Loop-Free Alternates with Loop Detection for Fast Reroute in Software-Defined Carrier and Data Center Networks

Wolfgang Braun · Michael Menth

Abstract Loop-Free Alternates (LFAs) are a local fast-reroute mechanism defined for IP networks. They are simple but suffer from two drawbacks. Firstly, some flows cannot be protected due to missing LFAs, i.e., this concept does not provide full protection coverage, which depends on network topology. Secondly, some LFAs cause loops in case of node or multiple failures. Avoiding those LFAs decreases the protection coverage even further. In this work, we propose to apply LFAs to OpenFlow-based networks. We suggest a method for loop detection so that loops can be avoided without decreasing protection coverage. We propose an implementation with OpenFlow that requires only a single additional flow rule per switch.

We further investigate the percentage of flows that can be protected, not protected, or even create loops in different types of failure scenarios. We consider realistic ring and mesh networks as well as typical topologies for data center networks. None of them can be fully protected with LFAs. Therefore, we suggest an augmented fat-tree topology which allows LFAs to protect against all single link and node failures and against most double failures.

Keywords Software-Defined Networking · OpenFlow · Resilience · Loop-Free Alternates · Scalability

This work has been supported by the German Federal Ministry of Education and Research (BMBF) under support code 16BP12307 (EUREKA-Project SASER). The authors alone are responsible for the content of the paper.

Wolfgang Braun · Michael Menth
University of Tübingen, Department of Computer Science, Sand 13, 72076 Tübingen, Germany E-mail: {wolfgang.braun,menth}@uni-tuebingen.de

1 Introduction

Software-defined networking [1](SDN) has gained a lot of attention in the recent years due to its ability to program the network and facilitate fast and simple introduction of new features and services. Many SDN applications require fine-grained flow definitions and a large number of forwarding rules. However, flow tables in OpenFlow switches have only moderate size and, therefore, scalability problems easily occur. As an example, inter-domain routing information is so extensive that it cannot be stored by contemporary, commercially available OpenFlow switches [2].

Resilience to failures is a challenging topic for OpenFlow-based SDN due to its separation of the control and data plane. Controllers may fail and require backups. In case of link or node failures, predecessor nodes just drop packets until the failure is detected and traffic is rerouted. Involving the controller for that purpose takes time, therefore, local fast reroute techniques are beneficial. If switches send traffic and communicate with their controller over the same physical infrastructure, they may become unreachable for their controllers if links or nodes fail. OpenFlow offers features to automatically forward traffic to alternate next-hops if an interface is down without intervention of a controller. However, there is no recommendation how to leverage this mechanism for the operation of networks.

Resilience mechanisms for OpenFlow networks should protect against all single link failures, they should handle node and multiple failures appropriately, in particular they should not worsen the situation, and they should generate only little overhead in OpenFlow forwarding tables due to their moderate size. Especially the last requirement makes the design of resilient OpenFlow networks challenging because it rules out many straightforward solutions.

In IP networks, traffic may be locally rerouted to an alternate next-hop if the regular next-hop is not reachable if this deviation does not create a forwarding loop. This mechanism is known as Loop-Free Alternate (LFAs). LFAs are well understood, simple, and already available on most modern IP routers. However, they cannot protect all single link failures and their protection coverage depends on the network topology. Some LFAs cause microloops in case of node failures or in case of multiple failures. Avoiding such LFAs generally reduces the protection coverage even further.

In this work, we propose to apply LFAs to OpenFlow-based networks. We suggest a method for loop detection so that loops can be avoided without decreasing protection coverage. We propose an implementation with OpenFlow that requires only a single additional flow rule per switch. We further investigate the percentage of flows that can be protected, not protected, or even create loops in different types of failure scenarios. We consider realistic ring and mesh networks as well as typical topologies for data center networks. None of them can be fully protected with LFAs. Therefore, we suggest an augmented fat-tree topology which allows LFAs to protect against all single link and node failures and against most double failures.

The remainder of the paper is structured as follows. Section 2 discusses related work in the area of fast reroute and scalability for OpenFlow. We present the concept of LFAs in Section 3. Section 4 explains how LFAs can be implemented with OpenFlow and we propose the loop-detecting LFAs. We present our evaluation methodology in Section 5. In Section 6 and Section 7 we present the coverage statistics for carrier-grade and data center networks, respectively. We summarize and discuss our future work in Section 8. Finally, Section 9 concludes this work.

2 Related Work

In [3], the restoration process of OpenFlow in carrier-grade networks was analyzed. They measured the time a controller re-configures the switches in a real testbed when network failures occur. Their results show that the controller reacts within 80 and 130 ms. They also state that the restoration time will be a magnitude higher for large networks and, therefore, local protection schemes are required.

Local failure protection is considered in [4]. They introduce a bidirectional forwarding detection (BFD) component to the OpenFlow switch design to achieve fast protection. A BFD component is responsible to monitor the ports by sending *hello* and *echo* messages frequently over the links to detect network failures. The authors show the viability of the approach using experiments based on MPLS Transport Profile. A similar approach is presented in [5]. The authors also provide fast protection using BFD for the Open vSwitch and analyze the protection switching times. They were able to show that the implementation was able to react within 3 and 30 ms depending on different BFD configurations.

The SlickFlow approach [6] provides resilience in data center networks (DCN) using OpenFlow and is based on source routing. Primary and alternative backup paths are encoded in the packet header and OpenFlow primitives enable hardware-based forwarding. The authors showed positive benefits in a virtualized testbed on DCN topologies.

IP fast reroute (FRR) is an important topic for the routing working group in the IETF and various approaches are discussed: LFAs [7], remote LFAs (rLFAs) [8,9], not-via addresses, and maximally redundant trees [10]. LFAs and rLFAs are simple and require no additional forwarding entries but cannot protect against all single link failures. The latter mechanisms provide full coverage but require additional forwarding entries. The combination of LFAs and not-via addresses cannot significantly reduce the required state [11,12]. MPLS FRR [13] defines several ways to protect the network against failures but explicit backup paths require additional entries which can be reduced by the use of shared tunnels. A comparison of IP-based and MPLS FRR is given in [14].

Inter-domain routing for OpenFlow is improved using wildcard compression in [2]. However, the required forwarding state is still challenging and there is not much space left in the forwarding entries for backup paths. Another compression for Access Control Lists (ACLs), i.e., applicable to OpenFlow table entries, is proposed in [15]. This method preserves space in the forwarding table that can be used for backup paths.

In [16] a reliability analysis of data center networks is given. The authors investigate the reliability potential of three data center topologies: fat-tree, BCube, and DCell networks. The authors investigate maximum and average relative sizes of the connected components after failures. Other metrics are the diameter of the components and path stretches. In our study, we investigate the resilience of LFAs in these topologies, focusing on the applied protocol and required signaling instead of the graph properties of the topologies.

