes of a large network. Therefore, multiple nodes may be associated with the same bit. If one of these nodes reroutes a packet, the packet is dropped if it is received by another of those nodes. This causes erroneous drops for rerouted packets.

2) LFA Selection: For some PLRs there are several LFAs available for a specific destination. The authors of [1] suggested to prefer NP-LFAs over LP-LFAs in such a case. They showed for various network topologies that significantly fewer destinations can be protected by NP-LFAs than by LP-LFAs. Therefore, they suggested to protect the remaining destinations with LP-LFAs if possible. In addition, they proposed to utilize loop detection based on bit strings to avoid rerouting loops caused by LP-LFAs. They did not consider rLFAs.

C. Topology-Independent LFAs

In this subsection we explain topology-independent LFAs (TI-LFAs) [3]. First, we review segment routing (SR) [28]. Then, we describe TI-LFAs.

1) Segment Routing: IP networks leverage destination-based forwarding to deliver packets. That is, a packet carries the IP address of the destination in its header which is used by network devices to determine the appropriate next-hop according to entries in a forwarding table. In contrast, with SR the packet source determines the processing of a packet. To that end, SR leverages forwarding instructions in the packet header. The packet source constructs a set of header segments that are added to the packet. Each header segment corresponds to a specific action. Nodes process a packet according to the segments in its header. To that end, network devices maintain a certain number of forwarding entries to map a header segment to a specific action.

Currently, there are two major technologies that implement SR. SRv6 [47] is based on IPv6 and its extension header. Each IPv6 address in the extension header corresponds to one header segment. SR-MPLS ([48]) leverages stacked MPLS labels, i.e., the header stack, where each MPLS label is a header segment. To facilitate readability we only use the terminology of SR-MPLS, i.e., header stack and label, in the following.

Header segments may instruct nodes to perform arbitrary actions, e.g., forwarding a packet, pushing or removing other header segments, etc. In the following we focus on two specific types of header segments. The first type are header segments for global forwarding. We refer to such header segments with the term “global labels”. Global labels instruct the nodes to forward a packet according to shortest paths towards a specific destination. As a result, a global label is similar to destination-based forwarding in IP networks. At the destination the global label is removed and the node processes the next header segment. When global labels are used for all destinations, every nodes requires $n-1$ forwarding entries where n is the number of destinations in the network. The second type are header segments for local forwarding. We refer to that kind of header segments with the term “local labels”. Local labels instruct nodes to forward a packet over a specific link towards a next-hop. Before a node forwards a packet to the NH, it removes its local label from the header stack. When local labels are used for all nodes, every node requires d forwarding entries where d is the number of neighbors of that node.

A source may construct a header stack that contains both global labels and local labels. As a result, forwarding differs depending on which type of label is on top of the header stack. On some subpaths the packet is forwarded according to a global label and on some subpaths the packet is forwarded according to a local label.

2) Concept of TI-LFAs: TI-LFAs leverage SR to forward packets on explicit paths around a failure. That is, TI-LFAs are not restricted to shortest paths because they construct a header stack with explicit forwarding instructions so that the packet avoids the failure. As a result, TI-LFAs with LP protect against any single link failure independently of link costs, and TI-LFAs with NP protect against any single node failure independently of link costs. However, multiple header segments may be necessary which increases the size of the header stack and thereby the overhead in terms of additional packet headers. The authors of TI-LFAs state that "in an MPLS world, this may create a long stack of labels to be pushed that some hardware may not be able to push." ([3], 2021, p. 6).

The size of a specific header stack depends on how the explicit backup path is implemented. The straightforward approach is to use one explicit forwarding instruction for every hop, i.e., local labels. However, this requires one header segment for each hop which causes large header stacks. The size of the header stack can be reduced if subpaths of the explicit path are implemented with already existing global labels. That is, one global label replaces multiple local labels. This is possible when working shortest paths are subpaths of the explicit path. However, this may not be possible for all subpaths because sometimes no working shortest subpath is available due to the failure.

The authors of [3] do not specify how the header stack to implement explicit paths is built. In particular, this is an optimization that highly depends on the failure scenario, topology, link costs, and path selection. Therefore, we see research potential for the optimization of the TI-LFA header stack. This, however, is out of scope of this document. In the following we assume that TI-LFAs implement explicit paths only with local labels.

IV. Robust LFA Protection for Software-Defined Networks (RoLPS)

LFAs originated from IP networks. They are attractive for SDN because they entail only little overhead in terms of additional forwarding state. However, they have three major shortcomings. They have been designed only for shortest-path routing based on link costs, they cannot protect all destinations, and they may cause loops under some conditions.

In the following we explain how LFAs can be applied in SDN which allows for general destination-based forwarding. We present explicit LFAs so that all destinations can be protected in case of a failure, provided they can be physically reached by a working path. We describe an advanced loop detection method to detect and stop loops and prevent erroneous packet drop after up to n reroute actions. Finally, we propose how to utilize these components and consider different protection variants.

A. Applicability of LFAs for SDN

In the context of IP networks, equations considering link costs are used to classify neighboring nodes into non-LFAs, LP-LFAs, and NP-LFAs with regard to some destination [12]. Forwarding in SDN does not need to follow shortest path routing based on link costs, but general destination-based forwarding may be applied. Therefore, we briefly explain how (r)LFAs can be used in that context. Essentially, we need to classify neighboring nodes into no-LFAs, LP-LFAs, and NP-LFAs. A PLR's neighboring node is

- no LFA if its standard forwarding procedure forwards the traffic to the destination via a path containing the PLR.
- an LP-LFA if its standard forwarding procedure forwards the traffic to the destination via a path that does not contain the link from PLR to its next-hop towards the destination.
- an NP-LFAs if its standard forwarding behavior forwards the traffic to the destination via a path that does not contain the PLR's next-hop towards the destination.

This definition can be applied to normal LFAs, rLFAs, and to eLFAs that are presented later in this section.

Path computation is not a focus of this paper. To limit the parameter space for ease of understanding, we consider in the evaluation in Section V link-cost-based forwarding which is a special case of the more general destination-based forwarding.

B. Explicit LFAs

We first give an example where (r)LFAs cannot protect a destination. Such destinations can be protected by explicit LFAs (eLFAs) which are based on explicit tunnels. However, explicit tunnels require additional forwarding entries. In [2] we suggested to implement explicit tunnels with explicit point-to-point rerouting tunnels. In this paper, we propose explicit multipoint-to-point rerouting tunnels as an alternative which requires significantly less additional forwarding entries. Finally, we explain the relation between eLFAs and TI-LFAs.

1) Protection through Explicit Tunnels: The network in Figure 2 forwards traffic on shortest paths based on costs that are annotated on the links. *PLR* sends a packet to *D* but the primary next-hop is unreachable. Although there is a physical path via *N1* and *eLFA*, there is no (r)LFA available. *N1* is not an LFA because it sends traffic to *D* via *PLR*. *eLFA* cannot serve as rLFA because the shortest path from *PLR* to *eLFA* traverses *D*. The problem can be solved by setting up an explicit tunnel via *N1* to *eLFA* a priori. If *D* is no longer reachable, *PLR* can send the packet over that explicit tunnel, and from *eLFA* the packet reaches *D* via a shortest path. Thus, *eLFA* is an eLFA for *PLR* with regard to *D*.

2) Explicit Point-to-Point Rerouting Tunnels: Now we explain the concept of explicit point-to-point tunnels which we introduced in [2]. In Subsection VI-C3 we describe technical details about the implementation of explicit tunnels in general with P4.

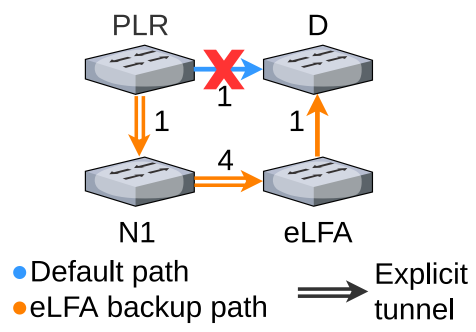


Figure 2: In case of a failure, a PLR may reroute a packet to an eLFA via an explicit tunnel which then forwards the packet via a shortest path to its destination. In contrast to rLFAs, the PLR cannot reach the eLFA via a shortest path.

Explicit point-to-point rerouting tunnels do not follow standard paths. Therefore, they are configured with a unique identifier, e.g., a unique number or IP address, in advance. When a PLR reroutes a packet through an explicit point-to-point rerouting tunnel, it adds the identifier of that tunnel to the packet. Nodes use the identifier to forward the packet along the explicit path. To that end, the nodes along an explicit path need additional forwarding entries for the identifier of that tunnel. Additional forwarding entries for FRR purposes are undesired overhead for the data plane as they limit its scalability.

3) Explicit Multipoint-to-Point Rerouting Tunnels: The overhead of additional forwarding entries from explicit tunnels can be reduced by using explicit multipoint-to-point tunnels. That is, the explicit tunnels from multiple PLRs towards the same endpoint, i.e., an eLFA, build a destination tree where the PLRs are the sources and the eLFA is the sink. Such an explicit multipoint-to-point rerouting tunnel corresponds to a specific eLFA and is identified by a single unique identifier. When a PLR reroutes a packet towards a specific eLFA, it adds the identifier of the corresponding multipoint-to-point rerouting tunnel to the packet. As a result, overlapping subpaths of explicit tunnels towards the same eLFA require only a single additional forwarding entry in nodes along that subpath. Therefore, multipoint-to-point rerouting tunnels are prefered over point-to-point rerouting tunnels. We evaluate the effect of multipoint-to-point rerouting tunnels in comparison to point-to-point rerouting tunnels in Section V-C.

