Scalability and Resilience Analysis of Software-Defined Networking

Dissertation
der Mathematisch-Naturwissenschaftlichen Fakultät
der Eberhard Karls Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von
Wolfgang Braun
aus Hechingen

Tübingen
2018

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:	09.08.2021
Dekan:	Prof. Dr. Thilo Stehle
1. Berichterstatter:	Prof. Dr. Michael Menth
2. Berichterstatter:	Prof. Dr. Oliver Bringmann

Kurzfassung

Software-defined Networking (SDN) ist eine moderne Architektur für Kommunikationsnetze, welche entwickelt wurde, um die Einführung von neuen Diensten und Funktionen in Netzwerke zu erleichtern. Durch eine Trennung der Weiterleitungs- und Kontrollfunktionen sind nur wenige Kontrollelemente mit Software-Updates zu versehen, um Veränderungen am Netz vornehmen zu können. Allerdings wirft die Netzstrukturierung von SDN neue Fragen bezüglich Skalierbarkeit und Ausfallsicherheit auf, welche in dezentralen Netzstrukturen nicht auftreten. In dieser Arbeit befassen wir uns mit Fragestellungen zu Skalierbarkeit und Ausfallsicherheit in Bezug auf Unicast- und Multicast-Verkehr in SDN-basierten Netzen. Wir führen eine Komprimierungstechnik für Routingtabellen ein, welche die Skalierungsproblematik aktueller SDN Weiterleitungsgeräte verbessern soll und ermitteln ihre Effizienz in einer Leistungsbewertung. Außerdem diskutieren wir unterschiedliche Methoden, um die Ausfallsicherheit in SDN zu verbessern. Wir analysieren sie auf öffentlich zugänglichen Netzwerken und benennen Vor- und Nachteile der Ansätze. Abschließend schlagen wir eine skalierbare und ausfallsichere Architektur für Multicast-basiertes SDN vor. Wir untersuchen ihre Effizienz in einer Leistungsbewertung und zeigen ihre Umsetzbarkeit mithilfe eines Prototypen.

Abstract

Software-Defined Networking (SDN) is a novel architecture for communication networks that has been developed to ease the introduction of new network services and functions. It leverages the separation of the data plane and the control plane to allow network services to be deployed solely in software. Although SDN provides great flexibility, the applicability of SDN in communication networks raises several questions with regard to scalability and resilience against network failures. These concerns are not prevalent in current decentralized network architectures. In this thesis, we address scalability and resilience issues with regard to unicast and multicast traffic for SDN-based networks. We propose a new compression method for inter-domain routing tables to address hardware limitations of current SDN switches and analyze its effectiveness. We propose various resilience methods for SDN and identify their key performance indicators in the context of carrier-grade and datacenter networks. We discuss the advantages and disadvantages of these proposals and their appropriate use cases. Finally, we propose a scalable and resilient software-defined multicast architecture. We study the effectiveness of our approach and show its feasibility using a prototype implementation.

Danksagung

Mein Dank gilt Professor Menth für die Gelegenheit zur Promotion und die Betreuung meiner Dissertation. Professor Bringmann danke ich für die Anfertigung des Zweitgutachtens. Bedanken möchte ich mich ebenfalls bei Professor Walter und Professor Zell für die Teilnahme als Prüfer bei der Disputation.

Ich bedanke mich bei meinen Abschlussarbeitern Manuel Albert, Christian Duta, Joshua Hartmann, Daniel Merling und Gregor Kovacs für die Mitarbeit ihm Rahmen meiner Forschungsprojekte.

Herzlich bedanken möchte ich mich bei Gülsen Ergün und Susanne Uresch für die Unterstützung bei organisatorischen Angelegenheiten.

Mein Dank gilt ebenfalls meinen ehemaligen Kollegen Frederik Hauser, Florian Heimgärtner, Michael Höfling, Mark Schmidt und Andreas Stocknagel für die kollegiale und vielfältige Unterstützung und den fachlichen Austausch. Insbesondere der gute Zusammenhalt und die gute Atmosphäre am Lehrstuhl und die freundschaftliche Verbundenheit mit meinen Zimmerkollegen waren für mich in meiner Promotionszeit sehr wichtig.

Ganz besonders möchte ich mich bei meinen Eltern Elvira und Alexander bedanken. Eure Unterstützung, Zuverlässigkeit und großer Rückhalt haben mir das Studium und die Promotion erst ermöglicht. Ebenfalls bedanke ich mich bei meiner Ehefrau Julia, die mich in den langen Jahren der Promotion mit Geduld und Zuversicht begleitet hat.

Contents

1 Introduction

Software-Defined Networking (SDN) began with the paper "OpenFlow: Enabling Innovation in Campus Networks" [1] back in 2008. McKeown et al. propose an open Application Programming Interface (API) to Ethernet switches called OpenFlow. They illustrate the effectiveness of OpenFlow by introducing new services and network applications in their university campus without updating or changing the OpenFlow switches. In 2011, the Open Networking Foundation (ONF) was founded that standardizes the OpenFlow protocol. The ONF developed the SDN architecture that bases on the core principles of the OpenFlow protocol. Since then, SDN has gained a lot of attention by researchers and industry and many novel approaches in network management, traffic engineering, security, network access control, etc. were proposed.

With OpenFlow, control elements install forwarding rules in the network that resemble traffic forwarding, network services, or network functions. OpenFlow provides fine-grained control on various packet header fields across multiple network layers including Ethernet, IP, and TCP/UDP. Network applications may require fine-grained rules to implement a service, but fine-grained rules increase the number of rules significantly. Moreover, a forwarding rules operate often only on few header fields directly. OpenFlow implements wildcard matching to support non-required fields or bits in the header. However, OpenFlow switches require hardware support for wildcard lookups to support large traffic volumes. OpenFlow switches leverage Ternary-Content Addressable Memory (TCAM) to provide wildcard lookup in constant time. Note that TCAM is relatively expensive, large, and suffers from heat generation and OpenFlow switches generally support only a limited number of flows. This in turn is a limited factor for the number of services and features an OpenFlow-based network may support. Therefore, it is important to address scalability aspects with regard to required forwarding entries in

SDN-based networking.

An SDN network separates the data and the control plane. The data plane is built from network devices that forward and modify traffic according to specific rules. The control plane configures and programs the data plane through a control channel. The separation of data and control plane provides advantages with regard to network innovation and control algorithms but poses challenges to the network design. Switches rely on the forwarding rules provided by the control elements. When network failures occur, the control plane must reconfigure all affected switches. Depending on the current load of control plane and the availability of the control channel between switch and controller, network reconfiguration may be delayed significantly – especially for larger networks. Obviously, the longer the forwarding rules are not reconfigured, packets will be dropped and the traffic does not reach its destination. Therefore, Fast Reroute (FRR) approaches minimize packet by providing pre-established backup paths that carry the traffic around the failed network element. A fast detection of network failures is critical to minimize packet loss and switches can often detect link failures within 50 ms. OpenFlow supports FRR methods since OpenFlow version 1.1 that was standardized in December 2011. However, FRR methods require information that is pre-programmed in the switches and generally require additional forwarding entries to support a FRR scheme. Due to the scalability issues discussed above, it is unknown which FRR mechanism is the most suitable for OpenFlow-based SDN.

1.1 Scientific Contribution

This dissertation is based on various research papers that were developed during two research projects. We briefly describe both research projects and the topics related to this work. We present the resulting works (8 research papers, a demonstrator, and two Internet drafts) and their relating chapters in this dissertation.

The *Safe and Secure European Routing* (SASER) [2] project addressed the challenges of future Internet networks with a focus on security, privacy, service quality, reliability, and scalability. It was organized by Celtic between 2012 and 2015 and funded by the

"Federal Ministry of Education and Research" in Germany, "Direction Générale des Entreprises" in France, and "Finnish Funding Agency for Technology and Innovation" in Finland. Within SASER project, we identified and addressed reliability and scalability issues for the SDN architecture that was proposed in this project. We also investigated traffic engineering potential in resilient OpenFlow-based networks.

The *Design and Performance Evaluation of Future Internet Routing Architectures* (FIR) [3] project is funded by the "Deutsche Forschungsgemeinschaft" (DFG) since 2012. We addressed resilience aspects for IP and SDN-based networks based on Maximally Redundant Trees (MRTs) which are standardized in the Internet Engineering Task Force (IETF). In addition, we designed and analyzed the scalability and resilience of future multicast architectures within this project. A prototype demonstrator shows the feasibility of our approach.

The first paper [4] created in SASER provides an overview of the SDN architecture and lays the foundation of this work. We illustrate the advantages of SDN and highlight issues related to scalability, performance, design options, and reliability. Chapter 2 contains a condensed and revised version of this paper.

Chapter 3 is mainly taken from [5]. We analyze the scalability of OpenFlow-based SDN with regard to inter-domain routing. We found that current routing tables are challenging to hold in memory for OpenFlow switches. We developed a novel compression mechanism for OpenFlow but showed that it only mitigates the memory problem and cannot solve it completely.

The content of Chapter 4 is mainly taken from [6–9]. The first three papers were created within the SASER project and the latter in FIR. We adapt loop-free alternates (LFAs) for SDN in [6] by providing a novel loop-detection mechanism (LD-LFAs) and analyze them on carrier-grade networks. We extend the analysis towards datacenters in [8]. We found that coverage with LD-LFAs is significantly increased but does not guarantee protection against all single link failures. We investigated the potential of load-balancing traffic on backup paths for OpenFlow-based networks in [7]. We found that this approach is effective when traffic overloads and network failures do not occur simultaneously. Otherwise, the complexity of the approach does not justify its usage.

Finally, we investigated multi-protocol label switching (MPLS) and various MRTs variants as resilience mechanisms for SDN in [9]. We found notable differences in performance with regard to path lengths and state requirements and provide recommendations based on network requirements.

Finally, the Chapter 5 is mainly comprised of [10, 11] which were carried out in FIR. These works discuss scalable and reliable multicast using Bit Indexed Explicit Replication (BIER) for IP and SDN-based networks. We propose and analyze different BIER resilience mechanisms on a large network data set. We propose a software-defined multicast architecture and provide a prototype demonstration showing the feasibility of BIER in SDN as open source implementation in [12]. The results and insights gathered in the study are merged into the IETF standardization process into two Internet drafts [13, 14].