3 Loop-Free Alternates (LFAs)

Loop-Free Alternates (LFAs) [7] were proposed by the IETF for IP fast reroute (IP FRR). LFAs are simple but

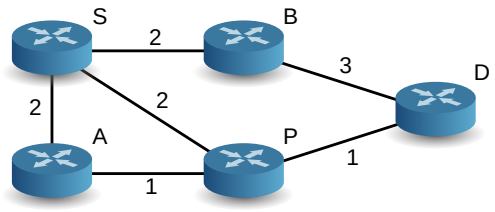

Fig. 1: Example network for LFA computation.

cannot protect against all single link and node failures. They do not require additional forwarding entries but a forwarding entry has to contain a list of alternate next-hops.

The idea of LFAs is very simple: if a failure occurs, packets can be sent to an alternative neighbor instead of the regular next-hop if this redirection will not cause a loop. For each hop that may fail, we have to determine all potential LFAs. The forwarding device can select one or more valid alternates using a tiebreaker or ECMP, respectively. Some LFAs can protect against (1) single link failures, (2) node failures, and even (3) multiple network failures depending on the use of the following three LFA conditions. The protection level is a policy decision of the network operator. We only require the distance function $\text{dist}(u, v)$ to determine whether neighbors fulfill these conditions. No additional computation is required because $\text{dist}(u, v)$ is already required for the primary hop computation. We will explain the conditions by example with the network shown in Figure 1. Packets are sent from source S towards destination D on the shortest path through node P.

The loop-free condition (LFC) protects against single link failures. A neighbor N of S fulfills the condition if

$$\text{dist}(N, D) < \text{dist}(N, S) + \text{dist}(S, D) \tag{1}$$

holds. In the example, this is true for the nodes A (LFC: $2 < 5$) and B (LFC: $3 < 5$) because the distance from them towards destination is smaller than the redirection over S.

The node-protecting condition (NPC) prevents the selection of neighbors that may cause a loop if the primary next-hop has failed. A neighbor is node-protecting if its shortest path to the destination does not lead over P. The condition is defined as

$$\text{dist}(N, D) < \text{dist}(N, P) + \text{dist}(P, D). \tag{2}$$

Consider that node P in the example has failed. Node A fulfills the LFC but not the NPC ($2 \not< 2$). Packets sent to A will loop because S is a simple LFA for A to D (LFC: $3 < 4$). Node B is node protecting ($3 < 5$) and packets sent to B will not traverse P.

The downstream condition (DSC) protects against multiple failures by only redirecting traffic to neighbors that are closer to the destination:

$$\text{dist}(N, D) < \text{dist}(S, D). \tag{3}$$

Node B is not downstream ($3 \not< 3$) and packets would loop between B and S when both $S \to P$ and $B \to D$ are failing because S is a simple LFA for B towards D ($3 < 5$). A is downstream ($2 < 3$) and will not cause a loop even if $A \to P$ has failed because S is not downstream for A towards D ($3 \not< 2$). Note that DSC and NPC are orthogonal, i.e., a downstream LFA can either be node protecting or not.

In this work we compute three kinds of LFAs. First, simple LFAs that only fulfill the loop-free condition (LF-LFAs) which may cause loops when node or multiple failures occur. Second, node-protecting LFAs that both fulfill LFC and NPC (NP-LFAs) that prevent loops for single link and single node failures, and finally loop-preventing LFAs that fulfill the downstream condition (DS-LFAs).

4 OpenFlow-Based LFAs with Loop Detection

In this section, we propose LFAs with loop detection that use additional failure information in the packet header to drop looping packets. We discuss how LFAs with and without loop detection can be implemented with OpenFlow. Then, we discuss how failure information can be encoded in packets with little packet overhead.

4.1 Improving LFAs with Loop Detection

When LFAs for a network are computed to protect against node failures, the node protecting condition can exclude neighbors that protect against single link failures. In general, this leads to fewer alternate next-hops that are allowed to be selected. In addition, if only link failures occur, there may be an LFA available but it is not allowed to be chosen due to the more restricted condition.

Our LFA approach selects LFAs with the highest protection first and reverts to less protecting LFAs if necessary, i.e., we select potential LFAs with degrading degree of protection: (a) NPC and DSC, (b) DSC, (c) LFC and NPC, and (d) LFC only. LFAs of category (a) cannot loop and LFAs with (b)-(d) may loop depending on the exact failure scenario. Therefore, we apply a mark for LFAs (b)-(d) into the packet header. This mark encodes the failure detecting node. Packets either are successfully redirected towards the destination or a

loop occurs. In the latter case, the node can check if the packet contains its location mark and prevents the loop by dropping the packet.

It is important that marks can be incrementally applied to a packet that is sent on alternative paths when multiple failures occur. We explain this with the example network shown in Figure 1. Consider that packets are sent from S to D and the links $S \to P$ and $A \to P$ have failed. Packets have to be redirected over A because there is no LFA of category (a) and A is an LFA of category B. The mark for node S is applied and redirected to A. The packet is not dropped because it has no mark for node A and S is an LFA of category d for A. The mark for A is applied and the packet is sent to S. The loop can be successfully detected in S if the mark of A does not overwrite the mark of S.

Such marks can be implemented using a bit string of a certain length n. Each node has an ID i with $1 \leq i \leq n$ and its mark corresponds to the i-th bit in the string. If the number of available bits in the field is greater than or equal to the number of nodes, the loop detection is optimal. If there are more nodes in the network than available bits, we share IDs across nodes. This can cause false positives when detecting loops and, thus, can lead to unnecessary packet drops. We provide an algorithm to compute appropriate IDs in Section 4.3 and analyze the impact of various bit lengths in Section 6. We refer LFAs with loop detection to LD-LFAs.

4.2 Implementation in OpenFlow and State Requirements

OpenFlow data plane resilience requires OpenFlow 1.1 [17] or later and such an OpenFlow switch contains both flow tables and a group table. A flow table consists of multiple entries and each entry consists of a match and instructions. Flow table entries match packets to flows and each flow entry can define various actions or refer to a group table entry for advanced packet processing. Note that the flow tables have to reside in TCAM which is expensive and limited in size but enables fast packet processing. The group table must not be stored inside TCAM and can be part of less expensive memory.

A group table entry can be referenced by multiple flow table entries and handles all related packets in the same fashion. The group table entry contains a group type, a counters field, and a field for action buckets. The group type defines the kind of group. Backup paths can be implemented using the *fast failover* group type. The action buckets are used to implement the primary and secondary paths. For *fast failover* groups, each action bucket has an associated port which defines its *liveness* and triggers the use of a bucket.

We explain the group table for simple LFAs by the example given in Section 4.1. Packets are sent from S to D. The switch at S contains a flow table entry that matches specific header fields, e.g., its IP address and this entry refers to a group entry. The group entry has type *fast failover* and consists of two action buckets. The first action bucket contains the action to "forward to P" and the second bucket "forward to A". If the link to node P goes down, packets are immediately sent over the next live defined bucket, i.e., to node A. Note that no additional flow table entries are required for LFAs.