4) Relation to TI-LFAs: Explicit tunnels can be implemented in different ways. We suggest eLFAs which implement explicit tunnels with a single tunnel header and additional forwarding entries in forwarding devices. Alternatively, TI-LFAs leverage a header stack with explicit forwarding instructions based on already available forwarding entries. Section III-C contains details about the construction of the TI-LFA header stack. Either way creates overhead to implement explicit tunnels. In Section V-C we evaluate the number of additional forwarding entries that are required by eLFAs. In Section V-D we quantify the size of the packet header stack when TI-LFAs are used.

C. Advanced Loop Detection

The loop detection method in [1] suffered from scalability problems. Therefore, we propose that packets are dropped if they are rerouted more than n times. This requires only a counter in the packet header which is increased with each reroute action. When the counter reaches the limit, the packet is dropped. We denote this advanced loop detection (ALD). Generally, ALD can be configured to support an arbitrary number of redirects. However, a large number can be counter-productive as packets are dropped later in case of loops and consume more bandwidth. In our context, we allow a packet to be rerouted twice so that double failures can be survived.

1) Implementation in OpenFlow: Due to technical restrictions of OpenFlow, conditions can be checked only at the beginning of the forwarding pipeline. However, at that stage, there is no knowledge about the packet's next hop and failed interfaces. Fortunately, it is possible to increase the reroute counter while rerouting. Thus, only the next-hop of a rerouted packet can determine whether the packet's reroute counter exceeds the limit and then the packet is dropped. This wastes bandwidth on the last link over which the packet was rerouted.

We provided a more detailed sketch of an OpenFlow-based implementation of ALD in [2]. That particular proposal was still based on bit strings. However, it avoids erroneous packet drops after a single reroute in contrast to the solution in [1].

2) Implementation in P4: P4 offers more implementation flexibility. Therefore, it is possible to check whether a packet is rerouted and whether its rerouting counter exceeds the limit before the packet is forwarded to the egress port. As a consequence, packets are dropped before transmission, which does not waste bandwidth. More details about the P4-based implementation of ALD are given in Section VI-D.

D. RoLPS Protection Variants

With SDN a controller configures flow entries on data plane devices. Alternative paths can be configured so that the device can switch over to a secondary next-hop if the first hop becomes unreachable. The secondary next-hop is also configured by the controller. In this section we present a ranking scheme for LFAs to choose the best one as a secondary next-hop. We further define protection variants and propose a corresponding nomenclature.

1) LFA Ranking: A controller can classify neighboring and remote nodes of a potential PLR into LFAs, rLFAs, and eLFAs, and as LP or NP for a specific destination. These LFAs can be ranked according to their protection level, i.e., NP is better than LP. Recall that NP-LFAs are also LP-LFAs, but not

Rank	LFA Type
0	NP-LFA
1	NP-rLFA
2	NP-eLFA
3	LP-LFA
4	LP-rLFA
5	LP-eLFA

Table 1: Ranking of LFA types according to protection level and complexity. Preference is given to LFAs with lower rank number.

Mechanism	C-LFA (nLD-LP-LFA)	C-rLFA (nLD-LP-rLFA)	LD-LFA (ALD-NP-LFA)	ALD-NP-rLFA	ALD-LP-eLFA	ALD-NP-eLFA
Loop detection			●	●	●	●
Protection against all SLF		o		o	●	●
Protection against all SNF						●
Additional forwarding entries					●	●

Table 2: Properties of protection variants.
Legend: o = only for unit link costs; ● = independent of link costs.

vice-versa. They can also be ranked according to complexity. Normal LFAs are simplest as they do not require tunneling. eLFAs are most complex as they entail additional forwarding entries for explicit tunnels.

With SDN, it is important to have an alternative next-hop in case the primary next-hop is unreachable as it may take too long until the forwarding is fixed by the controller. Therefore, we rank LFAs first according to their protection level and then according to their complexity. This yields the ranking given in Table 1. The ranking is used to select the best available LFA during computation.

2) Protection Variants: We define several protection variants with respect to loop detection, LFA complexity, and protection level. The following naming scheme is used: {nLD, ALD}-{LP, NP}-{LFA, rLFA, eLFA}. Loop detection may be activated or not {ALD, nLD}. Either the LP property is sufficient or NP is desired {LP, NP}. Only normal LFAs may be allowed, normal and rLFAs may be allowed, or normal, remote, and explicit LFAs are supported {LFA, rLFA, eLFA}.

eLFAs are preferably implemented with explicit multipoint-to-point rerouting tunnels (see Section IV-B3). However, for comparison we sometimes refer to eLFAs with point-to-point tunnels. To that end, we add the suffix "-p2p" to the protection variant. We omit a suffix for eLFAs with multipoint-to-point rerouting tunnels because this is the preferable way That is, *-*-eLFA refers to protection variant with eLFAs with multipoint-to-point rerouting tunnels and *-*-eLFA-p2p refers to protection variants with eLFAs with point-to-point rerouting tunnels.

If a protection variant requires the NP property, the LFA selection process starts with the search for an LFA of rank 0. If the search is successful, this LFA is configured as secondary next-hop for a specific destination, and the algorithm stops. Otherwise the search continues with the next higher rank number. This possibly continues up to rank 5. That means, NP-(e/r)LFAs are preferentially utilized, but LP-(e/r)LFAs may be used if the destination cannot be protected otherwise. This is needed, e.g., if the protected next-hop is the destination. If no LFA has been found for the last rank, there is no physical connection between PLR and destination.

If a protection variant requires only the LP property, the LFA selection process starts with the search for an LFA of rank 3. The algorithm also stops if no LFAs has been found for the last rank. In that case there is no physical path between PLR and destination. Note that LFAs of rank 3 may also be NP as every NP-LFA also fulfills the LP property. LP-LFAs are just not preferred over NP-LFAs when the protection variant requires only the LP property.

Protection variants requiring the NP property may still suffer from loops since some destinations can be protected only with LP-(e/r)LFAs. For example they occur when the destination of a flow fails. nLD-LP-LFA and nLD-LP-rLFA leverage only the classic LP-LFAs [20] and LP-rLFAs [23]. They are widely used in IP networks and we denote them as the classic LFA and rLFA variants (C-LFA, C-rLFA). ALD-NP-LFA[1] has been investigated as a preferred protection variant in [1] under the name LD-LFA.

Table 2 summarizes the most important protection variants investigated in our study. It summarizes properties regarding protection level and complexity. ALD-mechanisms prevent loops in any failure scenario. *-*-rLFA protect against all protectable SLF in networks with unit link costs. *-*-eLFA methods achieve that protection level even in networks with non-unit link costs. *-NP-eLFA protects even against all protectable SNF in networks with either unit or non-unit link costs.

V. Simulative Performance Evaluation of LFA-Based Protection

In this section we analyze the efficiency of LFA-based FRR mechanisms. First, we describe the methodology. The performance metrics of interest are protection coverage, required amount of additional forwarding entries, required amount of header segments for TI-LFAs, and path lengths. We compare them for RoLPS protection variants and other well-known FRR mechanisms. Finally, we discuss the presented results.

A. Methodology

We explain the methodology for the simulation-based evaluation. We describe the general approach, and discuss the topology data set and link costs used in the evaluation.

1) General Approach: We take a network topology including link costs and a RoLPS protection variant as input parameters. Then we compute LFAs according to Section IV-D. We evaluate different protection variants against various sets of failure scenarios, i.e., $\mathcal{S} \in \{\mathrm{SLF}, \mathrm{SNF}, \mathrm{DLF}, \mathrm{SLF}{+}\mathrm{SNF}\}$ (see Section III-A4). To that end, we consider all source-destination pairs $f \in \mathcal{F}$ in the network and analyze how their traffic is forwarded in a specific failure scenario $s \in \mathcal{S}$.

Although RoLPS works for general destination-based forwarding (see Section IV-A), we limit the evaluation to shortest paths routing based on link costs to reduce the parameter space.

[1]Approximation of LD-LFAs with better loop detection.

2) Network Topologies: We evaluate 205 wide area, commercial, research, and academic networks from the Internet topology zoo [49] and three typical data center topologies (fat-tree, DCell, BCube) which were studied in [1]. For each topology we calculate both average values and maximum values for the considered metrics. We explain these metrics in Sections V-B1, V-C1, and V-E1. We visualize the results in bar diagrams or complementary cumulative distribution functions (CCDFs).

3) Link Costs: In Subsection III-A3 we showed that link costs have a significant impact on the protection properties of LFAs. To account for that fact, we perform evaluations on then networks with both unit link cost and non-unit link cost. However, the topology zoo does not include link costs for all networks. Therefore, we calculate link costs on all networks as proposed in [50]. For each link we derive the specific load based on a homogeneous traffic matrix, shortest paths, and unit link costs. The link cost of each link is the inverse of its load multiplied by the largest link load in the network so that the smallest link cost is 1. Over all topologies this leads to an average link cost of 6.8 and a coefficient of variation of link costs of 1. Thus, the generated link costs differ substantially.