1.2 Thesis Outline

The remainder of this dissertation is structured as follows. In Chapter 2 we introduce the technological background of this work. We briefly describe IP networks and their routing and FRR principles. Then, we present the SDN architecture, explain OpenFlow, and discuss design choices and its implications of OpenFlow-based SDN. Chapter 3 provides a scalability analysis of SDN-based inter-domain routing that investigates the effectiveness of novel compression of routing tables based on OpenFlow features. We propose various protection mechanisms for SDN in Chapter 4. We analyze their effectiveness on various networks with regard to multiple key performance indicators. We provide recommendations of their applicability in SDN deployments. Chapter 5 discusses scalable and resilient multicast for SDN. We provide a performance analysis and present a prototype demonstration. Finally, Chapter 6 summarizes this dissertation.

2 Technological Background

We briefly outline the structure and operations of IP and MPLS networks. We explain the SDN architecture, its most important protocols and interfaces and discuss design choices of OpenFlow-based SDN. Note that Sections 2.2–2.3 are based on [9]. Sections 2.4–2.6 constitute a condensed and revised version of [4]. Section 2.7 is based on [5] and Section 2.8 on [9].

2.1 IP Networks

In this section we provide a brief overview how communications networks based on IP technology operate. We focus on the architectural design, the computation of routes, and how they react to network failures.

2.1.1 Structure and Organization of IP Networks

IP networks are generally organized in a decentralized fashion. An IP network is comprised of multiple IP routers that are interconnected in a topology. The routers exchange network and protocol information between each other and leverage the information to calculate how the network traffic is routed through the network. This includes the topology information that each router has learned by identifying its direct neighbors and distributing its current topology information throughout the network. The calculation process is performed on each device independently but in such a way that the resulting forwarding tables are consistent throughout the network. Routing protocols often leverage the Dijkstra shortest path algorithm to find the routes in the network.

In the context of communication networks, the data plane is responsible for handling the network traffic according to a network-wide policy. The data plane is generally configured using lookup tables and associated actions, e.g., an IP prefix should be sent through a specific port or a packet destined to a node should be dropped. The control plane is responsible to calculate and configure tables of the forwarding devices properly. Since routers exchange information between each other and compute appropriate tables and actions independently, IP networks are based on a decentralized control plane in contrast to SDN networks that rely on a more centralized control.

Routing protocols implement the control plane of IP networks. The most common IP routing protocols in current IP networks are Open Shortest Path First (OSPF) and Intermediate System to Intermediate System (IS-IS). They are generally used on the internal network (domain) of Internet service providers (ISPs) or business networks and called Interior Gateway Protocols (IGPs). IP networks support inter-domain routing using Exterior Gateway Protocols (EGPs) such as the Border Gateway Protocol (BGP).

OSPF and IS-IS are designed extensible in such a way that new protocol information can be distributed by them. For example, the BIER architecture that will be presented in Chapter 5 leverages extensions to OSPF [15] and IS-IS [16] to distribute BIER-specific information in a new Type-Length-Value (TLV). Although IP networks support extensions and allow to add new protocol information, new services are installed by updating all or some selected routers in the network. This can be a difficult or expensive task in a production network to achieve.

2.1.2 Network Failure Handling

Routers periodically check if their neighbors are still reachable through their corresponding link. To check the reachability of a neighbor, a router sends a *hello* message and expects that the neighbor responds with an *echo* message. If the neighbor does not respond within a specific time frame or to a number of *hello* messages, a failure of the link (or node) is assumed. This approach is generally known as Bidirectional Forwarding Detection (BFD) and can be implemented within a routing protocol or on the network interface card in hardware.

If a failure is detected, the router updates its topology map and informs its neighbors about the failure. This triggers the reconfiguration of the network and all the routers will compute the new routes after the topology change was propagated throughout the network. The reconfiguration time is dependent on the number of nodes in the network and may require several seconds to complete. Packets cannot be forwarded and will be dropped at the router. The number of lost packets depends on the bandwidth of the link and significant packet loss may occur in backbone networks.

To mitigate the effects of network failures, network devices often provide pre-established backup paths that are only used when a failure occurs and the reconfiguration process has not finished. In combination with fast failure detection provided by BFD components, traffic can be redirected on backup paths in less than 50 ms. This approach is known as Fast Reroute (FRR).

2.2 Multi-Protocol Label Switching (MPLS)

In this section, we briefly explain MPLS and present MPLS FRR and its two variants called one-on-one backup and facility backup. Then, we discuss their state requirements.

2.2.1 MPLS Forwarding

MPLS follows a connection-oriented forwarding approach. A connection is established for a forwarding equivalence class (FEC) between two routers. In IP networks, a FEC is generally an IP prefix. When an IP packet enters the MPLS domain, the ingress node identifies the appropriate MPLS path called label switched path (LSP). The node pushes the MPLS label associated with the path on the packet. The label is only locally relevant and has to be changed from hop to hop. Therefore, each MPLS node checks the label and retrieves the next-hop (NH) and its associated label for the destination. It swaps the label to the new one and forwards it to the NH. At the final node, the MPLS label is popped and the packet is forwarded by IP address. Thus, MPLS is useful to create overlay networks, e.g., for BGP, and as lightweight tunnel mechanism.

2.2.2 Fast Reroute with MPLS

MPLS fast reroute is based on the local repair principle [17]. A node that detects the failure, called point of local repair (PLR), redirects traffic on backup paths that avoids the failed network element. Backup paths may be constructed to reroute around link or node failures, which is a policy decision made before computation. MPLS FRR provides two backup path options for the redirection: one-to-one backup (detour) and facility backup (bypass).

Detour backup paths have to be setup for each LSP in the network. For each node, a detour is installed around the failed element. The detour path merges with the MPLS path further downstream at the node called the merge point (MP). The MP may be any node of the remaining path. Note that, for an LSP of length n, $n - 1$ detour LSPs are required. When a failure is detected, the PLR swaps the label of the default path with

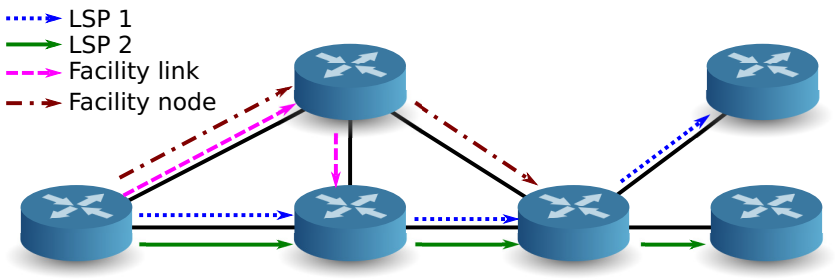


Figure 2.1: MPLS facility backup: link and node protection. The backup path bypasses the failed element. The backup tunnel can be leveraged for different LSP simultaneously.

the label of the detour path. When the packet arrives at the merge point, the labels of the primary paths are used again.

Facility backup requires additional paths for each failed network element, i.e., a tunnel around the failed link to the NH or multiple tunnels to the next-next-hops (NNHs) around failed nodes. The backup paths are illustrated in Figure 2.1. In case of a failure, the PLR swaps the MPLS label appropriate for the end of the backup path. The PLR pushes the backup label on the MPLS label stack and forwards the packet. The backup label is removed at the last hop of the backup path. Finally, the packet is forwarded normally.

2.2.3 State Requirements

Detour backup paths provide more degrees of freedom since they allow to merge with the primary path further downstream. Facility backup always tunnels to the NH or NNHs and may result in longer paths. However, detours are LSP specific and cannot be reused by other LSPs as with facility backup as illustrated in Figure 2.1. In [18], the authors showed that the state required for facility backup is less than for detour backup. Since forwarding state is of concern especially for SDN, we focus on facility backup in the remainder of the work. MPLS_l refers to the link and MPLS_n to the node protecting variant.

2.3 IP and MPLS Fast Reroute Mechanisms

Various approaches for IP FRR were proposed in the IETF. Loop-free alternates (LFAs) [19] redirect traffic to neighbors that avoid loops. They do not require additional forwarding entries but have limited coverage [20]. We adopt LFAs for SDN in Chapter 4 in such a way that the coverage is increased and forwarding loops are prevented. Remote LFAs (rLFAs) achieve full coverage by redirecting packets to alternate nodes using shortest paths for unit link costs [21]. Full coverage guarantees cannot be made by rLFAs. Topology-independent LFAs [22] leverage source routing to guarantee coverage against single link failures independent of the topology.

MPLS FRR [17] provides pre-established backup paths around the failed network element and can be configured to protect against link or node failures as discussed in Section 2.2. We investigate the effectiveness for MPLS FRR for SDN in Chapter 4. Not-via addresses [23] follow the same path layout as MPLS FRR but are not considered anymore in the IETF because of their state and operational complexity. Several works [24–26] exist that compare MPLS and IP resilience methods.

MRTs are currently LFAs one of the most promising resilience mechanism for IP networks in the IETF. MRTs provide two additional forwarding entries per destination that protect against single link and node failures. In [27], we compare the performance of MRTs and not-via addresses and measured long backup paths for MRTs depending on the topology and MRT computation. We adopt, improve, and analyze MRTs in this work for SDN in Chapter 4. The combination of LFAs and MRTs is discussed in [28] and the authors show that their approach can reduce the backup path length. Multiple routing configurations (MRCs) [29] leverage additional routing topologies that are used in failure cases. MRCs also provide multiple routing topologies similar to MRTs but generally require more forwarding state to achieve the same level of protection.

2.4 Software-Defined Networking (SDN)

In this section, we give an overview of SDN. We review the interfaces of SDN and discuss potential control protocols for SDN.

2.4.1 Definition

We discuss SDN along the definition given by the ONF [30]. It is the most accepted SDN definition worldwide because most global players in the networking industry and many IT corporations participate in the ONF.