LFAs with loop detection are implemented similarly. Consider that the ID for S is 1 and represented by the fifth bit in a bit string with 5 bits. We add an additional flow table entry that matches for exactly that bit using the wildcard expression ****1 and its action is "packet drop". Thus, only packets that loop back to S, i.e., are marked with ID 1, will be dropped. The first action bucket of the group is unchanged. The second action bucket now contains two actions: "apply ID 1" and "forward to A".

In OpenFlow, all header fields that are not required for the forwarding process and additional labels have the potential for ID encoding. For example, the DSCP and ECN field (8 bits) of an IP header can be used if they are not needed in the OpenFlow network. However, they are often required in network operation and we suggest the usage of an additional MPLS label which provide up to 20 bits for ID encoding.

4.3 Computing IDs for Fixed Bit Lengths

When each node has a unique ID, switches do not erroneously detect forward loops. Thus, we want to avoid that two nodes that are part of the same backup path have the same ID. We have developed an algorithm that computes IDs for nodes in such a way that each node with ID x is least interfering towards nodes with the same ID x.

Algorithm 1 assigns IDs $0 \leq i < n_b$ to nodes that we call colors in the following. Initially, the set of uncolored nodes $\mathcal{U}$ comprises all nodes $\mathcal{V}$ and the set of colored nodes $\mathcal{V}$ is empty. Then, nodes $v \in \mathcal{U}$ are assigned a color $c[v]$ in the order of descending node degree δ. Thus, the first n_b nodes are assigned different colors. Afterwards, a node v is assigned a color such that it interferes the least with the colors of already colored nodes. We define the interference inverse to the hop distance $dist(u, v)$ of two nodes u, v having the same color. We compute the overall color interference $cif[i]$ for a color i and the color interfering the least is assigned. The algorithm terminates if all nodes are colored.

Algorithm 1: ID assignment for length-restricted bit label.

input : $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, distance function dist, and number of bits n_b
output: $c[v]$

$\mathcal{U} = \mathcal{V}$ // set of uncolored nodes
$\mathcal{C} = \emptyset$ // set of colored nodes
$i = 0$ // start color
while $\mathcal{U} \neq \emptyset$ **do**
  // choose node v with highest node degree
  $v \leftarrow \text{argmax}_{v \in \mathcal{U}} (\delta(v))$;
  if $i \leq n_b$ **then** $c[v] \leftarrow i$;
  else
    // initialize color interference
    foreach $u \in \mathcal{C}$ **do**
      $cif[u] \leftarrow 0$;
    end
    // compute color interference
    foreach $u \in \mathcal{C}$ **do**
      $cif[c[u]] \leftarrow cif[c[u]] + \frac{1}{\text{dist}(v,u)}$;
    end
    // assign least interfering color
    $c[v] \leftarrow \text{argmin}_{0 \leq j \leq n_b} (cif[j])$;
  end
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{v\}$;
  $\mathcal{U} \leftarrow \mathcal{U} \setminus \{v\}$;
  $i \leftarrow i + 1$;
end

Note that, the proposed algorithm can be changed very easily in an SDN environment. The computation runs in a logically centralized control plane and, thus, only a few elements must be updated.

5 Methodology

In this section we discuss how we analyze the protection of the various LFA approaches for different failure cases. We define multiple kinds of failure scenarios and explain the term protection and coverage in detail.

5.1 Failure Scenarios

A failure scenario s is a set of failed links and nodes. We define several failure scenarios that cover types of network failures. The set $\mathcal{S}^{l,n}$ contains all failure scenarios where l links and n nodes have failed.

We analyze all single link failures $\mathcal{S}^{1,0}$ and all single node failures $\mathcal{S}^{0,1}$. We also consider multiple failure scenarios because LFAs do not require additional forwarding state to protect against multiple failures. For that purpose we use all double link failures $S^{2,0}$ and the combination of single link and single node failures $S^{1,1}$. We do not consider scenarios with additional simultaneous failed elements because their probability is usually significantly lower than two failing elements [18].

	$\lvert\mathcal{T}\rvert$	$\lvert\mathcal{V}\rvert$	$avg(\lvert\mathcal{V}\rvert)$	$\lvert\mathcal{E}\rvert$	$avg(\lvert\mathcal{E}\rvert)$	δ
$\mathcal{T}_S$	37	4 – 82	25	4 – 82	24.6	1.89
$\mathcal{T}_R$	68	6 – 103	31	6 – 103	34.6	2.2
$\mathcal{T}_M$	82	6 – 76	32	10 – 105	44.9	2.96

Table 1: Statistics for the topology sets. For each topology set we provide the number of topologies $|\mathcal{T}|$, nodes $|\mathcal{V}|$, and bidirectional edges $|\mathcal{E}|$. We also provide the average node degree δ.

5.2 Metrics

In our analysis, we route the flows using shortest paths for a specific failure scenario s through the network while applying certain fast reroute algorithms. We investigate whether the flow (1) successfully reaches the destination, (2) is dropped because s removed all physical paths to its destination, (3) is dropped although a physical path to the destination still exists, and (4) causes a microloop. We denote flows as protected if (1) or (2) hold, as unprotected if (3) applies, and as looped if (4) holds. We consider all possible flows in the network and calculate the percentage of protected, unprotected, and looped flows for a single failure scenario s. Considering a set of failure scenarios $\mathcal{S}$, we average these values over all failures contained in that set.

6 Results for Carrier-Grade Networks

In this section we quantify the percentage of flows that LFAs can protect, cannot protect, or for which LFAs cause loops. The latter can be avoided through loop detection. We discuss the considered networks, study the performance of different types of conventional LFAs, and compare it to the one of LFAs with loop detection. Finally, we discuss the impact of the bit length for the ID encoding in packets.

6.1 Carrier-Grade Networks under Study

We evaluate the fast reroute mechanisms for various networks from the topology zoo [19]. We classify these topologies into three categories: star topologies $\mathcal{T}_S$, ring topologies $\mathcal{T}_R$, and mesh topologies $\mathcal{T}_M$. Table 1 provides an overview of the selected networks. We omit $\mathcal{T}_S$ in our analysis due to the lack of alternate neighbors.