B. Protection Coverage

In this subsection we evaluate and compare the coverage of RoLPS protection variants. First, we explain the metric. Then, we briefly describe the evaluated protection mechanisms. Finally, we discuss results for networks with unit link costs and with non-unit link costs.

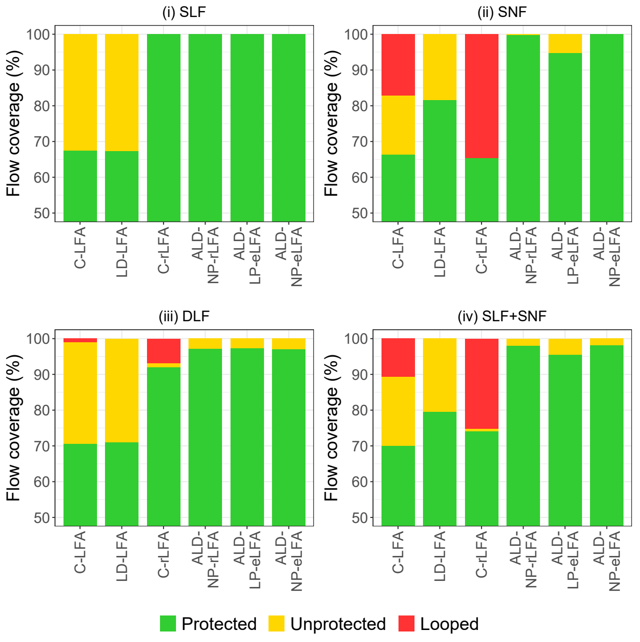
1) Metric: We introduce the three terms 'protected', 'unprotected', and 'looped' to refer to the quality of protection which is provided by a FRR mechanism for a flow in a specific scenario that consists of topology, failure scenario, and link costs. A flow is considered protected in two cases. First, if the packet is still successfully delivered at the destination although the path from source to destination was interrupted by a failure. Second, if a packet is dropped to prevent a loop because the destination is not reachable anymore. A flow is unprotected if the packet is dropped although the destination is still reachable. Finally, a flow is denoted as looped if a microloop was caused by local rerouting. We report the average fraction of protected, unprotected, and looped flows over all 208 topologies (see Section V-A2) in bar diagrams. The term coverage refers to the fraction of protected flows in a scenario.

2) Evaluated Protection Variants: We consider the classic protection variants C-LFA (nLD-LP-LFA) and C-rLFA (nLD-LP-rLFA) as well as the LD-LFA (ALD-NP-LFA) from [1]. We further study the new protection variants ALD-NP-rLFA and ALD-{LP,NP}-eLFA since they have stronger protection properties.

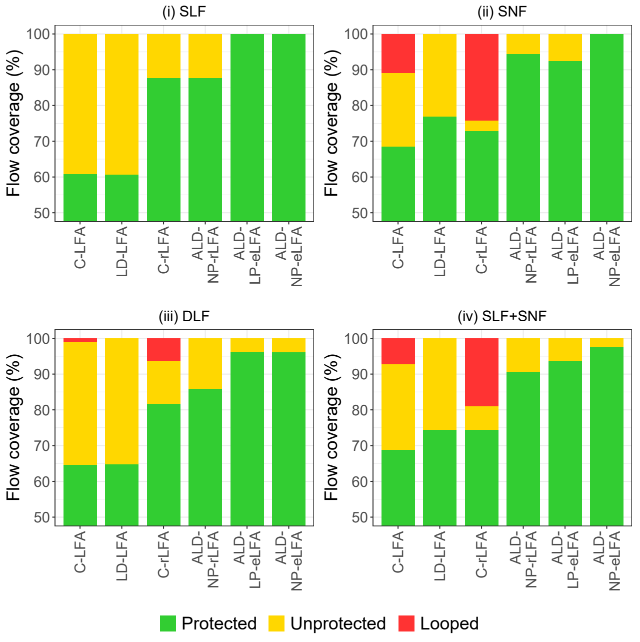
3) Coverage: In this section we present results for the number of protected destinations for different failure scenarios. First, we evaluate unit link cost networks. Then, we discuss non-unit link cost networks.

a) Networks with Unit Link Costs: Figure Figure 3(a) shows the coverage in percent for different sets of failure scenarios in networks with unit link costs. Subfigure 3(a) (i) shows that only C-LFA and LD-LFA cannot protect all destinations against SLF, i.e., their coverage is less than 100%. All other protection variants provide full coverage.

Subfigure 3(a) (ii) shows that SNFs cause many rerouting loops with C-LFA (17%) and C-rLFA (34%). This is mostly caused by failed destinations. As C-rLFA protect more destinations than C-LFA, they also cause more loops when the next-hop is the destination. Thus, loop detection is even more important when C-rLFA is used because more flows loop in case of node failures than with C-LFA. LD-LFA protects more



(a) Networks with unit link costs.



(b) Networks with non-unit link costs.

Figure 3: Coverage averaged over 208 topologies depending on protection method and set of failure scenarios.

traffic (81%) than C-(r)LFA in case of SNF as it preferentially uses NP-LFAs if available. Moreover, it prevents loops.

The new protection variants have significantly higher coverage. ALD-NP-rLFA protects around 99% of the destinations with SNF. This results from dropping packets that cannot be delivered anymore due to a failed destination; if they looped, the corresponding flow would count as looped. The coverage of ALD-LP-eLFA is slightly lower, i.e., 94%. This is because NP-(e/r)LFAs are not preferentially chosen for this protection variant so that there are more LFAs in use without the NP property. Finally, ALD-NP-eLFA protects all destinations for three reasons. First, it leverages rLFAs or eLFAs to provide protection for destinations that cannot be protected with LFAs. Second, it uses NP-(e/r)LFAs to protect against node failures and falls back to unsafe LP-(e/r)LFAs only when (e/r)LFAs with NP property are not available. Third, ALD detects and stops all loops that may be caused by LFAs with LP. This turns flows that cannot reach their destination into protected flows instead of looped flows.

Subfigure 3(a) (iii) shows the coverage against DLFs. No mechanism is able to protect all destinations. C-LFA and LD-LFA protect around 70% of the destinations. C-rLFA cover more flows (92%). However, protection variants without loop detection, i.e., C-LFA and C-rLFA, lead to loops. All newly proposed protection variants achieve roughly the same coverage, i.e., 96%, and prevent loops.

Finally, Subfigure 3(a) (iv) shows results for SLF+SNF. They are similar to the results of DLFs, but the fraction of rerouting loops caused by both C-LFA and C-rLFA is significantly higher. This is due to node failures which cause significant rerouting loops for protection variants without loop detection.

b) Networks with Non-Unit Link Costs: Figure Figure 3(b) shows the coverage for different sets of failure scenarios in networks with non-unit link costs. Subfigure 3(b) (i) shows the coverage against SLF. Both C-LFA and LD-LFA protect only around 60% of the destinations. In networks with non-unit link costs, C-rLFA cannot protect all destinations anymore against SLF and achieve only a coverage of 88%. The same holds for ALD-NP-rLFA. Only the eLFA-based protection variants are able to protect all destinations against SLF.

Subfigure 3(b) (ii) shows the coverage against SNF. Both C-LFA and C-rLFA cause many rerouting loops. LD-LFA prevents loops but protects only 76% of the destinations. ALD-NP-rLFA and ALD-LP-eLFA protect a higher fraction of destinations, i.e., 94% and 93%, because they prevent loops of unsafe LFAs with LP, but they have no suitable backup path for some node failures. ALD-NP-eLFA protects all destinations against SNF even in networks with non-unit link costs as it prevents loops and leverages NP-(e/r)LFAs wherever possible.

Finally, Subfigure 3(b) (iii) and Subfigure 3(b) (iv) present the coverage for DLF and SLF+SNF. The results are similar to those from networks with unit link costs, but the coverage here is slightly lower.

C. Additional Forwarding Entries

We now evaluate the number of additional forwarding entries to implement explicit tunnels. First, we explain the metric. Then, we discuss the investigated FRR mechanisms. Finally, we present results for networks with unit link costs and non-unit link costs.

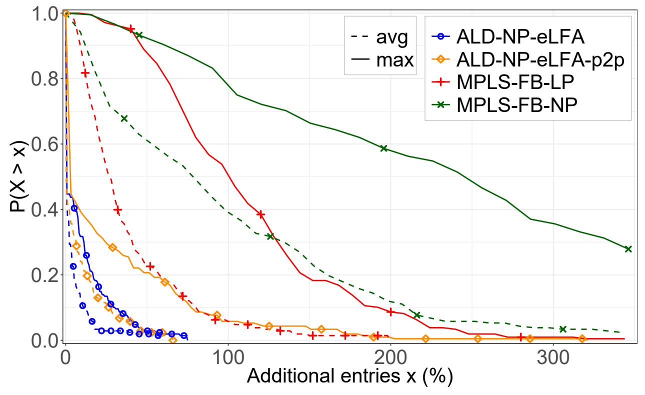
1) Metric: In a network with n nodes, each node maintains $n - 1$ forwarding entries for destination-based forwarding. eLFAs require additional forwarding entries to implement explicit tunnels. In contrast, both LFAs and rLFAs are based on shortest paths, and therefore, do not need additional forwarding entries. We calculate the average and maximum amount of additional forwarding entries per node relative to $n - 1$ for each network and present the results for all topologies in a CCDF.