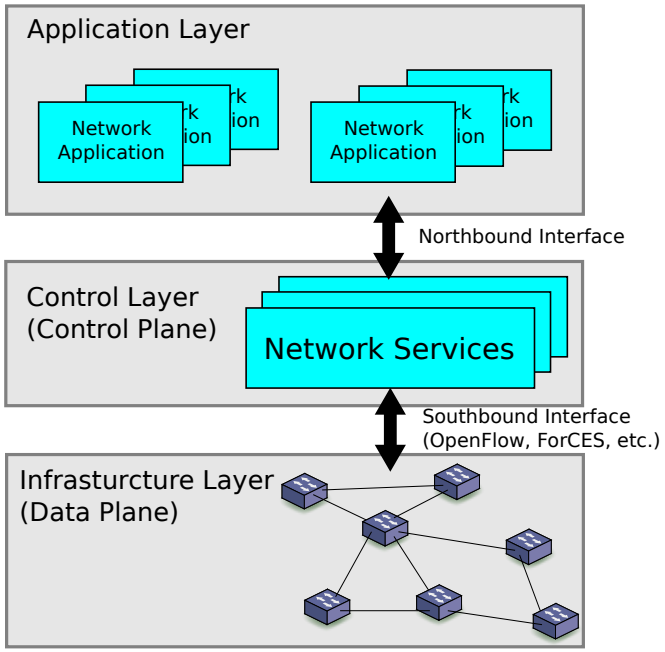


Figure 2.2: A three-layer SDN architecture.

Figure 2.2 illustrates the SDN framework which consists of three layers. The lowest layer is the infrastructure layer, also called data plane. It comprises the forwarding network elements. The responsibilities of the forwarding plane are mainly data forwarding as well as monitoring local information and gathering statistics.

One layer above we find the control layer, also called control plane. It is responsible for programming and managing the forwarding plane. To that end, it makes use

of the information provided by the forwarding plane and defines network operation and routing. It comprises one or more software controllers that communicate with the forwarding network elements through standardized interfaces, which are referred to as southbound interface. We review protocol options for the southbound interface in Section 2.4.2. OpenFlow, which is one of the mostly used southbound interfaces mainly considers switches whereas other SDN approaches consider other network elements such as routers. A detailed description of the OpenFlow protocol is given in Section 2.5.

The application layer contains network applications that can introduce new network features such as security and manageability, forwarding schemes, or assist the control layer in network configuration. Various examples of network applications are presented in [4]. The application layer can receive an abstracted and global view of the network from the controllers and use that information to provide appropriate guidance to the control layer. The interface between the application layer and the control layer is referred to as northbound interface. For the latter, no standardized API exists today and in practice the control software provides its own API to applications. We briefly discuss the northbound interface in Section 2.4.3.

2.4.2 Protocol Options for the Southbound Interface

The most common southbound interface is OpenFlow which is standardized by the ONF. OpenFlow is a protocol that describes the interaction of one or more control servers with OpenFlow-compliant switches. An OpenFlow controller installs flow table entries in switches so that these switches can forward traffic according to these entries. Thus, OpenFlow switches depend on configuration by controllers. A flow is classified by match fields that are similar to Access Control Lists (ACLs) and may contain wildcards. In Section 2.5, we provide a detailed description of OpenFlow and describe the features offered by different versions of the protocol.

OpenFlow was initially designed to provide an open API for Ethernet switches and networks. The ONF provides significant efforts to make the OpenFlow more extensible and allow more use cases to be served by OpenFlow as will be presented in Section 2.5. However, OpenFlow is still lacking the capability to extend the data plane and misses

some advanced control mechanisms. Therefore, the authors of [31] propose "P4: Programming Protocol-Independent Packet Processors" that enables SDN programmers to define new data plane features such as new headers and to write advanced control programs using a programming language. In this thesis, we require the flexibility of P4 to implement scalable software-defined multicast in Chapter 5. We provide a detailed explanation of the P4 language in Section 5.6.3 and describe an complex P4 program of our SDN multicast proposal in Section 5.6.4.

Another option for the southbound interface is Forwarding and Control Element Separation (ForCES) [32, 33] which is discussed and standardized by the IETF since 2004. ForCES is also a framework, not only a protocol; the ForCES framework also separates the control plane and data plane but is considered more flexible and powerful than OpenFlow [34,35]. Forwarding devices are modeled using Logical Functional Blocks (LFBs) that can be composed in a modular way to form complex forwarding mechanisms. Each LFB provides a given functionality such as IP routing. The LFBs model a forwarding device and cooperate to form even more complex network devices. Control elements use the ForCES protocol to configure the interconnected LFBs to modify the behavior of the forwarding elements.

The SoftRouter architecture [36] also defines separate control and data plane functionality. It allows dynamic bindings between control elements and data plane elements which allows a dynamic assignment of control and data plane elements. In [36], the authors present the SoftRouter architecture and highlight its advantages on the BGP with regard to reliability.

ForCES and SoftRouter have similarities with OpenFlow and can fulfill the role of the southbound interface. Other networking technologies are also discussed as well as possible southbound interfaces in the IETF. For instance, the Path Computation Element (PCE) [37] and the Locator/ID Separation Protocol (LISP) [38] are candidates for southbound interfaces.

2.4.3 Northbound APIs for Network Applications

The OpenFlow protocol provides an interface that allows a control software to program switches in the network. Basically, the controller can change the forwarding behavior of a switch by altering the forwarding table. Controllers often provide a similar interface to applications, which is called northbound interface, to expose the programmability of the network. The northbound interface is not standardized and often allows fine-grained control of the switches. Applications should not have to deal with the details of the southbound interface, e.g., the application does not need to know all details about the network topology, etc. For instance, a traffic engineering network applications should tell the controller the path layout of the flows, but the controller should create appropriate commands to modify the forwarding tables of the switches. Thus, network programming languages are needed to ease and automate the configuration and management of the network.

The requirements of a language for SDN are discussed in [39]. The authors focus on three important aspects. (1) The network programming language has to provide means for querying the network state. The language runtime environment gathers the network state and statistics which is then provided to the application. (2) The language must be able to express network policies that define the packet forwarding behavior. It should be possible to combine policies of different network applications. Network applications possibly construct *conflicting* network policies and the network language should be powerful enough to express and to resolve such conflicts. (3) The reconfiguration of the network is a difficult task especially with various network policies. The runtime environment has to trigger the update process of the devices to guarantee that access control is preserved, forwarding loops are avoided, or other invariants are met.

Popular SDN programming languages that fulfill the presented requirements are Frenetic [40], its successor Pyretic [41], and Procera [42]. These languages provide a declarative syntax and are based on functional reactive programming. Due to the nature of functional reactive programming, these languages provide a composable interface for network policies. Various other proposals exist as well. The European FP7 research project NetIDE addresses the northbound interfaces of SDN networks [43].

2.5 The OpenFlow Protocol

The OpenFlow protocol is the most commonly used protocol for the southbound interface of SDN which separates the data plane from the control plane. The white paper about OpenFlow [1] points out the advantages of a flexibly configurable forwarding plane. OpenFlow was initially proposed by Stanford University and it is now standardized by the ONF [30]. In the following, we first give an overview of the structure of OpenFlow, and then describe the features supported by the different specifications ranging from OpenFlow 1.0 to OpenFlow 1.4.

2.5.1 Overview

The OpenFlow architecture consists of three basic concepts. (1) The network is built up by OpenFlow-compliant switches that compose the data plane. (2) The control plane consists of one or more OpenFlow controllers. (3) A secure control channel connects the switches with the control plane. In the following, we discuss OpenFlow switches and controllers and interactions among them.

Figure 2.3: Flow table entry for OpenFlow 1.0.

An OpenFlow-compliant switch is a basic forwarding device that forwards packets according to its flow table. This table holds a set of flow table entries each of which consists of match fields, counters, and instructions as illustrated in Figure 2.3. Flow table entries are also called flow rules or flow entries. The “header fields” in a flow table entry describe to which packets this entry is applicable. They consist of a wildcard capable match over specified header fields of packets. To allow fast packet forwarding with OpenFlow, the switch requires TCAM that allows fast lookup of wildcard matches. The header fields can match different protocols depending on the OpenFlow specification, e.g., Ethernet, IPv4, IPv6, or MPLS. The “counters” are reserved for collecting statistics about flows. They store the number of received packets and bytes as well as the duration

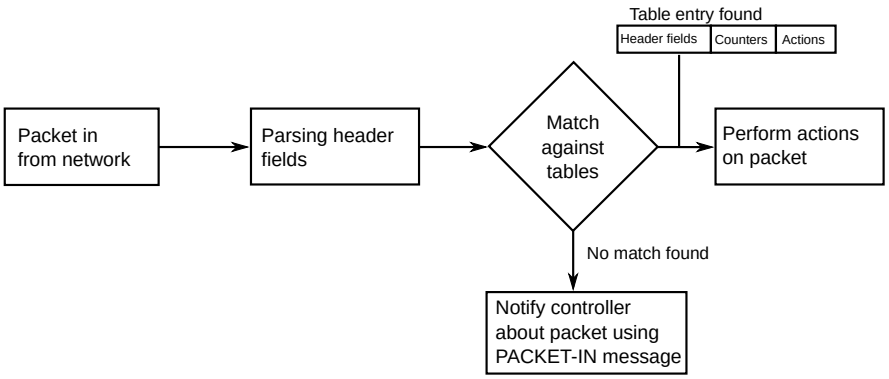


Figure 2.4: Basic packet forwarding with OpenFlow in a Switch.

of the flow. The “actions” specify how packets of that flow are handled. Common actions are “forward”, “drop”, “modify field”, etc.

A software program, called controller, is responsible for populating and manipulating the flow tables of the switches. By insertion, modification, and removal of flow entries the controller can modify the behavior of the switches with regard to forwarding. The OpenFlow specification defines the protocol that enables the controller to instruct the switches. To that end, the controller uses a secure control channel.

Three classes of communication exist in the OpenFlow protocol: controller-to-switch, asynchronous, and symmetric communication. The controller-to-switch communication is responsible for feature detection, configuration, programming the switch, and information retrieval. Asynchronous communication is initiated by the OpenFlow-compliant switch without any solicitation from the controller. It is used to inform the controller about packet arrivals, state changes at the switch, and errors. Finally, symmetric messages are sent without solicitation from either side, i.e., the switch or the controller are free to initiate the communication without solicitation from the other side. Examples for symmetric communication are hello or echo messages that can be used to identify weather the control channel is still live and available.