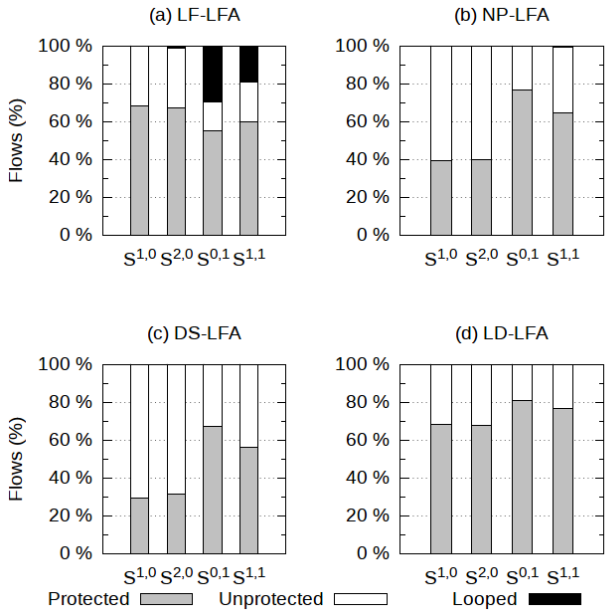

Fig. 2: Percentage of protected flows in mesh topologies.

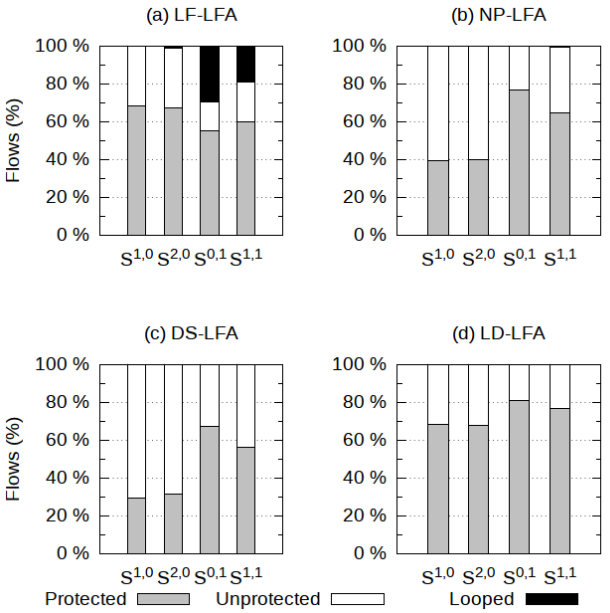

Fig. 3: Percentage of protected flows in ring topologies.

6.2 Flow Analysis for LFAs without Loop Detection

We evaluate the percentage of protected, unprotected, and looped flows for various types of LFAs and for various failures sets. As LFAs can protect significantly fewer flows in ring topologies than in mesh topologies, we conduct our study separately for mesh and ring topologies.

Figures 2 (a) and (c) show the percentage of protection in mesh topologies $\mathcal{T}_M$ for LF-LFAs and DS-LFAs. For single link failures, LF-LFAs protect approximately 68.1% of the flows which is 2.3 times more effective compared to DS-LFAs that only protects about 29.5%.

LF-LFAs cause loops for the other failure scenario sets. There is a similar protection ratio in $\mathcal{S}^{2,0}$ but LF-LFAs cause loops for approximately 1.2% of the traffic. For $\mathcal{S}^{0,1}$ and $\mathcal{S}^{1,1}$, a significant number of loops occur. 29.2% of the traffic loops in single node failure scenarios and 19.1% in $\mathcal{S}^{1,1}$. DS-LFAs protect a similar amount of traffic but cause no loops in these scenarios.

Figures 3 (a) and (c) show the percentage of protection for ring topologies $\mathcal{T}_M$ for LF-LFAs and DS-LFAs. The coverage of LFAs is clearly reduced for all failure scenarios for LF-LFAs. For single link failures, 23.4% less traffic is protected which corresponds to 44.7%. The protection for double link failures is reduced from 67.2% to 59.1%. The number of caused loops is generally reduced. In particular, for $\mathcal{S}^{0,1}$ the amount of loops is reduced by 21.2% to 8%. We observe a reduction by 14.4% to 4.7% for $\mathcal{S}^{1,1}$. We explain this behavior due to the fact that the overall number of available LFAs is significantly smaller in ring structures. DS-LFAs perform very similarly in mesh topologies and ring topologies.

The protection for NP-LFAs is shown in Figure 2 (b) and Figure 3 (b). For all topologies, we observe that loops are significantly reduced compared to LF-LFAs: there are no loops for single link or node failures and only a minimum amount of traffic ($< 0.5\%$) loops for multiple failures. The protection for single and double link failures is reduced to 40% by approximately 28.9% and 27%, respectively. The protection is only slightly reduced for ring topologies.

6.3 Flow Analysis for LFAs with Loop Detection

With loop detection, all LF-LFAs can be used for fast reroute because potential loops can be detected and prevented by dropping packets. In this section, we evaluate the performance of LFAs with loop detection when unique IDs can be assigned per node to record redirecting nodes which prevents any erroneously detected loops. The results for LD-LFAs are shown in Figure 2 (d) and Figure 3 (d).

The protection against single link failures is equivalent to simple LF-LFAs with corresponds to 68.1% in mesh and 44.7% in ring topologies. LD-LFAs prevents all loops for all network scenarios which can also be achieved with DS-LFAs. However, LD-LFAs provide significantly more coverage. We observe an improvement of approximately 36% – 38% for single and double link failures, 13.6% for single node failures, and 20% for

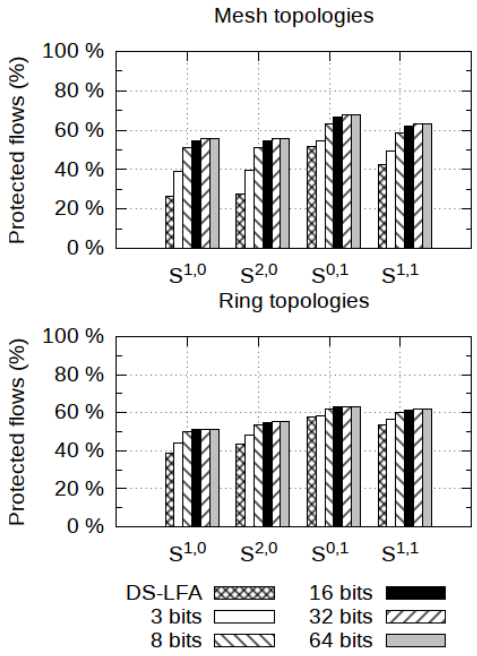

Fig. 4: Percentage of protected flows in large mesh and ring networks for DS-LFAs and LD-LFAs with varying bit lengths. Bit lengths of 8 or 16 are sufficient and their protection is comparable to unlimited ID lengths in large networks.

single link and node failures in mesh topologies. There is less improvement in ring topologies which correspond to about 6.5% – 14.2% more coverage in the different scenarios.

6.4 Impact of Number of Bits Available for ID Encoding

In this section we discuss the impact of the bit length for ID encoding. The number of available bits for the ID can be a limiting factor for protection coverage in large networks. If there are more nodes than available IDs, some nodes share the same ID. If packets are redirected over an LD-LFA and traverse a node with the same ID, the packet is discarded although there is no microloop.