2) FRR Mechanisms under Study: We compare the required amount of additional forwarding entries only for eLFA-based RoLPS protection variants as others do not require additional forwarding entries. To evaluate the efficiency of multipoint-to-point rerouting rerouting tunnels, we report results for ALD-{LP,NP}-eLFA and compare them to the corresponding mechanisms with point-to-point rerouting tunnels, i.e., ALD-{LP,NP}-eLFA-p2p. In addition, we present results for state-of-the-art MPLS-facility-backup (MPLS-FB-{LP,NP}) with LP and NP property.

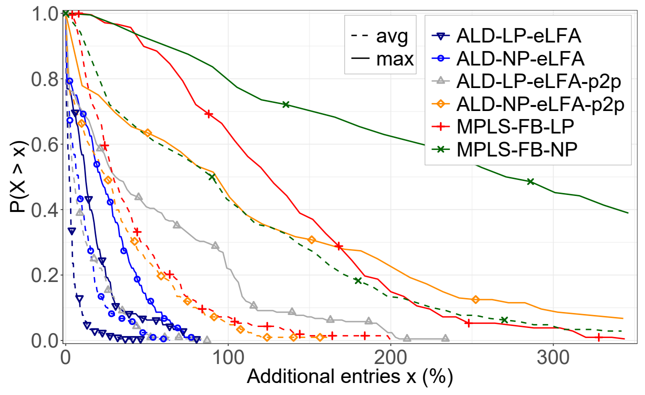
3) Results: We present results for the fraction of additional forwarding entries. First, we evaluate unit link cost networks. Then, we discuss non-unit link cost networks.

a) Networks with Unit Link Costs: Figure 4(a) shows a CCDF for the relative amount of additional forwarding entries for the considered FRR mechanisms in networks with non-unit link costs. First, we compare LP mechanisms. With MPLS-FB-LP, in 40% of the networks at least one node requires 120% or more additional entries (max-curve). However, on average in only 6% of the networks more than 100% additional entries are needed (avg-curve). The curves for ALD-LP-eLFA and ALD-LP-eLFA-p2p are omitted because those protection variants do not induce any additional forwarding entries. This is because (r)LFAs alone protect all destinations against all SLF in networks with unit link costs. Therefore, explicit LFAs are not needed and no additional forwarding entries are required.

Now, we compare NP mechanisms. MPLS-FB-NP requires most additional entries by far. 62% of the topologies have at least one node that requires 200% or more additional entries. And in 40% of the topologies 100% or more additional entries are required on average. Protection mechanisms with eLFAs, i.e., ALD-NP-eLFA and ALD-NP-eLFA-p2p, require less forwarding entries because they protect most of the destinations by NP-rLFAs and only the few remaining destinations are protected by eLFAs which induce forwarding state in the network. When ALD-NP-eLFA-p2p is used, only 20% of topologies have a node that requires 50% or more additional entries. However, some topologies contain at least one node that requires 200% or more additional entries. On average, no topology requires more than 65% or more additional entries. ALD-NP-eLFA is even more efficient because it leverages multipoint-to-point rerouting tunnels to reduce the number



(a) Networks with unit link costs. ALD-LP-eLFA does not induce any additional entries and is omitted in the figure.



(b) Networks with non-unit link costs.

Figure 4: CCDFs for fraction of additional forwarding entries.

of additional forwarding entries even further. There is no topology with a node that requires more than 70% of additional entries. 90% of the networks require only 15% or less additional entries on average.

b) Networks with Non-Unit Link Costs: Figure 4(b) shows a CCDF for the relative amount of additional forwarding entries for the considered FRR mechanisms in networks with non-unit link costs. Again, we compare LP mechanisms first. MPLS-FB-LP requires lots of additional entries. Around 55% of the topologies have at least one node that requires 120% or more additional entries (max-curve). However, in only 8% of the networks more than 100% additional entries are needed on average (avg-curve). eLFA-based protection mechanisms, i.e., ALD-LP-eLFA and ALD-LP-eLFA-p2p, are more efficient. When ALD-LP-eLFA-p2p is used, 22% of networks contain at least one node that requires 100% or more additional entries. On average, in 20% of networks nodes require 25% or more additional entries. ALD-LP-eLFA reduces the number of additional forwarding entries by leveraging multipoin-to-point tunnels. There is no topology with a node that requires more than 80% of additional entries and in 95% of the networks less than 15% additional entries are needed on average.

Now we compare NP mechanisms. MPLS-FB-NP requires most additional entries by far. 75% of networks have at least one node that requires 120% or more additional entries, 40% even more than 340%. In around 44% of the networks, 100% or more entries are required on average, and in 8% of the networks even 250% or more additional entries are required. ALD-NP-eLFA-p2p requires less entries. Only 45% of networks contain a node that requires 100% or more additional entries, but 20% of networks require even more than 210%. On average, 22% of networks require 50% or more additional entries. ALD-NP-eLFA is even more efficient. No network contains a node that requires more than 80% additional entries. In 90% of the networks, less than 30% additional entries are required on average.

Thus, in networks with non-unit link costs, somewhat more additional entries are needed but ALD-{LP,NP}-eLFA still require significantly less entries than MPLS-FB-{LP,NP} and ALD-{LP,NP}-eLFA-p2p.

D. *Size of Header Stacks for TI-LFAs*

In this section we evaluate the number of required segments to implement explicit paths with TI-LFAs using local labels. First, we explain the metric, and the studied mechanisms. Then, we present the results.

1) Metric: We count the number of header segments that are added to the packet by the respective mechanism for FRR purposes. That is, we do not count the header segment that identifies the original destination of the packet. For each network we record both the average and maximum number of additional header segments added to a packet and present the results as a CCDF.

2) Reroute Mechanisms under Study: We evaluate TI-LFAs that use only local labels (see Section III-C) because they implement explicit tunnels with multiple header segments. However, we leverage TI-LFAs only when there are no LFAs or rLFAs to protect a destination to avoid unnecessary additional header segments.

rLFAs, and eLFAs also leverage tunnels. However, both require only one additional header segment for tunneling which is why we omit those curves in the figure to facilitate readability. LFAs do not require additional header segments.

3) Results: Figure 5 shows the results for networks with non-unit link costs.

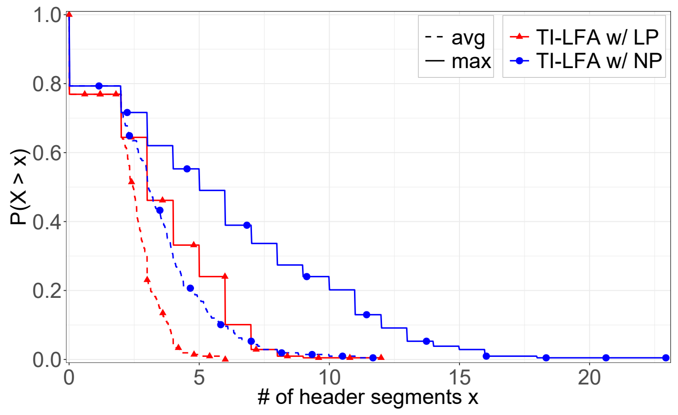


Figure 5: CCDF for number of additional header segments.

First, we discuss TI-LFAs with LP. In most networks, i.e., roughly 80%, TI-LFAs with LP require at least two additional header segments. In 20% of networks TI-LFAs with LP require on average 3 or more additional header segments. However, 23% of networks have at least one TI-LFA with LP that requires 6 or more additional header segments.

Now, we discuss TI-LFAs with NP. In general, TI-LFAs with NP require more additional header segments than TI-LFAs with LP. In 19% of networks TI-LFAs with NP require on average 5 or more additional header segments. 40% of networks even contain at least one TI-LFA with NP that requires 6 or more additional header segments.

In Section III-C2 we mentioned that the size of the TI-LFA header stack may be reduced. This, however, requires optimization which is a promising approach and an interesting research issue, but it is out of the scope of this document.

We omit a figure for results for networks with unit-link costs because of two reasons. First, TI-LFAs with NP require slightly fewer header segments but the results show no further insights. Second, for LP all destinations can be protected with either LFAs or rLFAs, i.e., no TI-LFAs are used, which was to be expected.

E. Path Lengths

In this section we report results for path lengths. First, we explain the metric and evaluated FRR mechanisms, then, we present the results.

1) Metric: We measure the path lengths of all flows that are affected by SLF but were successfully delivered due to local rerouting. For each topology, we calculate the average and maximum path lengths and present the results for all topologies in a CCDF.

2) Reroute Mechanisms under Study: We choose path lengths for rerouting as a baseline which recomputes shortest paths after a failure. We compare these results to the ones for ALD-{LP,NP}-eLFA and MPLS-FB-{LP,NP}.

3) Results: Figure 6 shows a CCDF for average and maximum path lengths of successfully delivered flows with SLF in networks with unit link costs.