The basic packet forwarding mechanism with OpenFlow is illustrated in Figure 2.4. When a switch receives a packet, it parses the packet header which is matched against the flow table. If a flow table entry is found where the header field wildcard matches the

header, the entry is considered. If several such entries are found, packets are matched based on prioritization, i.e., the most specific entry or the wildcard with the highest priority is selected. Then, the switch updates the counters of that flow table entry. Finally, the switch performs the actions specified by the flow table entry on the packet, e.g., the switch forwards the packet to a port. Otherwise, if no flow table entry matches the packet header, the switch generally notifies its controller about the packet which is buffered when the switch is capable of buffering. To that end, it encapsulates either the unbuffered packet or the first bytes of the buffered packet using a PACKET-IN message and sends it to the controller; it is common to encapsulate the packet header and the number of bytes defaults to 128. The controller that receives the PACKET-IN notification identifies the correct action for the packet and installs one or more appropriate entries in the requesting switch. Buffered packets are then forwarded according to the rules which is triggered by setting the buffer ID in the flow insertion message or in explicit PACKET-OUT messages. Most commonly, the controller sets up the whole path for the packet in the network by modifying the flow tables of all switches on the path.

2.5.2 OpenFlow Specifications

We now review the different OpenFlow specifications by highlighting the supported operations and the changes compared to their previous major version, and summarize the features of the different versions. Finally, we briefly describe the OF-CONFIG protocol which adds configuration and management support to OpenFlow switches.

2.5.2.1 OpenFlow 1.0

The OpenFlow 1.0 specification [44] was released in December 2009. Ethernet and IP packets can be matched based on source and destination address. In addition, Ethernet type and VLAN fields can be matched for Ethernet, the Differentiated Services (DS) and Explicit Congestion Notification (ECN) fields, and the protocol field can be matched for IP. Moreover, matching on TCP or UDP source and destination port numbers is possible.

Figure 2.4 illustrates the packet handling mechanism of OpenFlow 1.0 as described in Section 2.5.1. The OpenFlow standard exactly specifies the packet parsing and matching

algorithm. The packet matching algorithm starts with a comparison of the Ethernet and VLAN fields and continues if necessary with IP header fields. If the IP type signals TCP or UDP, the corresponding transport layer header fields are considered.

Several actions can be set per flow. The most important action is the forwarding action. This action forwards the packet to a specific port or floods it to all ports. In addition, the controller can instruct the switch to encapsulate all packets of a flow and send them to the controller. An action to drop packets is also available. This action enables the implementation of network access control with OpenFlow. Another action allows modifying the header fields of the packet, e.g., modification of the VLAN tag, IP source and destination addresses, etc.

Statistics can be gathered using various counters in the switch. They may be queried by the controller. It can query table statistics that contain the number of active entries and processed packets. Statistics about flows are stored per flow inside the flow table entries. In addition, statistics per port and per queue are also available.

OpenFlow 1.0 provides basic Quality of Service (QoS) support using queues and OpenFlow 1.0 only supports minimum rate queues. An OpenFlow-compliant switch can contain one ore more queues and each queue is attached to a port. An OpenFlow controller can query the information about queues of a switch. The “Enqueue” action enables forwarding to queues. Packets are treated according to the queue properties. Note that OpenFlow controllers are only able to query but not to set queue properties. The OF-CONFIG protocol allows to modify the queue properties but requires OpenFlow 1.2 and later. We present OF-CONFIG in Section 2.5.2.8

2.5.2.2 OpenFlow 1.1

OpenFlow 1.1 [45] was released in February 2011. It contains significant changes compared to OpenFlow 1.0. Packet processing works differently now. Packets are processed by a pipeline of multiple flow tables. Two major changes are introduced: a pipeline of multiple flow tables and a group table.

We first explain the pipeline. With OpenFlow 1.0, the result of the packet matching is a list of actions that are applied to the packets of a flow. These actions are directly

Header Fields	Counters	Instructions

Figure 2.5: Flow table entry for OpenFlow 1.1 and later.

Instruction	Argument	Semantic
Apply-Actions	Action(s)	Applies actions immediately without adding them to the action set
Write-Actions	Action(s)	Merge the specified action(s) into the action set
Clear-Actions		Clear the action set
Write-Metadata	Metadata mask	Updates the metadata field
Goto-Table	Table ID	Perform matching on the next table

Table 2.1: List of instructions for OpenFlow 1.1

specified by flow table entries as shown in Figure 2.3 and Figure 2.4. With OpenFlow 1.1, the result of the pipeline is a set of actions that are accumulated during pipeline execution and are applied to the packet at the end of the pipeline. The OpenFlow table pipeline requires a new metadata field, instructions, and action sets. The metadata field may collect metadata for a packet during the matching process and carry them from one pipeline step to the next. Flow table entries contain instructions instead of actions as shown in Figure 2.5. The list of possible instructions for OpenFlow 1.1 are given in Table 2.1. The “Apply-Actions” instruction directly applies actions to the packet. The specified actions are not added to the action set. The “Write-Actions” instruction adds the specified actions to the action set and allows for incremental construction of the action set during pipeline execution. The “Clear-Actions” instruction empties the action set. The “Write-Metadata” instruction updates the metadata field by applying the specified mask to the current metadata field. Finally, the “Goto” instruction refers to a flow table and the matching process continues with this table. To avoid loops in the pipeline, only tables with higher ID than the current table are allowed to be referenced.

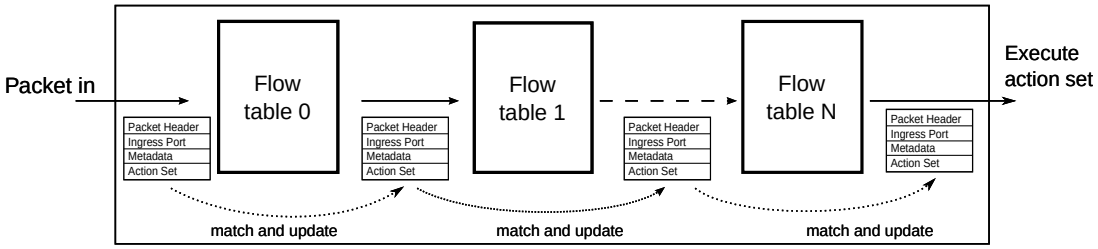


Figure 2.6: OpenFlow pipeline.

Group Identifier	Group Type	Counters	Action Buckets

Figure 2.7: Group table entries for OpenFlow 1.1 and later.

Thus, the matching algorithm will deterministically terminate. If no “Goto” instruction is specified, the pipeline processing stops and the accumulated action set is executed on the packet.

Figure 2.6 illustrates the packet processing of the pipeline. Before the pipeline begins, the metadata field and the action set for a packet are initialized empty. The matching process starts on the first flow table. The packet is matched against the consecutive flow tables from each of which the highest-priority matching flow table entry is selected. The pipeline ends when no matching flow table entry is found or no “Goto” instruction is set in the matching flow table entry. The pipeline supports the definition of complex forwarding mechanisms and provides more flexibility compared to the switch architecture of OpenFlow 1.0.

The second major change is the addition of a group table. The group table supports more complex forwarding behaviors which are possibly applied to a set of flows. It consists of group table entries as shown in Figure 2.7. A group table entry may be performed if a flow table entry uses an appropriate instruction that refers to its group identifier. In particular, multiple flow table entries can point to the same group identifier so that the group table entry is performed for multiple flows.

The group table entry contains a group type, a counters field, and a field for action buckets. The counters are used for collecting statistics about packets that are processed by this group. A single action bucket contains a set of actions that may be executed

depending on the group type. There are possibly multiple action buckets for a group table entry. The group types define which of them are applied. Four group types exist and we illustrate the use of two of them.

The group type “all” is used to implement broadcast and multicast. Packets of this group will be processed by all action buckets. The actions of each bucket are applied to the packet consecutively. For example, a group table entry with type “all” contains two action buckets. The first bucket consists of the action “forward to port 1”. The second bucket consists of the action “forward to port 2”. Then, the switch sends the packet both to port 1 and port 2.

The group type “fast failover” is used to implement backup paths. We first explain the concept of *liveness* for illustration purposes. An action bucket is considered live if all actions of the bucket are considered live. The liveness of an action depends on the liveness of its associated port. However, the liveness property of a port is managed outside of the OpenFlow protocol. The OpenFlow standard only specifies the minimum requirement that a port should not be considered live if the port or the link is down. A group with type “fast failover” executes the first live action bucket and we explain this by the following example. The primary path to a flow’s destination follows port 3 while its backup path follows port 4. This may be configured by a group table with a first action bucket containing the forwarding action towards port 3, and a second action bucket containing the forwarding action towards port 4. If port 3 is up, packets belonging to this group are forwarded using port 3; otherwise they are forwarded using port 4. Thus, the “fast failover” group type supports reroute decisions that do not require immediate controller interaction. Thus, fast reroute mechanisms can be implemented that ensure minimum packet loss in failure cases. Fast reroute mechanisms such as MPLS fast reroute [17] or IP fast reroute [19] can be implemented with OpenFlow group tables.

As an optional feature, OpenFlow 1.1 performs matching of MPLS labels and traffic classes. Furthermore, MPLS-specific actions like pushing and popping MPLS labels are supported. In general, the number of supported actions for OpenFlow 1.1 is larger than for OpenFlow 1.0. For example, the Time-To-Live (TTL) field in the IP header can be decremented which is unsupported in OpenFlow 1.0.

Meter Identifier	Meter Bands	Counters

Figure 2.8: Meter table entry.

OpenFlow 1.1 provides additional statistics fields due to the changed switch architecture. Controllers can query statistics for the group table and group entries as well as for action buckets.

2.5.2.3 OpenFlow 1.2

OpenFlow 1.2 [46] was released in December 2011. It comes with extended protocol support, in particular for IPv6. OpenFlow 1.2 can match IPv6 source and destination addresses, protocol number, flow label, traffic class, and various ICMPv6 fields. Vendors have new possibilities to extend OpenFlow by themselves to support additional matching capabilities. A TLV structure, which is called OpenFlow Extensible Match (OXM), allows to define new match entries in an extensible way.

With OpenFlow 1.2, a switch may simultaneously be connected to more than a single controller, i.e., it can be configured to be administrated by a set of controllers. The switch initiates the connection and the controllers accept the connection attempts. One controller is defined master and programs the switch. The other controllers are slaves. A slave controller can be promoted to the master role while the master is demoted to the slave role. This allows for controller failover implementations.

2.5.2.4 OpenFlow 1.3

OpenFlow 1.3 [47] introduces new features for monitoring and operations and management (OAM). To that end, the meter table is added to the switch architecture. Figure 2.8 shows the structure of meter table entries. A meter is directly attached to a flow table entry by its meter identifier and measures the rate of packets assigned to it. A meter band may be used to rate-limit the associated packet or data rate by dropping packets when a specified rate is exceeded. Instead of dropping packets, a meter band may optionally recolor such packets by modifying their DS field. Thus, simple or complex QoS

frameworks can be implemented with OpenFlow 1.3 and later specifications.