Therefore, we analyze the impact of the bit length on networks that consist of 50 or more nodes. There are 14 mesh and 11 ring networks of the required size in $\mathcal{T}_M$ and $\mathcal{T}_R$. We compare DS-LFAs and LD-LFAs with varying bit lengths. The percentage of protected flows is illustrated in Figure 4. We observe significantly more protected flows for LD-LFAs compared to DS-LFAs even for short bit lengths. LD-LFAs protect 12.4% – 29.2% more traffic for single link failures than basic LFAs. Very short bit lengths lead to clearly less protection than LD-LFAs with long bit lengths, i.e., a bit length of 64 protects 1.4 times more traffic than a bit length of 3. Bit length 8 leads to 51% and bit length 16 to 54.6% protection. The difference in protection coverage of bit lengths from 16 to 64 is negligible.

In ring topologies we observe the same basic trend as in mesh topologies. However, the differences between DS-LFAs and LD-LFAs are generally less significant which can be explained by the reduced availability of LFAs in ring structures.

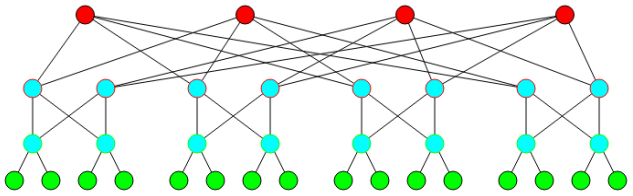

Fig. 5: The fat-tree topology with $k = 4$ consists of four pods and supports up to 16 servers (green nodes).

7 Results for Data Center Networks

In this section we analyze data center networks which structurally differ from carrier-grade networks. We present and discuss three common data center topologies with regard to protection with LFAs. We briefly describe each data center topology, discuss its structure, and provide the protection statistics for the LFA variants discussed in this paper. Moreover, we suggest an augmented fat-tree topology that provides higher protection coverage for LFA than its normal variant.

7.1 Fat-Tree Networks

Fat-tree topologies [20] leverage largely commodity Ethernet switches to interconnect servers in a hierarchical fashion. A fat-tree topology is parameterized with k which denotes both the number of ports of a switch and the number of pods which are described later in this section. The number of servers in the topology are determined by these parameters. Figure 5 shows a fat-tree topology with $k = 4$ that consists of four pods. In the following we discuss the different parts of the topology.

A fat-tree consists of a core layer, an aggregation layer, and an edge layer of switches. The edge layer switches connect to the servers. A so-called pod consists of a set of edge and aggregation switches where every edge switch is connected to every aggregation switch. Every aggregation switch is connected to several core

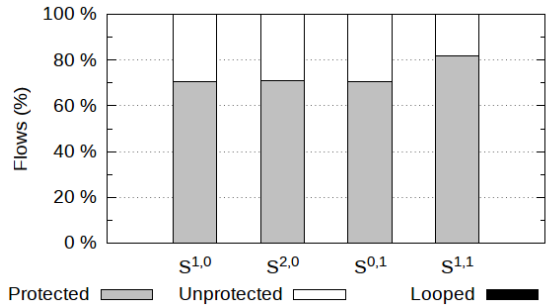

Fig. 6: Percentage of protected flows in fat-tree with $k = 6$. All LFA variants (LF, NP, DS, LD) provide identical protection.

switches but in such a way that any core switch has only a single connection to every pod. Nevertheless, aggregation or core switches can fail while there is still an alternative path available between any pair of edge switches. The maximum size of that structure is limited by the number of switch ports. Given k ports, $\frac{k}{2}$ servers per edge switch and both $\frac{k}{2}$ edge and aggregation switches are supported per pod. Thus, an edge switch has $\frac{k}{2}$ to its servers and another $\frac{k}{2}$ to the aggregation switches within the pod. Every aggregation switch connects $\frac{k}{2}$ core switches. Thus, up to $(\frac{k}{2})^2$ core switches may be used. With this design, $\frac{k^2}{4}$ servers can be supported per pod and $\frac{k^3}{2}$ servers can be supported in total. That means, a fat-tree built of 48-port switches can support up to 27,648 servers. Considering the fact that each server may host multiple virtual machines, large data centers can be constructed using the fat-tree topology.

For the evaluation of fat-trees, we generate only two pods for parameter $k = 6$ and interconnect them redundantly with 4 core switches. Due to the symmetry of fat-trees, this reduction is without loss of generality regarding with regard to resilience aspects.

The results are shown in Figure 6. We observed two key findings. Firstly, we obtain the same coverage results for all LFA variants. Secondly, LFAs do not generate loops in any considered failure scenarios.

In case of single link failures, LF-LFAs can protect only 70.8% of the flows while 29.2% cannot be protected in spite of the high physical redundancy. The reason is that LFAs are only available on the same level or the next level towards the destination. Therefore, core routers lack LFAs if a link towards a pod fails. In a similar way, aggregation switches lack LFAs if their link towards the edge switch fails. For single node failure, 89% of the flows can be protected. Again, core switches cannot find LFAs if the next-hop fails. Finally, 81.9% of the flows can be protected in case of single link and node failures. We conclude that the overall protection is higher than in carrier-grade mesh networks but LFAs cannot achieve full coverage in fat-trees.

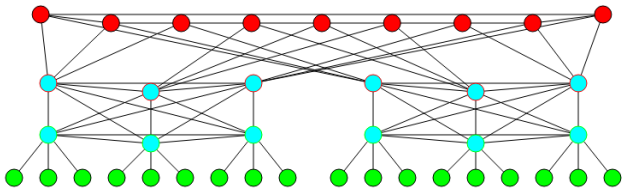

Fig. 7: Augmented fat-tree topology with $k = 8$ and two pods.

7.2 Link-Augmented Fat-Tree Networks

We augment the fat-tree topology using additional links in the individual switch layers. This enables the potential for more alternate next-hops that may be selected as LFA in failure cases. Figure 7 illustrates a link-augmented fat-tree topology using $k = 8$ and two constructed pods. We consecutively connect two neighboring switches in each layer using an additional link. We also connect the first and the last switch in the layer. Thus, each switch layer is transformed into a ring.

Due to additional links within the switch layer, some ports cannot be used to connect servers which reduces the total number of servers supported by the augmented fat-tree. In the following we discuss the trade-off between the normal and the link-augmented fat-tree topology. Each switch requires two ports to be reserved for the additional links. The number of switches is reduced by two in the core layer, in the aggregation and edge layer. Finally, the number of servers per edge switch is also reduced by two. Therefore, the topology supports $k - 2$ pods with each $\frac{k-2}{2}$ aggregation and edge switches. Therefore, the total number of servers is $\frac{(k-2)^3}{4}$. There are up to 24,334 servers supported with 48-port switches which leads to a reduction of approximately 12% to the unmodified topology with 27,648 servers.

In the following, we assume $k = 8$ for the link-augmented fat-tree topology because the number of servers in total and per pod are equal to an unmodified fat-tree topology with $k = 6$. This allows for easy comparison to the unmodified fat-tree presented in Section 7.1. We generate only two pods instead of the possible six pods.