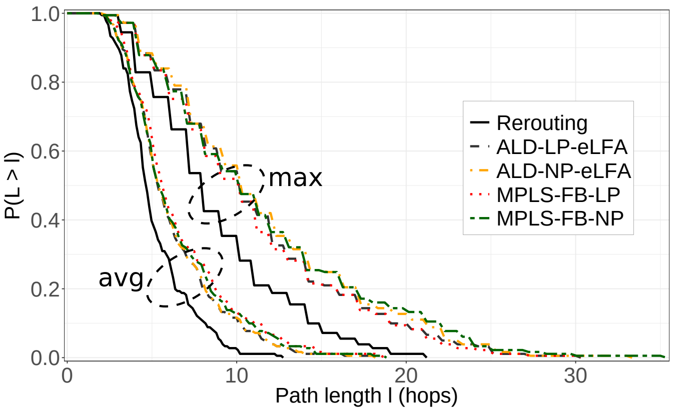


Figure 6: CCDF for path lengths of successfully delivered flows for SLF in networks with unit link costs.

We observe that rerouting leads in fact to the shortest maximum and average path lengths. All FRR mechanisms under study lead to longer maximum and average path lengths. The path lengths of the different FRR mechanisms does not differ.

The same analysis in networks with non-unit link costs leads to slightly longer paths but without any further insights. Therefore, we omit the corresponding figure.

F. Discussion

We investigated various RoLPS protection variants with regard to protection coverage, additional forwarding entries, and path lengths on a set of 208 topologies with both unit link costs and non-unit link costs, and compared them with MPLS-facility-backup.

The evaluations of protection coverage showed that C-LFA cannot protect many destinations in case of link failures. C-rLFAs can protect all destinations in case of SLF in networks with unit link costs. However, the usage of C-(r)LFA leads to many loops in case of node failures. The use of ALD avoids such loops. LD-LFA [1] prevents loops but cannot protect all destinations. ALD-NP-eLFA protects all destinations against SLF and SNF in networks with unit and non-unit link costs because it leverages eLFAs to complement (r)LFAs.

The explicit LFAs induce additional forwarding entries in the data plane, which is not desired. Therefore, we compared the additional forwarding entries for ALD-{LP,NP}-eLFA, ALD-{LP,NP}-eLFA-p2p, and MPLS-FB-{LP,NP}. ALD-{LP,NP}-eLFA require only very few additional entries compared to ALD-{LP,NP}-eLFA-p2p, and MPLS facility backup. Both MRCs [14] and IDAGs [18] always require 100% additional entries, and MRTs [15] need 200% more. Not-via addresses [11] need $100\% \cdot d$ more entries where d is the average node degree. Although TI-LFAs require at most d additional forwarding entries per node, they impose significant overhead in form of multiple additional header segments. ALD-{LP,NP}-eLFA add only one additional packet header for tunneling and our evaluation shows that they require less additional forwarding entries than other comparable FRR mechanisms. Therefore, ALD-{LP,NP}-eLFA can be considered very lightweight which makes them attractive for FRR in SDN.

All evaluated FRR mechanisms, i.e., ALD-{LP,NP}-eLFA and MPLS-FB-{LP,NP} extend backup paths by about the same, and backup paths are only slightly longer than the average and maximum length of recomputed shortest paths.

VI. Implementation of RoLPS in P4

We start with a short introduction of P4 and the implementation platform. Then we summarize important basics of P4 and describe the implementation of the RoLPS prototype.

A. Overview of P4 and the Implementation Target

P4 is a high-level programming language for protocol-independent packet processors [51]. P4 programs are mapped, i.e., compiled, to the programmable processing pipeline of so-called targets, e.g., the software switch BMv2 [52] or the switching ASIC Tofino [53]. When a P4 program is successfully compiled for a target, it offers an API to let the control plane configure the device during runtime, e.g., to write forwarding entries.

In [2] we sketched how the predecessor of RoLPS could be implemented in OpenFlow. However, due to technical restrictions of OpenFlow the implementation concept required multiple workarounds which made it complex (see Section III-B1 and Section IV-C1). P4 offers significantly more flexibility than OpenFlow. It allows a flexible description of the data plane, in particular, the definition of arbitrary packet headers and packet parsers, and conditional application of programmable match+action tables (MATs). Therefore, implementation of novel features in P4 is easier than in OpenFlow.

In this paper we describe the implementation of RoLPS in P4. Our target is the P4-programmable high-performance switching ASIC Tofino [53] which is used in the Edgecore Wedge 100BF-32X [54] switch with 32 100 Gb/s ports. We made the source code for the RoLPS data plane and control plane publicly available[2].

B. P4 Pipeline

Figure 7 illustrates the abstract forwarding model of P4. A user-programmable parser extracts the information from the packet header and stores them in so-called header fields. They are carried with the packet through the processing pipeline, possibly with additional metadata which are similar to regular variables from other high-level programming languages. Metadata are packet-specific and discarded after the packet is sent to an egress port.

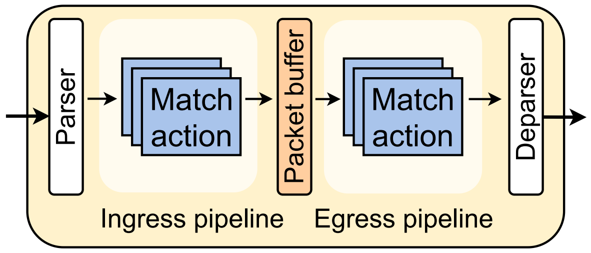


Figure 7: P4 abstract forwarding model according to [51].

The P4 abstract forwarding model is divided into two stages, the ingress and the egress pipeline, which are separated by a packet buffer. Match+action tables (MATs) allow for packet-specific processing. They have entries consisting of custom match fields and types that map header fields and metadata to actions, e.g., modifying header fields, and parameters.

P4 offers three match types: exact, longest-prefix match (LPM), and ternary. For an exact match the header field or metadata field must be exactly the same as the match field in the MAT, e.g., a specific IP address. LPM is well-known from standard IP forwarding. Ternary facilitates wildcard matches. P4 does not allow to match a packet multiple times on the same MAT to prevent processing loops.

[2]https://github.com/uni-tue-kn/p4-lfa

After the egress pipeline, the deparser writes the potentially modified header fields into the packet header and the packet is sent through the specified egress port.

However, P4 does not support FRR natively. Port status information cannot be accessed by the data plane by default. This makes the implementation of FRR in P4 a serious challenge.

C. Implementation of LFAs

First, we describe how the port status can be determined in P4. Afterwards, we describe the implementation of LFAs without tunnels followed by LFAs with tunnels, i.e., rLFAs and eLFAs, and ranking-based selection of LFA types.

1) Port Status Detection in P4: Executing backup actions, e.g., forwarding to an LFA, requires a reliable and timely detection when a port goes down. However, P4 does not support such a feature. In [55] we proposed a workaround for the Tofino platform which detects port-down events within 1 ms without controller interaction. We leverage this workaround to implement RoLPS-based protection and summarize it in the following.

Registers in P4 provide persistent storage, i.e., their content survives processed packets. The individual register fields can be accessed by an index. We leverage a register to store the current status of the egress ports by single bits (0: down, 1: up). Each register field stores the status of one port, i.e., one bit. The port ID serves as an index to access the corresponding register field. The challenge is updating the registers when the port status changes, which is platform-specific.

Port-down events are tracked as follows. Tofino has means outside the P4 programmable data plane to detect port-down events. We configured the Tofino such that it creates a 'port-down packet' in case of a port-down event. The packet contains the ID of the corresponding port and the packet is sent to a switch-intern port. We programmed the p4 pipeline such that the port status register for the respective port is set to zero upon reception of a port-down packet.

Port-up events are tracked differently. When the Tofino receives a packet over a specific port, it activates the status bit of that port in the register. To ensure that port-up events are detected sufficiently fast, we take advantage of topology packets that are regularly sent by the Tofino to all egress ports for neighbor detection. The frequency for topology packets can be configured to an appropriate value. While the detection of port-down events is time-critical, detection of port-up events is more relaxed because FRR mechanisms reroute affected traffic in the meantime via alternative ports.

2) Implementation of LFAs without Tunnels: As described in the previous section, the register fields provide information whether specific egress ports are up or down. However, the egress port of a packet is known only after matching the packet on a MAT. To mitigate this problem, we implemented FRR as shown in Figure 8. First, the packet is matched against a MAT that performs regular IPv4 routing, i.e., it determines the next-hop and thereby the egress port of a packet. Second, the ID of the selected egress port is used to access the register fields to retrieve the port status of that egress port. If the egress port is

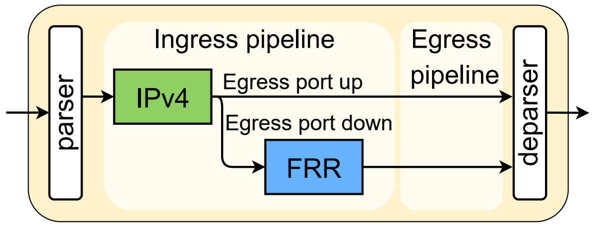


Figure 8: P4 implementation of FRR. A packet is matched against an IPv4 forwarding MAT to determine its egress port. If that port is down, the packet is matched against a FRR-MAT to determine its backup egress port.

up, the packet is forwarded. If the port is down, FRR actions are triggered, i.e., the packet is matched against a FRR-MAT using the IP destination address and the ID of the failed egress port. This selects a backup entry with a preinstalled LFA, i.e., backup egress port, for forwarding.