The support for multiple controllers is extended. With OpenFlow 1.2, only fault management is targeted by a master/slave scheme. With OpenFlow 1.3, arbitrary auxiliary connections can be used to supplement the connection with the master controller and the switch. Thereby, better load balancing in the control plane may be achieved. Moreover, per-connection event filtering is introduced. This allows controllers to subscribe only to message types they are interested in. For example, a controller responsible for collecting statistics about the network can be attached as auxiliary controller and subscribes only to statistics events generated by the switches.

OpenFlow 1.3 supports IPv6 extensions headers. This includes, e.g., matching on encrypted security payload (ESP) IPv6 header, IPv6 authentication header, or hop-by-hop IPv6 header. Also support for Provider Backbone Bridge (PBB) is added as well as other minor protocol enhancements.

2.5.2.5 OpenFlow 1.4

OpenFlow 1.4 [48] was released in October 2013. The ONF improved the support for the OXM. TLV structures for ports, tables, and queues are added to the protocol and hard-coded parts from earlier specifications are now replaced by the new TLV structures. The configuration of optical ports is now possible. In addition, controllers can send control messages in a single message bundle to switches. Minor improvements of group tables, flow eviction on full tables, and monitoring features are also included.

2.5.2.6 OpenFlow 1.5

The current OpenFlow 1.5 specification [49] was released in March 2015. There are four major changes compared to OpenFlow 1.4. Firstly, egress tables are added as optional feature. Egress tables enable packet processing in the context of the output port. This means that packets sent to an output port are processed by one or more egress tables where actions are allowed to change the header fields but cannot redirect the packet to a different output port or to group table entry. Secondly, OpenFlow 1.5 introduces logical ports that recirculate packets back to the OpenFlow pipeline after final packet processing

	1.0	1.1	1.2	1.3 & 1.4 & 1.5
Ingress Port	X	X	X	X
Metadata		X	X	X
Ethernet: src, dst, type	X	X	X	X
IPv4: src, dst, proto, ToS	X	X	X	X
TCP/UDP: src port, dst port	X	X	X	X
MPLS: label, traffic class		X	X	X
OXM			X	X
IPv6: src, dst, flow label, ICMPv6			X	X
IPv6 Extension Headers				X

Table 2.2: OpenFlow match fields of versions 1.0 to 1.5.

has finished. Thirdly, OpenFlow enables the support for different underlays than Ethernet, i.e., the final encapsulation of the packet can be done in other protocols such as Optical Transport Networks (OTN). Finally, generic statistic generation is introduced. The OpenFlow eXtensible Statisitcs (OXS) allow switches to encode any arbitrary flow entry statistics and transfer them to the controller. Moreover, controllers can configure switches to push statistics automatically when programmed thresholds are exceeded. In previous specifications, controllers needed to frequently pull statistics which increases the control plane load for larger networks notably.

2.5.2.7 Summary of OpenFlow Specifications and Controllers

Table 2.2 provides the supported protocols and available match fields of the discussed OpenFlow versions. Table 2.3 compiles the types of statistics collected by a switch which can be queried by a controller.

A list of open source controllers is given in Table 2.4. The NOX controller [50] was initially developed at Stanford University and can be downloaded from [51]. It is written in C++ and licensed under the GNU General Public License (GPL). The NOX controller was used in many research papers. The POX [51] controller is a rewrite of the NOX con-

	1.0	1.1	1.2	1.3 & 1.4	1.5
Per table statistics	X	X	X	X	X
Per flow statistics	X	X	X	X	X
Per port statistics	X	X	X	X	X
Per queue statistics	X	X	X	X	X
Group statistics		X	X	X	X
Action bucket statistics		X	X	X	X
Per-flow meter				X	X
Per-flow meter band				X	X
Extensible statistics					X

Table 2.3: Statistics that can be measured for different parts of the OpenFlow switch by OpenFlow version.

troller in Python and can be used on various platforms. Initially, POX was also published under the GPL but is available under the Apache Public License (APL) since November 2013. The Beacon [52] controller was also developed at Stanford University but is written in Java. The controller is available under a BSD license. The Floodlight controller [53] is a fork of the Beacon controller and is sponsored by Big Switch Networks. It is licensed under the APL. The Maestro controller [54] is developed at Rice University and written in Java. The authors emphasize the use of multi-threading to improve the performance of the controller in larger networks. It is licensed under the GNU Lesser General Public License (LGPL). The NodeFLow [55] controller is written in Java and is based on the Node.JS library. It is available under the MIT license. The Trema [56] controller is written in C and Ruby. It is possible to write plugins in C and in Ruby for that controller. It is licensed under the GPL and developed by NEC. The OpenDaylight controller [57] is written in Java and hosted by the Linux Foundation. The OpenDaylight controller has no restriction on the operating system and is not bound to Linux. The controller is published under the Eclipse Public License (EPL). Finally, the Ryu SDN framework [58] is written in Python and supports various OpenFlow specifications and other southbound interfaces, e.g., Netconf. Ryu provides a sophisticated API that aims

Name	Language	License	Comment
NOX [50]	C++	GPL	Initially developed in Stanford University. NOX can be downloaded from [51].
POX [51]	Python	Apache	Forked from the NOX controller. POX is written in Python and runs under various platforms.
Beacon [52]	Java	BSD	Initially developed in Stanford.
Floodlight [53]	Java	Apache	Forked from the Beacon controller and sponsored by Big Switch Networks.
Maestro [54]	Java	LGPL	Multi-threaded OpenFlow controller developed at Rice University.
NodeFLow [55]	JavaScript	MIT	JavaScript OpenFlow controller based on Node.JS.
Trema [56]	C and Ruby	GPL	Plugins can be written in C and in Ruby. Trema is developed by NEC.
OpenDaylight [57]	Java	EPL	OpenDaylight is hosted by the Linux Foundation but has no restrictions on the operating system.
Ryu [58]	Python	Apache	Component-based SDN framework. Provides a sophisticated API for control applications and supports multiple southbound interfaces.

Table 2.4: List of available open source OpenFlow controllers.

to make the creation of new network management and control applications easier. It is published under the Apache License 2.0 and is officially integrated into OpenStack Networking (neutron).

2.5.2.8 OF-CONFIG

The OF-CONFIG protocol adds configuration and management support to OpenFlow switches. It is also standardized by the ONF. OF-CONFIG provides configuration of various switch parameters that are not handled by the OpenFlow protocol. This includes features like setting the administrative controllers, configuration of queues and ports, etc. The mentioned configuration possibilities are part of OF-CONFIG 1.0 [59] which was released in December 2011. The initial specification requires OpenFlow 1.2 and later. As of this writing, the current specification is OF-CONFIG 1.2 and supports the configuration of OpenFlow 1.3 switches.

2.6 Design Choices for OpenFlow-Based SDN

Today, SDN is mostly used for flexible and programmable data centers. There is a need for network virtualization, energy efficiency, and dynamic establishment and enforcement of network policies. An important feature is the dynamic creation of virtual networks, commonly referred to as Network-as-a-Service (NaaS). Even more complex requirements arise in multi-tenant datacenter environments. SDN can provide these features easily due to its flexibility and programmability. However, SDN is also discussed in a network or ISP context. Depending on the use case, the design of SDN architectures varies a lot. In this section, we point out architectural design choices for SDN. We will discuss their implications with regard to performance, reliability, and scalability of the control and data plane, and refer to research on these topics.

2.6.1 Control Plane: Physically vs. Logically Centralized

Typically, a centralized control plane is considered for SDN. It provides a global view and knowledge of the network and allows for optimization and intelligent control. It can be implemented in a single server which is a physically centralized approach. Obviously, a single controller is a single point of failure as well as a potential bottleneck. Thus, a single control server is most likely not an appropriate solution for networks due to lack of reliability and scalability.

As an alternative, a logically centralized control plane may be used to provide more reliability and scalability. It consists of physically distributed control elements that interface with each other through the so-called east- and westbound interface. Since that distributed control plane interfaces with other layers like a centralized entity, the data plane and network applications see only a single control element. A challenge for logically centralized control is the consistent and correct network-wide behavior. Another common term for the SDN logically centralized control plane is “network operating system” (Network OS).

Several studies investigated the feasibility, scalability, and reliability of logically centralized control planes. One research issue is the placement of the distributed control ele-

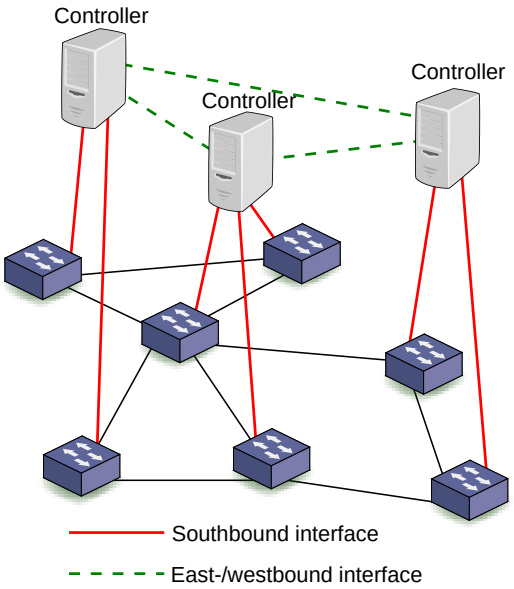


Figure 2.9: Logically centralized control plane.

ments inside the network. In [60], the importance of limited latency for control communication in OpenFlow-based networks is highlighted. To meet these latency constraints, they propose a number of required controllers with their position in the network. Hock et al. [61] optimize the placement of controllers with regard to latency as well as controller load, reliability, and resilience. Their proposed method can be used to implement a scalable and reliable SDN control plane.

The authors of [62] propose a control plane named HyperFlow that is based on the NOX OpenFlow controller. They discuss the management of network applications and the consistent view of the framework in detail. For example, when a link fails, one controller notices the failure but other controllers may not be aware of the link failure. The HyperFlow architecture ensures in such cases that network applications operate on consistent state of the network even though control elements may not share identical knowledge about the network. A hierarchical control platform called Kandoo is proposed in [63] which organizes controllers in lower and higher layers. Controllers in lower layers handle local network events and program the local portions of the network under their control. Controllers on higher layers take network-wide decisions. In

particular, they instruct and query the local controllers in lower layers.