The results are shown in Figure 8. We can see that LF-LFAs can protect against all single link failures but up to 31.6% of the flows cause loops when node or multiple failures are considered. NP-LFAs fully pro-

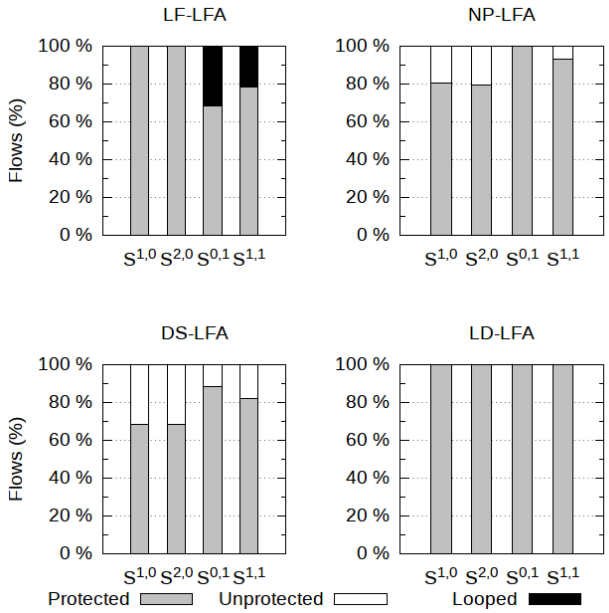

Fig. 8: Percentage of protected flows for the augmented fat-tree topology with $k = 8$. In a very few multiple failure cases ($\mathcal{S}^{2,0}$, $\mathcal{S}^{1,1}$), LD-LFAs cannot protect less than 0.3% of the traffic, which is not visible in the figure.

tect against single node failures and only generates minor percentages for loops ($\leq$ 0.2%). Coverage in single link failure scenario is reduced to 80%. DS-LFAs completely remove all loops but provides less coverage (68.4% – 88.2%) overall. LD-LFAs fully protect single link and node failures and protects almost all flows when multiple failures occur. Only a small amount of flows
($\leq$ 0.3%) cannot be protected. This may happen, e.g., when two links towards a destination edge switch fail.

Packet drops only occur in scenarios where packets are sent from one pod to another and the multiple failures are located between the core layer and the destination pod. We illustrate such a case by the example in Figure 7. Consider that packets are sent from the first server in the first pod to the first server of the second pod in Figure 7. The packet traverses to the first core switch up the tree. The link from the core switch to the second pod fails. There are only two LFAs available: the second or the last core switch and the packet is sent towards one of them. In this example, we assume that the last core switch is chosen and that there is another failure between the last core switch and the pod, i.e., caused by a link or node failure. The first and the seventh switch are both LFAs. The first core switch fulfills the node protecting condition while the second

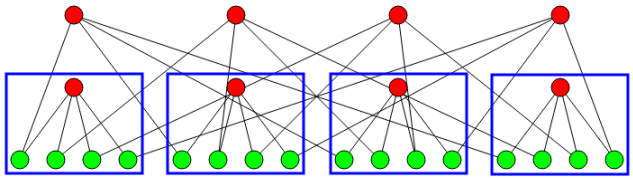

Fig. 9: BCube_1 network with $n = 4$ and $k = 1$ consists of 4 BCube_0 cells. The cells are interconnected using an additional layer of switches.

only fulfills the loop-free condition. Thus, the first core switch is selected and the packet is sent to this switch. The packet is eventually dropped by the loop detection of LD-LFAs.

LFAs can protect against multiple failures within the same pod. Moreover, interacting virtual machines in a data center are often placed as close together as possible, i.e., in the same server or in the same pod [21], so that these cases are quite rare in practice. Thus, we highly recommend the use of LD-LFAs in link-augmented fat-tree topologies because of its high coverage and minimal state requirements of one forwarding entry per switch. There is full protection against all single failures and almost all flows can be protected in multiple failure scenarios.

7.3 BCube Networks

BCube is a specifically designed data center topology intended for shipping-container based, modular data centers [22]. They are built from commodity off-the-shelf (COTS) mini-switches and servers. The servers are interconnected with multiple switches using multi-port network cards and the servers participate in the forwarding process. BCube is a recursively defined topology where a level-1 BCube is built from multiple level-0 BCubes.

In the following, we describe the structure of BCube networks by example. Figure 9 shows a BCube network with parameters $n = 4$ and $k = 1$ where n represents the number of ports per switch and k is the number of levels in the BCube. BCube_1 is built from four basic BCube_0. Each BCube_0 consists of one n-port switch that is connected to n servers. The switch is used for the communication within the BCube.

The BCube_1 is built from the BCube_0 by connecting them using an additional layer of switches in such a way that the i-th switch of the layer is interconnected to the i-th server of each BCube. Therefore, a server must provide $k + 1$ network interfaces to support a BCube_k architecture. Traffic from a BCube_0 to another traverses

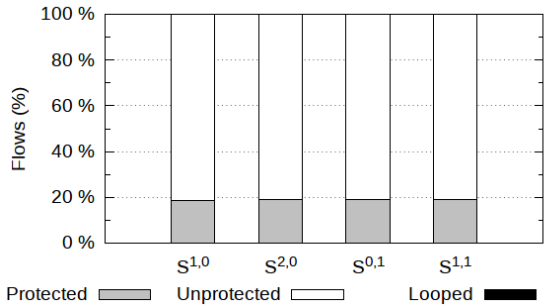

Fig. 10: Percentage of protected flows in BCube with $n = 4$ and $k = 1$. All LFA variants (LF, NP, DS, LD) provide identical protection.

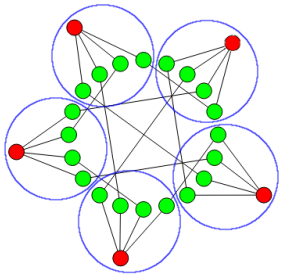

Fig. 11: DCell_1 with $n = 4$ and $k = 1$ consists of 5 DCell_0 cells (blue). Each switch (red node) is part of one DCell_0. Traffic from one cell to another is forwarded using servers (green nodes).

the switch layer above. A BCube_2 is constructed in the same way by connecting four BCube_1 using an additional layer of 16 switches. In general, the i-th layer consists of k^i switches. Note that servers are used for forwarding and, therefore, computing resources must be allocated for the forwarding process. Additional details on BCube networks and their construction can be obtained from the proposing publication [22].

We analyzed BCube networks of different size by altering the n and k parameter. We only provide the results for $n = 4$ and $k = 1$ because they are also representative for BCubes that are parameterized with higher port number n and number of levels k.

We found three key observations. (1) All LFA variants provide exactly the same protection. (2) There are no forwarding loops for all considered scenarios including multiple link and node failures. (3) Protection against failures is very low. Only 18.8% of the flows can be protected against single link failures and 19.1% against single node or multiple failures.