3) LFAs with Tunnels: LFAs with tunnels are implemented in a similar way as LFAs without tunnels. However, the backup actions in the FRR-MAT contain an encapsulation action which adds an additional IP header to the packet for tunneling to the remote node, i.e., the rLFA or eLFA.

If the remote node is an rLFA, the encapsulating IP header contains the IP address of that node. The packet is then forwarded on standard paths towards the rLFA.

If the remote node is an eLFA, the encapsulating IP header contains a unique IP address which identifies the explicit path towards the eLFA (see Section IV-B2). When the controller installs eLFAs in the network, it also sets up explicit tunnels towards the eLFAs. To that end, it calculates appropriate tunnel-specific forwarding entries and configures them on the forwarding devices along the explicit path. Thereby, the controller leverages explicit multipoint-to-point rerouting tunnels (see Section IV-B3) if possible to reduce the number of additional forwarding entries. That is, it configures only one additional forwarding entry on forwarding devices on overlapping subpaths of explicit paths towards the same eLFA.

4) Implementation of Ranking-Based Selection of LFA Types: The ranking-based selection of LFAs as described in Section IV-D is part of the control plane. The controller precomputes appropriate LFA types depending on the desired protection variant and installs corresponding egress ports and encapsulation actions in the FRR-MATs of the data plane devices.

D. Implementation of ALD

We implement ALD so that it allows two redirects, i.e., the packet is dropped when it has to be rerouted a third time. To that end, we define the ALD field as a 2-bit custom header field in the packet header. These bits track how often a packet has been rerouted. Packets initially carry the bit pattern ‘00’ in the ALD field. When a node reroutes a packet with bit pattern ‘00’, it replaces the bit pattern with ‘01’. When a node reroutes a packet with bit pattern ‘01’, it replaces the bit pattern with ‘10’. When a node cannot forward a packet with bit pattern ‘10’ due to a failed egress port, it drops the packet.

VII. Hardware-Based Performance Evaluation

In this section we conduct a performance evaluation of the RoLPS hardware prototype. It is based on the Tofino [53], a P4-programmable switch ASIC, which is used in the Edgecore Wedge 100BF-32X [54], a switch with 32 100 Gb/s ports. We present measurement results for throughput, restoration time, and loop detection.

A. Throughput

Every P4 program successfully compiled for the Tofino processes packets at a speed of 100 Gb/s. To verify that property for our prototype, we conducted the following experiment. We utilized an EXFO FTB-1 Pro traffic generator [56] which generates up to 100 Gb/s of traffic. We connected it to the Tofino which processes the traffic and sends it back to the traffic generator. This way we measure the traffic rate forwarded by Tofino. In fact, we obtained a throughput of 100 Gb/s for both failure-free forwarding and forwarding with activated FRR.

B. Restoration Time

The evaluation of restoration times is more complex. We describe the testbed, the measurement procedure and metric, as well as the experimental scenarios. Then, we present measurement results.

1) Testbed: Figure 9 shows the testbed for the performance evaluation. Center of the testbed is the above mentioned Tofino.

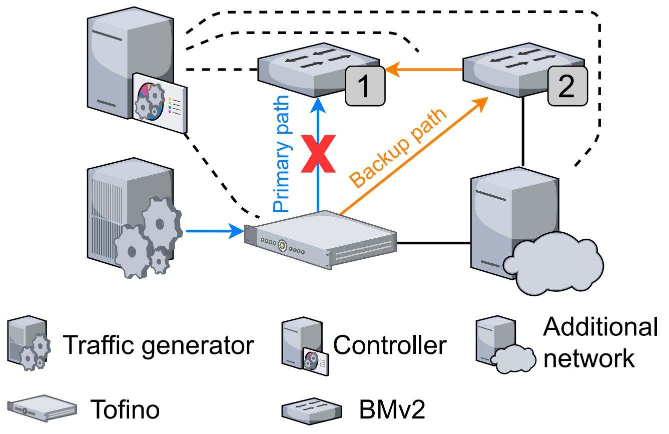


Figure 9: Topology for restoration time measurements. The additional network consists of five other BMv2s and 10 links.

It is connected to two BMv2 [52] P4 software switches. To perform evaluations for more realistic network sizes, we connected the Tofino to an additional network which consists of five BMv2s and 10 links. All BMv2s run on a server with an Intel Xeon Gold 6134 with 3.2 GHz and 12 cores, and 32 GB RAM. A controller is connected to the Tofino and all BMv2s. It configures them upon start, i.e., it discovers the topology, and computes and installs appropriate forwarding rules. It runs on the same server as the BMv2s. Furthermore, the above mentioned traffic generator is connected to the Tofino and serves as a traffic source in the experiment.

2) Measurement Procedure and Metric: The traffic generator sends traffic to the Tofino which forwards the packets on the primary path to the destination BMv2-1. BMv2-1 monitors the packet arrivals. Then, we deactivate the link from Tofino to BMv2-1 on the primary path to trigger a port-down event at the Tofino. We derive the restoration time for the FRR mechanism from a tcpdump log at BMv2-1. It is the duration of the interval within which BMv2-1 does not receive any packets.

In these experiments, the traffic generator sends only with 100 Mb/s instead of 100 Gb/s. This avoids overload on the BMv2s which can process packets only with around 900 Mb/s [57]. Avoiding overload is important only to obtain correct measurement results from BMv2-1. The restoration time on the Tofino is not affected by any overload.

3) Experiments: We perform two experiments to measure the restoration time without and with FRR.

a) Forwarding without FRR: For this experiment we disabled the FRR feature on Tofino. When the Tofino detects the failure, it notifies the controller. The controller then updates its topology, computes new forwarding entries, and installs them on the affected devices so that traffic can be forwarded again.

b) Forwarding with FRR: In this experiment the FRR feature is enabled. Thus, if BMv2-1 is no longer reachable, the Tofino forwards traffic destined to BMv2-1 to BMv2-2 which relays the traffic to BMv2-1.

4) Results: We performed the above described experiments 10 times. Figure 10 shows the average restoration time without and with FRR on the Tofino, including 95% confidence intervals.

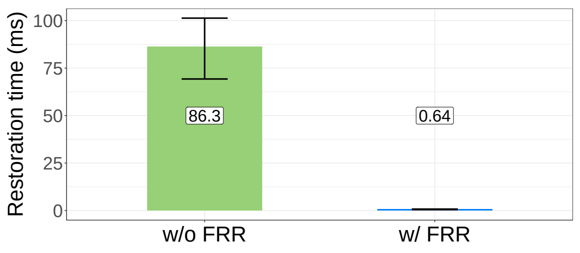


Figure 10: Restoration time on Tofino without and with FRR.

If FRR is disabled, traffic is delivered again after 86 ms. As rerouting without FRR requires controller interaction, the measured restoration time depends on controller load, network size, and communication delay. In this experiment, there is only a single flow affected by the faiure, the overall network is small despite the additional network, and the controller is directly connected to the Tofino. Therefore, the experimental result for the restoration time is likely lower than restoration times in production networks.

If FRR is enabled, traffic is delivered after a small restoration time of 0.6 ms. Here, the switchover from primary egress port to backup egress port at the Tofino is independent of controller load, network size, and communication delay as FRR is a switch-local mechanism. Thus, restoration times can be greatly reduced by FRR on P4-capable hardware. Moreover, the mechanism is general enough to support all RoLPS protection variants by appropriate configuration through the controller.

C. Loop Detection

We experimentally evaluate the capability of ALD to detect and stop loops. We present the modified testbed, explain two different experiments and the studied metric, and finally we discuss measurement results.

1) Testbed: Figure 11 shows the testbed. The Tofino is

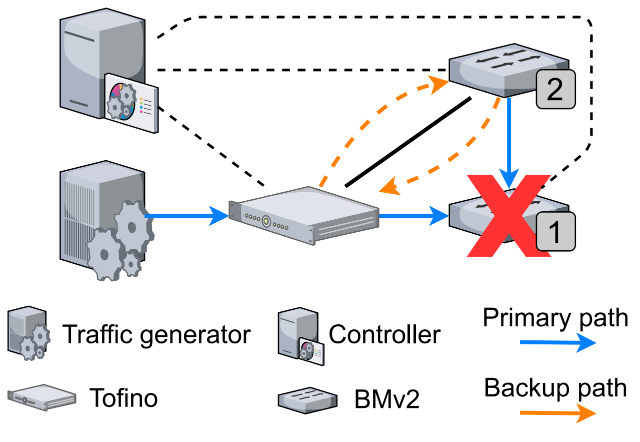


Figure 11: Testbed for evaluation of ALD.

now conntected to two BMv2s (BMv2-1, BMv2-2) which are also connected with each other. The controller configures the Tofino and all BMv2s with available LP-LFAs upon startup. In the experiments, the traffic generator sends a packet towards BMv2-1. The Tofino has BMv2-2 as an LFA when BMv2-1 is not reachable. Likewise, BMv2-2 has the Tofino as an LFA when BMv2-1 is not reachable. If BMv2-1 fails, traffic destined to that node loops between the Tofino and BMv2-2. However, the TTL in the IP header is set to 64 when sent by the traffic generator and decremented whenever forwarded by a node. The packet is dropped when its TTL reached 0.

2) Experiments and Metric: We perform two experiments with ALD disabled and ALD enabled on the switches. We track packet arrivals at BMv2-2 using tcpdump. Thereby we can observe how often a looping packet is received.