Another study [64] compares the performance of network applications that run on a distributed control platform. Network applications that are aware of the physical decentralization showed better performance than applications that assume a single network-wide controller.

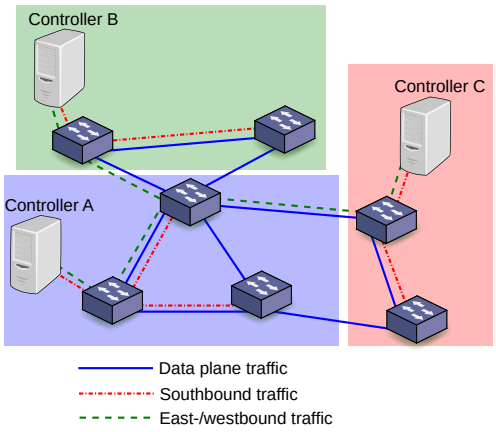
Jarschel et al. [65] model the performance of OpenFlow controllers by a M/M/1 queuing system. They estimate the total sojourn time of a packet in a controller which is mostly affected by its processing speed. Furthermore, they calculate the packet drop probability of a controller.

2.6.2 Control Plane: Inband vs. Out-of-Band Signalling

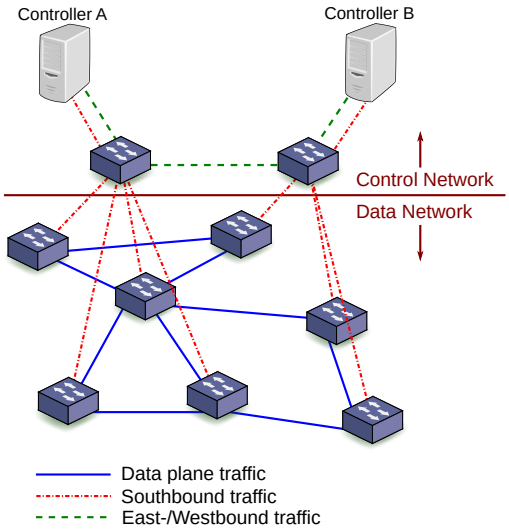
In the following, we will discuss the communication between the control elements and the forwarding devices on the data plane. This control channel has to be secure and reliable. In SDN-based datacenter networks, this control channel is often built as a separate physical network in parallel to the data plane network. Carrier and ISP networks have different requirements. They often span over a country or a continent. Therefore, a separate physical control network might not be cost-efficient or viable at all. Thus, two main design options for the control channel exist: in-band control plane and out-of-band control plane.

With in-band control, control traffic is sent like data traffic over the same infrastructure. This is shown in Figure 2.10a. This variant does not require an additional physical control network, but has other major disadvantages. Firstly, the control traffic is not separated from the data traffic which raises security concerns. Failures of the data plane will also affect the control plane. Thus, it is possible that a failure disconnects the switch from its control element which makes restoration of the network more complicated.

Out-of-band control requires a separate control network in addition to the data network as illustrated in Figure 2.10b. This is a common approach in datacenters that are limited in geographical size. In the datacenter context, the maintenance and cost of an additional control network is usually acceptable. This may be different for wide-ranging networks such as a carrier networks where a separate control network can be costly with



(a) Inband signalling.



(b) Out-of-band signalling.

Figure 2.10: Inband and out-of-band signalling.

regard to Capital Expenditure (CAPEX) and Operational Expenditure (OPEX). The advantages of an out-of-band control plane are that the separation of data and control traffic improves security. Data plane network failures do not affect control traffic which eases network restoration. Moreover, a separate control plane can be implemented more secure and reliable than the data plane. This ensures high availability of the control plane and can be crucial for disruption-free network operation.

A possible approach that combines in-band and out-of-band control planes for carrier networks is based on the control planes in optical networks such as Synchronous Optical Networking (SONET), Synchronous Digital Hierarchy (SDH), and OTN. In such networks, an Optical Supervisory Channel (OSC) may be established on a separate wavelength but on the same fiber over which data traffic is carried. In a similar way, a dedicated optical channel could be allocated for SDN control traffic when an optical layer is available. Also other lower layer separation techniques may be used to implement separated control and data networks over the same physical infrastructure.

2.6.3 Management of Flow Entries: Proactive vs. Reactive

We first explain why flow tables in OpenFlow switches are limited in size, and then we discuss two approaches for flow table entry management.

In the SDN architecture, the control plane is responsible for the configuration of the forwarding devices. With OpenFlow, the controller installs flow table entries in the forwarding tables of the switches. As discussed in Section 2.5, an entry consists of match fields, counters, and forwarding actions. The OpenFlow match fields are wildcards that match to specific header fields in the packets. Wildcards are typically installed in TCAM to ensure fast packet matching and forwarding. However, TCAM is very expensive so that it needs to be small; as a consequence, only a moderate number of flow entries can be accommodated in the flow table. In the following, we will describe two flow management approaches: *proactive* and *reactive* flow management. Both variants are not mutually exclusive: it is common in OpenFlow networks to install some flows proactively and the remaining flows reactively.

The controller is able to install flow entries permanently and in particular before they

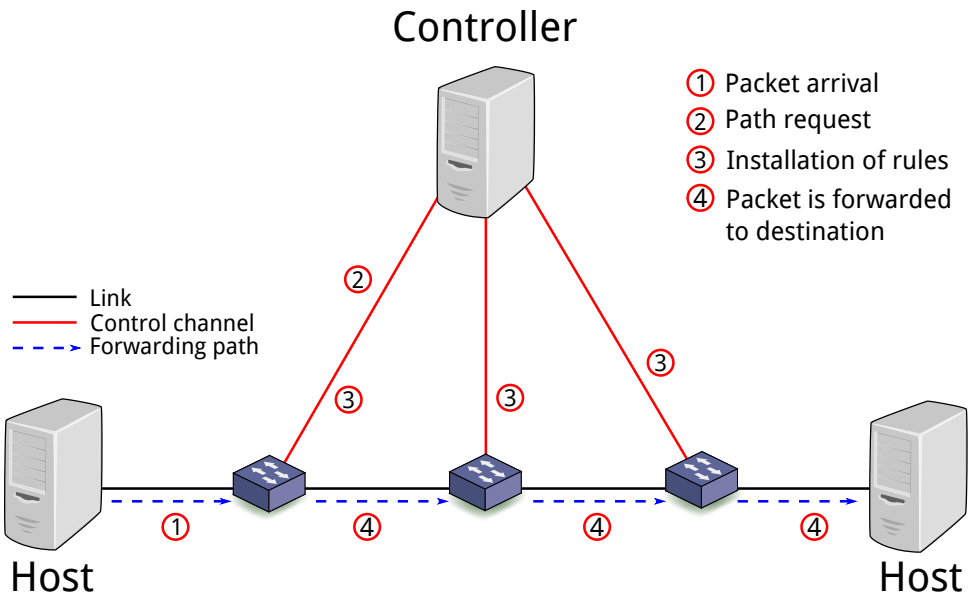


Figure 2.11: Reactive flow management.

are actually needed. This approach is referred to as *proactive* flow management [66]. But this approach has a disadvantage: flow tables must hold many entries that might not fit into the expensive TCAM.

To cope with small flow tables, flow entries can also be installed *reactively*, i.e., installed on demand. This is illustrated in Figure 2.11. (1) Packets can arrive at a switch where no corresponding rule is installed in the table and, therefore, the switch cannot forward the packet on its own. (2) The switch notifies the controller about the packet. (3) The controller identifies the path for the packet and installs appropriate rules in all switches along the path. (4) Then, the packets of that flow can be forwarded to their destination.

The mechanisms to enable reactive flow management in OpenFlow is based on timeouts. The controller sets an expiry timer that defaults to one second. The switch tracks the duration to the last match for all entries. Unused entries are removed from the switch. When more packets of an expired flow arrive, the controller must be queried for path installation again.

Both flow management approaches have different advantages and disadvantages. The

reactive approach holds only the recently used flow entries in the table. On the one hand, this allows to cope with small forwarding tables. On the other hand, controller interaction is required if packets of a flow arrive in a switch that has no appropriate entry in the flow table. After some time the correct entry will be eventually installed in the switch so that packets can be forwarded. The resulting delay depends on the control channel and the current load of the controller. Thus, reactive flow management reduces state in the switches and relaxes the need for large flow tables, but it increases the delay and reliability requirements of the control channel and control plane software. Especially failures of the control channel or the controller will have significant impact on the network performance if flow entries cannot be installed timely.

With a proactive flow management approach, all required flow entries are installed in the switches by the controller. Depending on the use case and network, a huge amount of flows must be installed in switches, e.g., BGP routing tables can contain hundreds of thousands IP prefixes. This may require switch memory hierarchy optimizations that are not needed with reactive flow management. While proactive flow management increases state requirements in switches, it relaxes the requirements on control plane and software controller performance, and is more robust against network failures in the control plane. That means, if the controller is overloaded or the communication channel fails, the data plane is still fully functional.

The problem with limited TCAM and proactive flow management is a special concern in carrier networks. They mostly use the BGP which has significantly high state requirements [67]. Thus, analyses of the feasibility and practicability of SDN in that context with regard to different flow management approaches and state-heavy protocols are relevant and viable solutions are needed.

The authors of [68] use a reactive approach to SDN and BGP. They leverage knowledge of the traffic distribution that they have measured on real data to offload flows with high traffic portion, called heavy hitters, to the forwarding plane. Entries for flows with low packet frequency are not held in the flow table when space is not available. Those flow entries are kept in the controller and packets belonging to low-frequency flows are sent from the switch to the controller which has the complete forwarding knowledge of

all flows. The proposed SDN software router is presented in [69]. Other approaches try to reduce the flow table size. For example, source routing techniques can be leveraged to significantly reduce the number of flow table entries as shown by Soliman et al. [70]. In their source routing method, the ingress router encodes the path in form of interface numbers in the packet header. The routers on the path forward the packet according to the interface number of the path in the header. Since packets of source-routed flows contain the path in their headers, the switches on the path do not require a flow table entry for them. However, their method requires some minor changes to the OpenFlow protocol to support the source routing.