We can explain these results by the structure of BCubes. Consider the fact that a link within a BCube_0 fails. The packets must be sent over a different BCube to reach the destination. However, there are only two hops in the BCube necessary while the redirection using a different BCube clearly consists of more hops. Thus, there is no neighbor that is a valid LFA because the basic loop-free condition cannot be fulfilled.

For BCube-to-BCube communication, there are cases where alternate paths have the same length as the primary path and, thus, can be used as LFA. We explain this by the example shown in Figure 9. Consider that the first server of the first pod sends traffic to the second server of the second pod. There are two equal-length paths towards the destination. Firstly, the traffic can be sent to the second server in the first BCube and then to the destination using the second switch in the connecting layer. Secondly, the traffic can be sent towards the first server in the second BCube and from there using the switch in the BCube. However, traffic sent from the first server in the first cube to the first server of second cube cannot be protected using LFAs.

We conclude that neither basic LFAs nor enhanced LFAs with loop detection can sufficiently protect flows against in link or node failures in BCube networks.

7.4 DCell Networks

DCell networks are proposed in [23] and are similar to BCube networks with respect to recursive definition, use of COTS hardware and mini-switches, and packet forwarding using servers instead of expensive forwarding switches. A DCell can be constructed incrementally and scales to large numbers of servers. The main difference between a BCube and DCell is that while BCube connects smaller BCubes into a larger one using an additional switch layer, DCell directly connects smaller cells to larger ones without the use of any additional switches at all. Thus, there are only switches inside a single cell.

We now discuss the structure of DCells in more detail. A DCell_1 with $n = 4$ and $k = 1$ is shown in Figure 11. It consists of five DCell_0 and each of those cells contains an n-port switch and n servers. Servers within the cell communicate with each other using the switch.

To form a DCell_1, the DCell_0 are fully meshed if treated as a virtual node. This means that each server of a DCell_0 connects to a server of a different DCell_0. Therefore, we require an additional port of each server for an additional level of DCells resulting in $k + 1$ network ports for each server. Packets destined into a different DCell are forwarded using the servers.

DCell_k cells are constructed in the same fashion as DCell_1 by connecting each smaller cell with each other in a full mesh. A DCell_k is constructed using $g_k = t_{k-1} + 1$ DCell_{k-1} where t_{k-1} is the number of

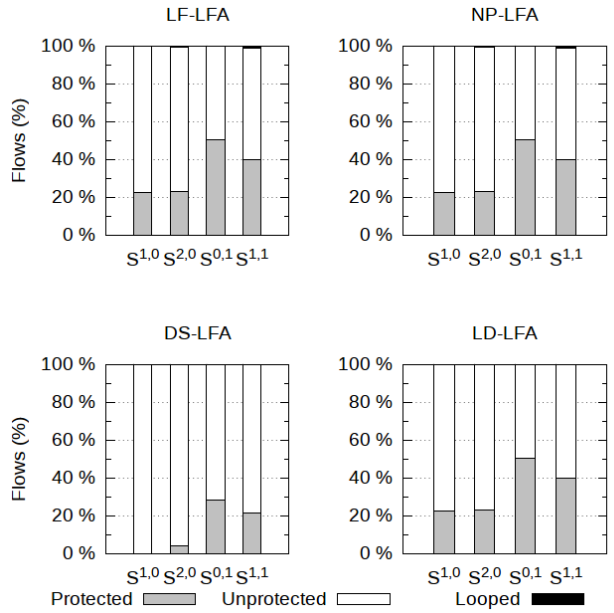

Fig. 12: Percentage of protected flows for the DCell topology with $n = 4$ and $k = 1$.

servers inside a DCell_{k-1}. Therefore, the total number of servers in a DCell_k is $t_k = g_k \cdot t_{k-1}$. For a single DCell_0 there are $t_0 = n$ servers and $g_0 = 1$ holds. For example, a DCell_3 with $n = 6$ can support up to 3.26 million servers. Exact details how to construct a DCell_n network and how to connect the individual ports of each server to servers of different cells can be obtained from the proposal in [23].

We present the results for the DCell network with $n = 4$ and $k = 1$. The results are similar for larger networks and additional results do not provide further insights. The results are shown in Figure 12. The results for LF-LFAs and NP-LFAs are the same and there are a few loops for less than 1% of the flows when multiple failures occur. The protection is low for single and double link failures with approximately 22–23% of all traffic flows. The coverage is about 40–51% for node or multiple failures scenarios.

Downstream LFAs cannot protect any flows for single link failures but for larger DCells a small amount of flows ($\leq$ 7%) can be protected in $\mathcal{S}^{1,0}$. Moreover, DS-LFAs has significantly less coverage than the basic LFAs and varies from 4% to 28.4% protected flows for node and multiple failure scenarios.

LD-LFAs provide similar protection as LF-LFAs and NP-LFAs but remove all loops and, thus, has slightly higher protection. Similar to BCube, LFAs cannot be recommended as protection mechanism for DCell networks.

8 Discussion and Future Work

We addressed resilience and scalability issues in OpenFlow networks with LFAs) which are standardized in the IETF. Basic LFAs can cause microloops in the case of node and multiple failures. It is possible to achieve loop prevention with basic LFAs at the cost of protection coverage. We adopted LFAs to OpenFlow and developed an additional loop detection mechanism that prevents microloops in the case of node and multiple failures to minimize the coverage loss caused by more restrictive LFAs. We analyzed this trade-off of traditional LFAs compared to LFAs with loop detection in OpenFlow networks.

We found that basic LFAs can protect about 70% of the traffic in mesh networks for single link failure scenarios. However, approximately 30% of node failures lead to extra loops. Allowing only downstream LFAs that cannot cause loops in case of node failures, reduces the protected traffic to only 40% for single link failures. LFAs with loop detection successfully protect the 70% of the traffic for single link failures and prevents loops for multiple or node failures.

We observed similar results for ring networks, but the overall protection coverage and the number of microloops is much lower. Only 44.7% of the traffic can be protected when single link failures are considered. However, the amount of looping traffic is reduced by 21.2% to 8% when node failures occur.

We investigated the impact of the length of the bit string for ID encoding in large networks. A visible amount of traffic is dropped if the bit string for the ID is only 3 bits long. When bit string lengths of 16 and more bits are used, hardly any erroneous packet drops occur.

Finally, we analyzed three different types of data center topologies. There is a low coverage of about 18 – 22% in the DCell and BCube networks for single link failure scenarios. However, we found that LFAs protect approximately 70 – 82% of the traffic in the fat-tree network. We developed the link-augmented fat-tree topology variant that allows for full protection with LD-LFAs for single link and single node failures. Moreover, less than 0.3% of flows cannot be protected when multiple failures occur.