3) Results: Figure 12 illustrates a log of packet arrivals at BMv2-2, starting with time 0 at first packet arrival. Without

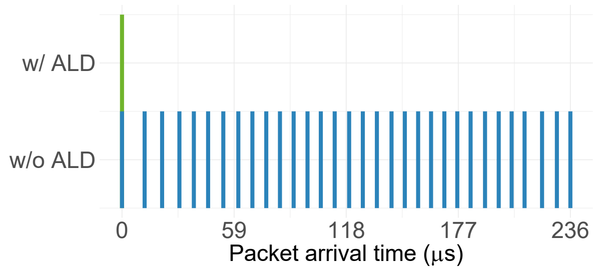


Figure 12: Packet arrivals at BMv2-2 without and with ALD.

ALD, BMv2-2 receives the packet 32 times. Thus, the packet looped between the Tofino and BMv2-2 until it was dropped due to TTL=0. With ALD, BMv2-2 receives the packet only once. It then redirects the packet to the Tofino which then drops the packet at the attempt to reroute the packet for the third time. Therefore, BMv2-2 receives the packet only once.

VIII. Conclusion

In this paper we presented robust LFA protection for software-defined networks (RoLPS). It leverages loop-free alternates (LFAs) and remote LFAs (rLFAs) known from IP networks to forward traffic over alternative next-hops if primary next-hops are not reachable. However, this alone cannot protect all destinations against failures and may cause forwarding loops under challenging conditions. Therefore, we proposed explicit LFAs (eLFAs) using explicit rerouting tunnels to cover all destinations. eLFAs are conceptually similar to topology-independent LFAs (TI-LFAs) but do require only a single additional header segment for protection while protection with typical TI-LFAs may require a clearly larger header stack. Furthermore we describe advanced loop detection (ALD) to stop forwarding loops. These mechanisms are simple and do not require controller interaction. We suggested various protection variants that utilize (e/r)LFAs with different protection quality and complexity.

We evaluated RoLPS through simulations based on 208 representative topologies. The results revealed that existing (r)LFAs cannot provide all destinations and lead to substantial forwarding loops in case of node failures. More elaborate RoLPS variants with eLFAs and ALD, e.g., ALD-NP-eLFA, protect all traffic against all single link or node failures in networks with both unit and non-unit link costs. Furthermore, they protect most destinations against multiple failures ($> 90\%$) and prevent forwarding loops. A drawback of eLFAs is that they required additional forwarding entries. However, our evaluation showed that RoLPS protection variants require only very few eLFAs, in particular compared to other FRR mechanisms such as MPLS facility backup, MRTs, MRCs, IDAGs, or not-via addresses. Thus, the full protection coverage against single link or node failures together with the need for only a few additional forwarding entries make RoLPS attractive for software-defined networks. In addition, RoLPS protection variants extends lengths of backup paths compared to those of shortest path recomputation, but there is no visible difference to backup path lengths with MPLS facility backup.

We implemented a P4-based prototype that features RoLPS-based protection variants. The source code is publicly available. A measurement study showed that the prototype achieves a throughput of 100 Gb/s, restores connectivity in less than 1 ms including failure detection, and reliably detects and stops forwarding loops.

Acknowledgment

We acknowledge the support from BelWü for borrowed high-performance hardware for the measurement-based evaluations. Likewise, we appreciate the work of Irene Müller-Benz for the development of an early prototype of RoLPS.

Acronyms and Glossary

FRR	fast reroute
PLR	point of local repair
LFA	loop-free alternate [20]
rLFA	remote LFA [23], [24]
eLFA	explicit LFA [2]
TI-LFA	topology-independent LFA [3]
MPLS	multiprotocol label switching [8]
MRT	maximally redundant tree [15]
IDAG	independent directed acyclic graph [18]
MRC	multiple routing configuration [14]
SLF	single link failure
SNF	single node failure
DLF	double link failure
LP	link protecting
NP	node protecting
ALD	advanced loop detection
RoLPS	robust LFA protection for SDN

Table 3: Acronyms.

Point of local repair (PLR)	A node that cannot forward a packet to the default next-hop because of a failure. It executes precomputed backup actions to locally reroute packets around the failure.
Loop-free alternate (LFA)	Alternative next-hop that successfully forwards failure-affected traffic towards the destination. Simple LFAs cannot protect all destinations.
rLFA	Remote nodes in the network that successfully forward traffic towards the destination. PLRs reach rLFAs through shortest path tunnels. rLFAs protect more destinations than LFAs. However, they cannot protect all destinations against SLF in non-unit link cost networks or SNF in general.
eLFA	Similar to rLFAs. However, PLRs reach eLFAs through explicit tunnels implemented by additional forwarding entries. eLFAs protect against all SLF and SNF independent of link costs. Multipoint-to-point tunnels reduce the number of additional forwarding entries.
Link protecting (LP)	A link protecting (e/r)LFA avoids the link between PLR and next-hop. They may cause rerouting loops for SNF.
Node protecting (NP)	A node protecting (e/r)LFA avoids the next-hop. There are significantly less NP-(e/r)LFAs than LP-(e/r)LFAs. NP implies LP, i.e., it is the stronger property.
Loop detection (LD) [1]	A mechanism to detect and stop rerouting loops caused by LFAs. May erroneously drop packets.
LD-LFA [1]	LD-LFA preferably uses NP-LFAs for protection. Only when no NP-LFA is available, LP-LFAs are used to increase the number of protected destinations. In addition, LD-LFA leverages loop detection to prevent loops.
Advanced loop detection (ALD)	A mechanism to detect and stop loops caused by LFAs. Allows to reroute a packet two times to cope with double failures.
Robust LFA protection for SDN (RoLPS)	Protection concept presented in this paper. It defines eLFAs and ALD. RoLPS ranks (e/r)LFAs and selects the best one. Uses ALD to detect and stop loops.

Table 4: Glossary.

References

[1] W. Braun and M. Menth, "Loop-Free Alternates with Loop Detection for Fast Reroute in Software-Defined Carrier and Data Center Networks," *Journal of Network and Systems Management*, vol. 24, 2016.

[2] D. Merling, W. Braun, and M. Menth, "Efficient Data Plane Protection for SDN," in *IEEE Conference on Network Softwarization (NetSoft)*, Jun. 2018.

[3] P. Francois, C. Filsfils, A. Bashandy, B. Decraene, and S. Litkowski, *Topology Independent Fast Reroute using Segment Routing*, https://tools.ietf.org/html/draft-ietf-rtgwg-segment-routing-ti-lfa-06, Feb. 2021.

[4] S. Rai, B. Mukherjee, and O. Deshpande, "IP Resilience within an Autonomous System: Current Approaches, Challenges, and Future Directions," *IEEE Communications Magazine*, vol. 43, 2005.

[5] A. Raj and O. Ibe, "A Survey of IP and Multiprotocol Label Switching Fast Reroute Schemes," *Computer Networks*, vol. 51, no. 8, 2007.

[6] J. Papan, P. Segeč, P. Palúch, and L. Mikus, "The Survey of Current IPFRR Mechanisms," in *Federated Conference on Software Development and Object Technologies*, Dec. 2017.

[7] D. Hutchison and J. P. Sterbenz, "Architecture and design for resilient networked systems," *Computer Communications*, vol. 131, 2018.

[8] E. Rosen, A. Viswanathan, and R. Callon, *Multiprotocol Label Switching Architecture*, https://tools.ietf.org/html/rfc3031, Jan. 2001.

[9] Ping Pan and George Swallow and Alia Atlas, *RFC4090: Fast Reroute Extensions to RSVP-TE for LSP Tunnels*, https://tools.ietf.org/html/rfc4090, May 2005.

[10] K. Kompella and W. Lin, *No Further Fast Reroute*, https://tools.ietf.org/html/draft-kompella-mpls-nffrr-00, Mar. 2020.

[11] S. Bryant, S. Previdi, and M. Shand, *RFC6981: A Framework for IP and MPLS Fast Reroute Using Not-Via Addresses*, http://www.rfc-editor.org/rfc/rfc6981.txt, Jul. 2013.

[12] R. Martin, M. Menth, M. Hartmann, T. Cicic, and A. Kvalbein, "Loop-Free Alternates and Not-Via Addresses: A Proper Combination for IP Fast Reroute?" *Computer Networks*, vol. 54, 2010.

[13] S. Nelakuditi *et al.*, "Fast Local Rerouting for Handling Transient Link Failures," *IEEE/ACM Trans. on Networking*, Apr. 2007.

[14] A. Kvalbein, A. F. Hansen, T. Cicic, S. Gjessing, and O. Lysne, "Fast IP Network Recovery Using Multiple Routing Configurations," in *IEEE Infocom*, Apr. 2006.

[15] A. Atlas, C. Bowers, and G. Enyedi, *RFC7812: An Architecture for IP/LDP Fast Reroute Using Maximally Redundant Trees (MRT-FRR)*, http://www.rfc-editor.org/rfc/rfc7812.txt, Jun. 2016.

[16] M. Menth and W. Braun, "Performance Comparison of Not-Via Addresses and Maximally Redundant Trees (MRTs)," in *IEEE/IFIP IM*, Apr. 2013.

[17] K. Kuang, S. Wang, and X. Wang, "Discussion on the Combination of Loop-Free Alternates and Maximally Redundant Trees for IP Networks Fast Reroute," in *IEEE International Conference on Communications*, Jun. 2014.