Forwarding state for BGP is also a concern in traditional networks without challenging limitations of forwarding tables. Various approaches to solve this problem exist today. The authors of [71] try to improve the scaling of IP routers using tunneling and virtual prefixes for global routing. Virtual prefixes are selected in such a way that prefixes are aggregated efficiently. Their solution requires a mapping system to employ the virtual prefixes. Ballani et al. [72] have a similar solution where they remove parts of the global routing table with virtual prefixes. They show that this technique can reduce the load on BGP routers. The idea of virtual prefixes was standardized in the IETF [73]. Distributed hash tables can also be used to effectively reduce the size of BGP tables as shown in [74]. Other methodologies are based on the idea of efficient compression of the forwarding information state as shown in [75]. The authors use compression algorithms for IP prefix trees in such a way that lookups and updates of the Forwarding Information Base (FIB) can be done in a timely manner. However, match fields in OpenFlow may contain wildcards and, therefore, have more freedom with regard to aggregation than prefixes in IP routing tables. Appropriate methodologies must be developed and tested to reduce state in OpenFlow switches.

2.7 Existing Forwarding Table Compression Methods

One of the most important routing table compression method was developed in 1999. It is called Optimal Routing Table Constructor (ORTC) [76] and calculates minimum-size routing tables but does not support incremental routing table updates. The authors of [77] achieve fast incremental updates of compressed routing tables but only sub-optimal compression. Incremental updates are also considered in [78] and [79]; the latter is directly based on ORTC.

A different approach is the compression of the data structure that represents the routing table. In [75], entropy bound compression is investigated to significantly reduce the size of the routing table through shared data structures.

Compression of ACLs or firewall rules is similar to routing table compression. Entries of ACLs may match more than a single field; they typically support source and destination address, port number, type, etc. In [80], a compression method based on decision trees and hyper-cuts is presented. The TCAM Razor [81] is able to minimize lists of ACLs into smaller lists. The authors also proposed a more efficient compressor, called firewall compressor [82], that outperforms the TCAM razor. These methods do not consider the use of general wildcard expressions.

Wildcard-based compression for ACLs is considered in [83]. It uses a heuristic that runs in polynomial time which is based on two functions: bit swapping and bit merging. However, this method is less efficient than other ACL compressors. Wildcard compression of ACLs using logic minimization is considered in [84]. It achieves a significant compression ratio for both artificial and real firewall rules but it is rather time-consuming. Compression of ACLs differs from compression of FIB entries. ACL rules are more complex than FIB entries, but then number of entries in a FIB in carrier networks exceeds the number of entries in ACLs by far: typical ACLs contain a few thousand entries while current BGP Routing Information Bases (RIBs) contain more than 500,000 entries. In addition, compression of FIBs has stricter runtime constraints because FIB updates must be processed faster than ACL updates.

Routing tables of virtual routers are commonly stored in the hypervisor. Running multiple virtual routers on a single physical router increases its TCAM requirement. To limit that requirement, the authors of [85] compress multiple virtual FIBs using merged prefix trees. Up to 14 virtual FIBs can be stored in one TCAM and incremental routing updates are possible.

2.8 Existing FRR Mechanisms for the SDN Data Plane

In this section, we discuss related work with regard to SDN networks. We also discussed FRR mechanisms in decentralized networks in Section 2.3. Note that there are various approaches approaches to increase the reliability of the SDN control plane. The authors of [86] provide an extensive overview of this context. However this work focuses on the data plane resiliency of SDN-based networks.

The reconfiguration potential of SDN networks that do not leverage FRR was investigated in [87]. The authors analyzed the reconfiguration in carrier-grade networks. Their results show that the controller is able to reconfigure their testbed within 80 and 130 ms when network failures occur. In addition, they observed that the reconfiguration speed depends on the number of flows affected, path lengths and traffic burst in the control network. As conclusion they hint that reconfiguration times can be significantly higher in larger networks and FRR can assist reconfiguration in large networks notably.

Local failure protection for SDN is considered in [88]. They introduce a 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 [89]. 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.

SPIDER [90] leverages additional state in the OpenFlow pipeline to provide an alternative and more adaptive solution to OpenFlow's fast-failover mechanism. Their path layout is similar to MPLS crankback routing and optimizations are possible. The SlickFlow approach [91] provides resilience in datacenter networks (DCNs) using OpenFlow and is based on source routing. Primary and alternative backup paths are encoded in the packet header. OpenFlow switches are leveraged on the edge to encode the packet header and the core network is comprised of non-OpenFlow switches that are able to

interpret the header in line-speed. The authors showed positive benefits in a virtualized testbed on DCN topologies. The authors of [92] propose to encode the failure location in an additional label. When traffic is rerouted, switches select a backup path appropriate for the encoded failure. MRCs have been adopted for SDN in [93]. They are based on a similar principle as MRTs but require more state in the network.

In [94] a reliability analysis of datacenter networks is given. The authors investigate the reliability potential of three datacenter 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.

Independent trees (ITrees) can provide resilient multipath routing by sending traffic over one tree and switching in failure cases to the other. Independent Directed Acyclic Graphs (IDAGs) [95] are an extension to independent trees that augment ITrees by directing and using more topology links to increase failure tolerance. IDAGs operate on a similar structure as MRTs but the authors did not compare both mechanism. In [96], computation of node-resilient colored (independent) trees is investigated and optimized for fast convergence times.

2.9 Existing IP and SDN-based Resilient Multicast Approaches

There are various approaches to construct node-redundant pairs of multicast trees [95, 97–99] that can be used to implement a 1+1 protection scheme for multicast traffic. These approaches differ by the considered objective function such as cost, bandwidth, computation complexity, required network state, and update complexity. In Chapter 5, we leverage the routing topologies of MRTs [100] as node-redundant pairs of multicast trees for a SDN-based multicast protection mechanism. The Parallel Redundancy Protocol (PRP) [101] and the High Availability Seamless Redundancy (HSR) protocol [102] provide node-redundant multicast trees that suffice hard real-time constraints in industrial Ethernet networks.

Reliability for multicast traffic can be implemented using acknowledgments (ACKs) and selective repeat transmissions similar to Transmission Control Protocol (TCP). However, the number of ACKs of many receivers likely overburden a source. Therefore, the Reliable Multicast Transport Protocol (RMTP) [103] and other approaches [104, 105] use a shared ACK tree structure to overcome this problem.

MPLS is currently deployed in many ISP networks and provides multicast services with point-to-multipoint (P2MP) LSPs [106]. Such P2MP services support FRR by local repair when RSVP-TE is used. However, these solutions can be unsuitable when multicast group memberships change frequently [107].

There are various algorithms and mechanisms to compute traffic engineered multicast trees based on multiple objective functions. An exhaustive survey of existing methods is given in [108]. The approaches are classified by objective functions, constraints, etc. The authors also propose a generalized multi-objective optimization that is based on load-balanced multiple trees. Most works focus on the computing algorithm and do not discuss the required router state in detail.

There are several SDN approaches for IP multicast. In [109], the authors implement IP multicast using VXLAN in datacenters and remove the need for periodic control plane interaction. A highly scalable IP multicast datacenter method is proposed in [110]

which leverages flow aggregation to support large numbers of multicast joins simultaneously. “Dynamic Software-Defined Multicast” [111] reduces control plane complexity in multicast deployments of ISP networks and adds traffic engineering aspects using multiple trees similar to [108]. BIER and most SDN approaches simplify the control plane while supporting frequent multicast changes. BIER introduces header overhead but only requires a topology-dependent number of forwarding entries to accommodate arbitrary multicast flows in transit routers. In contrast, most SDN methods leverage existing header fields but require significantly more state in forwarding devices. An implementation of BIER for SDN is given in [112]. We present in Chapter 5 also an BIER-based SDN architecture and implementation in P4. Our approach implements traffic engineering and FRR that was not achieved in [112]. Finally, an extensive overview of multicast in SDN-based networks is given in [113].

3 Scalability Analysis of SDN-based Inter-Domain Routing

The interest for SDN within carrier networks has greatly increased in the last years and several researchers have analyzed the applicability of SDN and OpenFlow for these networks, e.g., [87]. The main motivation is based on hardware costs and reduced operational complexity due to centralized control. An important task in carrier networks is inter-domain routing which is performed by the Border Gateway Protocol (BGP). Current BGP routing tables hold more than 500,000 entries [114]. That information needs to be installed in OpenFlow switches if misses for flow tables entries should be avoided. Since a flow table miss involves the SDN controller, performance degradation of the flow is expected. However, forwarding tables in OpenFlow switches are usually smaller than those of core routers so that installing all necessary forwarding information is a challenge.

A reason for the small forwarding tables in OpenFlow switches is that they are implemented in Ternary-Content Addressable Memory (TCAM). TCAM allows packet matching in constant time and outperforms software-based packet matching. However, TCAM has high power consumption, a large footprint, suffers from heat generation [115], and TCAM is expensive compared to other memory such as SRAM [85]. Therefore, vendors tend to install small TCAMs into OpenFlow switches, e.g., many OpenFlow switches can handle in practice between 10K and 40K flow entries.

While conventional routers use only prefix-based match fields with Longest Prefix Match (LPM) in their forwarding tables, OpenFlow switches support general wildcard-capable match fields with priorities. Since general wildcard expressions with priorities are more flexible than prefix-based expressions with LPM, compression techniques could reduce the number flow table entries for OpenFlow switches so that they can be better accommodated in the limited TCAM.

In this chapter, we propose to use the Espresso heuristic [116] from logic minimization to compress a minimum-size set of prefix-based match fields generated by the ORTC algorithm [76] into a set of match fields with general wildcards. As the runtime of the Espresso scales exponentially with the input size, we suggest methods to keep the time for the compression of routing tables low. We apply this method to BGP routing tables of 2013 to quantify the compression potential for flow tables entries in carrier networks. We further analyze the structure of the compressed outcome and the tradeoff between runtime of the compression algorithm and its compression rate.

The content of this chapter is mainly taken from [5] and organized as follows. We first describe the structure OpenFlow-based carrier networks operate and how it interacts with necessary protocols. We discuss how other approaches address the intra-domain routing use case for SDN. Then, we discuss how we perform wildcard compression on IP forwarding tables, address compression times, and incremental updates. Finally, we provide a study that investigates the effectiveness of wildcard compression on inter-domain routing.

3.1 OpenFlow-Based SDN for Carrier Networks

Carrier networks require inter-domain routing information. Usually, one or several BGP routers serve as BGP speakers and collect such information via BGP from neighboring domains. To handle large routing tables and frequent BGP updates, such routers require additional CPU and memory resources as well as additional space in their forwarding table which increases their cost.