We recommend the implementation of LFAs in SDNs when flow table limitation are of concern in an SDN network. These limitation may be caused by inter-domain routing or fine-grained flow rules of an SDN application. LFAs only require a minimum amount of additional state of one entry per switch to implement the loop detection mechanism. Protection coverage can be high or low depending on the network topology.

We will further investigate the use of LFAs for OpenFlow networks. We will integrate remote LFAs (rLFAs) in OpenFlow-based SDN which can protect 100% traffic for single link failures when the network is configured with unit link costs. We will apply loop detection also for rLFAs and investigate its protection with regard to different failure scenarios. The objective is to enhance rLFAs to protect against all single link failures with introducing a minimum amount of additional forwarding entries.

9 Conclusion

Loop-free alternates (LFAs) are used for local fast reroute in IP networks. They mostly cannot protect all traffic and some LFAs may cause loops in case of node or multiple failures. Avoiding such LFAs reduces the protection coverage even further. However, LFAs can be easily implemented with OpenFlow's fast failover group. We proposed loop detection for LFAs in OpenFlow-based networks and an implementation that requires only a single additional forwarding rule per switch. It allows the use of all LFAs without creating loops and, thereby, increases the protection coverage if extra loops in failure cases must be avoided. In our evaluation on multiple mesh and ring networks, normal LFAs can protect 30% and of the traffic without creating extra loops while LD-LFAs increases this value to 70%. To detect loops, LD-LFAs set header bits to indicate nodes having rerouted the packet. An approximation allows to use labels with fewer bits than the number of nodes to scale the mechanism to large networks and minimize header overhead at the expense of packet loss in some failure cases. Our study shows that this packet loss is low for label sizes of 16 or more bits. Furthermore, we showed that LD-LFAs can protect only little traffic for typical data center topologies like BCube or DCell. It protects around 70% of the traffic with fat-trees. Augmenting fat-trees with a few extra links allows LD-LFAs to protect 100% of the traffic for all single link and node failures and even in case of multiple failures almost all traffic can be protected.

References

1. N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Computer Communications Review*, vol. 38, no. 2, pp. 69–74, 2008.
2. W. Braun and M. Menth, "Wildcard Compression of Inter-Domain Routing Tables for OpenFlow-Based Software-Defined Networking," in *European Workshop on Software Defined Networks (EWSDN)*, Sep. 2014, pp. 25–30.
3. S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "OpenFlow: Meeting Carrier-Grade Recovery Requirements," *Computer Communications*, 2012.
4. J. Kempf, E. Bellagamba, A. Kern, D. Jocha, A. Takàcs, and P. Sköldström, "Scalable Fault Management for OpenFlow," in *IEEE International Conference on Communications (ICC)*, 2012, pp. 6606–6610.
5. 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 (EWSDN)*, Sep. 2014, pp. 61–66.
6. R. M. Ramos, M. Martinello, and C. E. Rothenberg, "SlickFlow: Resilient Source Routing in Data Center Networks Unlocked by OpenFlow," in *IEEE Conference on Local Computer Networks (LCN)*, Oct. 2013.
7. A. Atlas and A. Zinin, "RFC5286: Basic Specification for IP Fast Reroute: Loop-Free Alternates ," Sep. 2008.
8. S. Bryant, C. Filsfils, S. Previdi, M. Shand, and N. So, "Remote LFA FRR," http://tools.ietf.org/html/draft-rtgwg-remote-lfa, May 2013.
9. L. Csikor and G. Retvari, "IP Fast Reroute with Remote Loop-Free Alternates: the Unit Link Cost Case," in *IEEE International Workshop on Reliable Networks Design and Modeling (RNDM)*, 2012.
10. M. Menth and W. Braun, "Performance Comparison of Not-Via Addresses and Maximally Redundant Trees (MRTs)," in *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, Ghent, Belgium, Apr. 2013.
11. 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, no. 8, pp. 1300 – 1315, Jun. 2010.
12. M. Hartmann and D. Hock and M. Menth, "Routing Optimization for IP Networks with Loop-Free Alternates," *Computer Networks*, vol. 95, pp. 35 – 50, 2016.
13. P. Pan, G. Swallow, and A. Atlas, "RFC4090: Fast Reroute Extensions to RSVP-TE for LSP Tunnels," May 2005.
14. M. Pioro, A. Tomaszewski, C. Zukowski, D. Hock, M. Hartmann, and M. Menth, "Optimized IP-Based vs. Explicit Paths for One-to-One Backup in MPLS Fast Reroute," in *International Telecommunication Network Strategy and Planning Symposium (Networks)*, Warsaw, Poland, Sep. 2010.
15. C. R. Meiners, A. X. Liu, and E. Torng, "Bit Weaving: A Non-prefix Approach to Compressing Packet Classifiers in TCAMs," *IEEE/ACM Transactions on Networking*, vol. 20, no. 2, pp. 488–500, Apr. 2012.
16. R. Couto, M. Campista, and L. Costa, "A Reliability Analysis of Datacenter Topologies," in *IEEE Globecom*, Dec. 2012, pp. 1890–1895.
17. OpenFlow Switch Consortium and others, "OpenFlow Switch Specification Version 1.1.0," 2011. [Online]. Available: http://archive.openflow.org/documents/openflow-spec-v1.1.0.pdf
18. M. Menth, M. Duelli, R. Martin, and J. Milbrandt, "Resilience Analysis of Packet-Switched Communication Networks," *IEEE/ACM Transactions on Networking*, vol. 17, no. 6, pp. 1950 – 1963, Dec. 2009.
19. 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, no. 9, pp. 1765 –1775, Oct. 2011.

20. M. Al-Fares, A. Loukissas, and A. Vahdat, "A Scalable, Commodity Data Center Network Architecture," *ACM SIGCOMM Computer Communications Review*, vol. 38, no. 4, pp. 63–74, Aug. 2008.
21. M. Banikazemi, D. Olshefski, A. Shaikh, J. Tracey, and G. Wang, "Meridian: An SDN Platform for Cloud Network Services," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 120–127, 2013.
22. C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers," in *ACM SIGCOMM*, Barcelona, Spain, Aug. 2009.
23. C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "Dcell: A Scalable and Fault-tolerant Network Structure for Data Centers," *ACM SIGCOMM Computer Communications Review*, vol. 38, no. 4, pp. 75–86, Aug. 2008.

Author Biographies

Wolfgang Braun studied computer science and mathematics at the University of Tuebingen in Germany and received his diploma degree in 2012. Since then, he is a researcher at the Department of Communications at the University of Tuebingen and pursuing his PhD. His main research interests include software defined networking, OpenFlow, routing scalability, resilience mechanisms, and traffic engineering.

Michael Menth is professor at the Department of Computer Science at the University of Tuebingen, Germany and chairholder of Communication Networks. His special interests are performance analysis and optimization of communication networks, resource management, resilience issues, smart grids, and future Internet. He holds numerous patents and received various scientific awards for innovative work.