[18] S. Cho, T. Elhourani, and S. Ramasubramanian, "Independent Directed Acyclic Graphs for Resilient Multipath Routing," *IEEE/ACM Transactions on Networking*, vol. 20, Feb. 2012.

[19] S. S. Lor, R. Landa, and M. Rio, "Packet re-cycling: Eliminating packet losses due to network failures," in *ACM Workshop on Hot Topics in Networks*, 2010.

[20] A. Atlas and A. Zinin, *RFC5286: Basic Specification for IP Fast Reroute: Loop-Free Alternates*, http://www.rfc-editor.org/rfc/rfc5286.txt, 2008.

[21] L. Csikor, M. Nagy, and G. Rétvári, "Network Optimization Techniques for Improving Fast IP-level Resilience with Loop-Free Alternates," *Infocommunications Journal*, vol. 3, 2011.

[22] L. Csikor, J. Tapolcai, and G. Retvari, "Optimizing IGP link costs for improving IP-level resilience with Loop-Free Alternates," *Computer Communications*, vol. 36, 2013.

[23] S. Bryant, C. Filsfils, S. Previdi, M. Shand, and N. So, *RFC7490:Remote Loop-Free Alternate (LFA) Fast Reroute (FRR)*, https://tools.ietf.org/html/rfc7490, 2015.

[24] P. Sarkar, S. Hegde, C. Bowers, H. Gredler, and S. Litkowski, *Remote-LFA Node Protection and Manageability*, https://tools.ietf.org/html/rfc8102, 2017.

[25] L. Csikor and G. Retvari, "On Providing Fast Protection with Remote Loop-Free Alternates: Analyzing and Optimizing Unit Cost Networks," in *Telecommunication Systems*, 2015.

[26] G. Retvari, J. Tapolcai, G. Enyedi, and A. Csaszar, "IP Fast ReRoute: Loop Free Alternates Revisited," in *IEEE Infocom*, Apr. 2011.

[27] W. Tavernier, D. Papadimitriou, D. Colle, M. Pickavet, and P. Demeester, "Self-configuring Loop-free Alternates with High Link Failure Coverage," *Telecommunication Systems*, vol. 56, 2014.

[28] A. Farrel and R. Bonica, "Segment Routing: Cutting Through the Hype and Finding the IETF's Innovative Nugget of Gold," *IETF Journal*, vol. 13, 2017.

[29] Y. E. Oktian *et al.*, "Distributed SDN Controller System: A Survey on Design Choice," *Computer Networks*, vol. 121, 2017.

[30] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "OpenFlow: Meeting Carrier-Grade Recovery Requirements," *Computer Communications*, vol. 36, 2013.

[31] A. S. da Silva, P. Smith, A. Mauthe, and A. Schaeffer-Filho, "Resilience support in software-defined networking: A survey," *Computer Networks*, vol. 92, 2015.

[32] M. Chiesa, A. Kamisiński, J. Rak, G. Rétvári, and S. Schmid, *A Survey of Fast Recovery Mechanisms in the Data Plane*, https://www.techrxiv.org/articles/preprint/Fast_Recovery_Mechanisms_in_the_Data_Plane/12367508/2, May 2020.

[33] J. Kempf, E. Bellagamba, A. Kern, D. Jocha, A. Takàcs, and P. Sköldström, "Scalable Fault Management for OpenFlow," in *IEEE International Conference on Communications*, 2012.

[34] N. L. van Adrichem, B. J. van Asten, and F. A. Kuipers, "Fast Recovery in Software-Defined Networks," in *European Workshop on Software Defined Networks*, Sep. 2014.

[35] R. M. Ramos *et al.*, "SlickFlow: Resilient Source Routing in Data Center Networks Unlocked by OpenFlow," in *IEEE Conference on Local Computer Networks*, Oct. 2013.

[36] C. Cascone, L. Pollini, D. Sanvito, A. Capone, and B. Sansó, "SPIDER: Fault Resilient SDN Pipeline with Recovery Delay Guarantees," in *IEEE Conference on Network Softwarization*, Jun. 2016.

[37] N. L. M. van Adrichem, F. Iqbal, and F. A. Kuipers, "Backup Rules in Software-Defined Networks," in *IEEE Conference on Network Function Virtualization and Software-Defined Networking*, Nov. 2016.

[38] S. Cevher, M. Ulutas, S. Altun, and I. Hokelek, "Multi Topology Routing Based IP Fast Re-Route for Software Defined Networks," in *IEEE Symposium on Computers and Communications*, Jun. 2016.

[39] S. Cevher, "Multi Topology Routing Based Failure Protection for Software Defined Networks," in *IEEE International Black Sea Conference on Communications and Networking*, Jun. 2018.

[40] Q. Li, Y. Liu, Z. Zhu, H. Li, and Y. Jiang, "BOND: Flexible failure recovery in software defined networks," *Computer Networks*, vol. 149, 2019.

[41] R. Sedar, M. Borokhovich, M. Chiesa, G. Antichi, and S. Schmid, "Supporting Emerging Applications With Low-Latency Failover in P4," in *Workshop on Networking for Emerging Applications and Technologies*, 2018.

[42] H. Giesen, L. Shi, J. Sonchack, A. Chelluri, N. Prabhu, N. Sultana, L. Kant, A. J. McAuley, A. Poylisher, A. DeHon, and B. T. Loo, "In-Network Computing to the Rescue of Faulty Links," in *Morning Workshop on In-Network Computing*, 2018.

[43] S. Lindner, D. Merling, M. Häberle, and M. Menth, "P4-Protect: 1+1 Path Protection for P4," *P4 Workshop in Europe (EuroP4)*, Dec. 2020.

[44] K. Hirata and T. Tachibana, "Implementation of Multiple Routing Configurations on Software-Defined Networks with P4," in *Asia-Pacific Signal and Information Processing Association Annual Summit and Conference*, 2019.

[45] K. Subramanian, A. Abhashkumar, L. D'Antoni, and A. Akella, *D2R: Dataplane-Only Policy-Compliant Routing Under Failures*, 2019.

[46] M. Chiesa, R. Sedar, G. Antichi, M. Borokhovich, A. Kamisiński, G. Nikolaidis, and S. Schmid, "PURR: A Primitive for Reconfigurable Fast Reroute," in *ACM Conference on emerging Networking EXperiments and Technologies*, 2019.

[47] C. Filsfils *et al.*, *RFC8986: Segment Routing over IPv6 (SRv6) Network Programming*, https://www.rfc-editor.org/rfc/rfc8986.txt, 2021.

[48] A. Bashandy *et al.*, *RFC8660: Segment Routing with the MPLS Data Plane*, https://www.rfc-editor.org/rfc/rfc8660.txt, 2019.

[49] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The Internet Topology Zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, 2011.

[50] S. Halabi, *OSPF DESIGN GUIDE*, http://rtfm.vtt.net/spf1euro.pdf, 1996.

[51] P. Bosshart *et al.*, "P4: Programming Protocol-Independent Packet Processors," *ACM CCR*, vol. 44, 2014.

[52] p4lang, *Behavioral-model*, https://github.com/p4lang/behavioral-model, 2019.

[53] Edge-Core Networks, *The World's Fastest & Most Programmable Networks*, https://barefootnetworks.com/resources/worlds-fastest-most-programmable-networks/, 2017.

[54] ——, *Wedge100BF-32X/65X Switch*, https://www.edge-core.com/_upload/images/Wedge100BF-32X_65X_DS_R05_20191210.pdf, 2019.

[55] D. Merling, S. Lindner, and M. Menth, "Hardware-Based Evaluation of Scalable andResilient Multicast with BIER in P4," *IEEE Transactions on Network and Service Management*, In Revision for TNSM special issue: Advanced Management of Softwarized Networks.

[56] EXFO, *FTB-1v2/FTB-1 Pro Platform*, https://www.exfo.com/umbraco/surface/file/download/?ni=10900&cn=en-US&pi=5404, 2019.

[57] A. Bas, *BMv2 Throughput*, https://github.com/p4lang/behavioral-model/issues/537#issuecomment-360537441, Jan. 2018.

Daniel Merling is a Ph. D. student at the chair of communication networks of Prof. Dr. habil. Michael Menth at the Eberhard Karls University Tuebingen, Germany. There he obtained his master's degree in 2017 and afterwards, became part of the communication networks research group. His area of expertise include software-defined networking, scalability, P4, routing and resilience issues, multicast and congestion management.

Steffen Lindner is a Ph.D. student at the Eberhard Karls University Tübingen, Germany. He wrote his bachelor and master thesis at the chair of communication networks of Prof. Dr. habil. Michael Menth. He started his Ph.D. in September 2019 at the communication networks research group. His research interests include software-defined networking, P4 and congestion management.

Michael Menth (Senior Member, IEEE) is professor at the Department of Computer Science at the University of Tuebingen/Germany since 2010 and chairholder of Communication Networks. He studied, worked, and obtained diploma (1998), PhD (2004), and habilitation (2010) degrees at the universities of Austin/Texas, Ulm/Germany, and Wuerzburg/Germany. His special interests are performance analysis and optimization of communication networks, resilience and routing issues, resource and congestion management, industrial networking and Internet of Things, software-defined networking and Internet protocols.