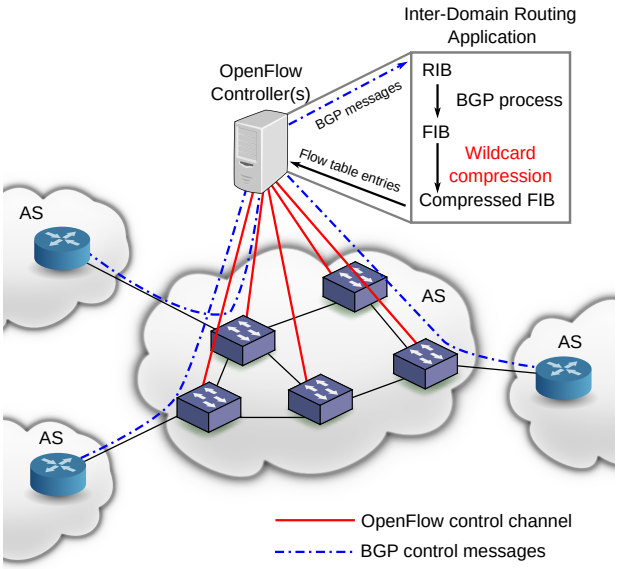


Figure 3.1: Inter-domain routing with OpenFlow-based SDN: a commodity server acts as a BGP speaker and collects the BGP runs in the control plane on commodity servers as SDN application or controller module.

In addition, state distribution within a single carrier network can cause scalability and stability issues and, thus, the authors of [117] proposed a centralized BGP control plane called Routing Control Platforms (RCP) in 2005. This idea was reconsidered for OpenFlow-based SDN in [118] and [119]. Figure 3.1 illustrates that idea. Expensive routers are removed from the network and replaced by simpler and less expensive OpenFlow switches. The switches forward BGP control messages such as prefix announcements and withdrawals to the OpenFlow controller. This can be achieved by installing forwarding entries that match on BGP information in the packet header. The controller

passes them to the BGP application. This application interacts with BGP speakers of neighboring domains and performs the BGP decision process. That means, it combines the inter-domain routing information from BGP and intra-domain routing information into a forwarding table for each switch in the network such that each prefix is associated with an appropriate next-hop. That information is generally compressed using prefix aggregation before being configured by the controller in the switches.

It is likely that the forwarding information cannot be completely included in a switch. Therefore, the authors of [68] take advantage of the fact that approximately 80% of the Internet traffic is caused by 20% of the IP prefixes so that the frequently used prefixes can always be installed in TCAM. Less frequently used prefixes are kept in the control plane but are not installed in the switch. If the switch receives a data packet for which it has no matching prefix, it forwards the packet to the controller that knows how to handle that packet. This approach generally degrades network performance. The approach presented above may be improved through compression of forwarding tables by combining appropriate forwarding rules through general wildcards in match fields that are supported by OpenFlow switches. In an SDN context this is easily feasibly as the control plane consists of cheap commodity hardware so that sufficient CPU power is available for computation-intensive wildcard compression. As a result, more entries fit into the limited TCAM and less traffic needs to be forwarded via the controller.

In [120], a software-defined Internet exchange point (SDX) is presented as well as a method to reduce the BGP state using virtual next-hops and addresses. Our method is orthogonal to this approach. Routing table updates must be propagated fast so that there may not be enough time for the compression of the entire FIB after the routing change. The majority of the routing table entries are quite stable but routing table updates occur frequently [121]. A compression method should be able to perform updates quickly which may happen incrementally on the existing compressed table. This also holds for the SDN context.

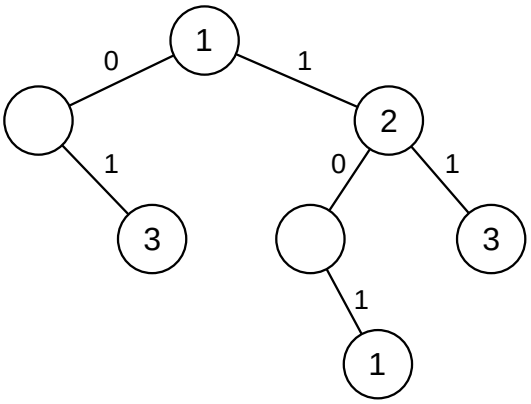


Figure 3.2: The prefix tree for the routing table in Table 3.1a.

3.2 Methodology

In this section we briefly describe the ORTC algorithm that provides the input and benchmark for our proposal. Then, we discuss LPM that is used in IP networks and wildcard matching using priorities. We present how prefixes can be compressed using the Espresso logic minimization heuristic. Finally, we discuss how Espresso can be incrementally applied and how routing table updates can be performed.

3.2.1 Prefixes, Forwarding Tables, and Longest-Prefix Match

An IPv4 prefix p consists of a 32-bit IP address and a 32-bit network mask. The network mask starts with a series of ones in the most significant bits followed by a series of zeroes for the least significant bits. The number of leading ones is the length of the prefix p which is usually indicated by the notation $length(p)$. As an alternative, a prefix can also be represented by a match field containing the $length(p)$ bits of the IP address followed by wildcard symbols '*', also known as 'don't cares', for each remaining positions in the address. Table 3.1a contains prefixes in that notation. A prefix matches an address if the leading $length(p)$ bits of the prefix address equal the first $length(p)$ bits of the address to be matched.

A routing or forwarding table associates a set of prefixes with a next-hop. It can be

FIB entry	Next-hop
***	1
101	1
1**	2
01*	3
11*	3

(a) Prefix-based forwarding table.

FIB entry	Next-hop	Priority
***	1	0
101	1	3
1**	2	1
1	3	2

(b) Forwarding tables using general wildcard expressions.

Table 3.1: Two forwarding tables with identical forwarding behavior.

represented by a prefix tree. Figure 3.2 illustrates the prefix tree for the routing table in Table 3.1a. Each entry in the routing table is represented by a labeled node in the tree whereby the label indicates the next-hop. The prefix for such an entry is composed of the numbers on the edges from the root to the corresponding node. The root of the tree corresponds to the least specific prefix $***$. If a node is empty, the corresponding routing tables has no entry associated with that prefix.

Forwarding in IP networks uses LPM. If multiple entries in a forwarding table match an address, the most-specific one is used, i.e., the one with the longest prefix. To perform that procedure, the prefix tree is traversed according to the bits in the address as far as possible. Thereby, a '0' or '1' in the address denotes a move to the left or right child in the tree. The last visited labeled node corresponds to the most-specific forwarding entry. The ORTC algorithm compresses a routing table with general prefixes into a minimum-size prefix-based table using prefix trees. Details are given in [76].

3.2.2 Wildcards Expressions and Logic Minimization

A wildcard expression w is a match field with wildcards '*' in arbitrary positions of the match field so that a prefix can be viewed as a very special wildcard expression. Wildcard expressions can be used in forwarding entries in OpenFlow. However, there are no analogue mechanisms for wildcard-based FIBs like prefix trees and LPM. Therefore, forwarding entries in OpenFlow require priorities to decide which of them is to be used

when several of them match an address. We denote the priority of a wildcard expression as $prio(w)$.

Logic minimization compresses a set of logical expressions into an equivalent set of logical expressions that covers the same on-set. The Espresso heuristic was developed for logic minimization in the context of very-large-scale integration (VLSI) synthesis. It does not guarantee minimum size results but is faster and requires less memory than exact minimizers [116]. We denote $E(\mathcal{P})$ as the minimized result of the Espresso on the prefix set $\mathcal{P}$.

As wildcard expressions can be interpreted as logical expressions, we can compress a set of wildcard expressions $\mathcal{P}$ into a possibly smaller set of wildcard expressions $E(\mathcal{P})$. As a consequence, we can also minimize a set of prefixes $\mathcal{W}$ into a smaller set of wildcard expressions $E(\mathcal{W})$.

3.2.3 Compression of Forwarding Tables Using Logic Minimization

We apply logic minimization for compression of prefix-based forwarding tables. A naive approach is to combine all prefixes with the same next-hop. As a result, the prefixes '01*' and '11*' with next-hop 3, i.e., ('01*', 3) and ('11*', 3), can be combined to forwarding entry ('*1*', 3) and the forwarding entry ('1**', 2) remains unchanged. It is important to assign higher priority to entry ('*1*',3) than for ('1**',2) to preserve the forwarding behavior. Otherwise, packets addressed to addresses '11*' would be forwarded to the wrong next-hop. A problem occurs if we combine ('***', 1) and ('101', 1) to ('***', 1) because then either packets destined to '1**' or packets destined to '101' are forwarded to the wrong next-hop, depending on whether ('***', 1) or ('1**', 2) is assigned higher priority. This happens because prefix '***' contains the other prefix '101' and the other node ('1**', 2) with a different next-hop lies on the path between ('***', 1) and ('101', 1). Thus, such situations need to be avoided in sets of prefixes $\mathcal{P}$ with same next-hops that are compressed by Espresso.

To avoid the mentioned problems, we partition all entries in a forwarding table into sets with equal next-hops and equal prefix length. Thus we compress all prefixes $\mathcal{P}_i^j$

with the same next-hop j on the same level i of a prefix tree with Espresso to a set of wildcard expressions $E(\mathcal{P}_i^j)$. The priority assigned them is the original prefix length, i.e., $prio(w) = i : w \in E(\mathcal{P}_i^j)$. These priorities assure the same forwarding behavior as LPM on the prefix-based forwarding table. As the proposed compression approach cannot compress forwarding entries with different-length prefixes, we first run the ORTC on the original forwarding table and use a minimum-size prefix tree as input to the described procedure.

We have investigated two other methods to define sets of forwarding entries to be compressed and appropriate priorities. These sets contained different-length prefixes. Evaluation results showed that these approaches lead to less compression so that we do not consider them in this work.

3.2.4 Compression Speedup

The runtime of the Espresso algorithm is exponential with regard to the number of input expression. Therefore, the compression is slow for a large set of prefixes to be compressed. To reduce the compression time, we propose a maximum set threshold T. We partition the sets $\mathcal{P}_j^i$ into smaller sets with at most T forwarding entries, taken consecutively from the minimum-size prefix tree when going from left to right on the same level. These subsets of prefixes are individually compressed into sets of wildcard-expression by Espresso. They lead to less compression potential but to shorter overall compression time compared to the approach without threshold.

3.2.5 Incremental FIB Updates

Routing table updates consist of prefix additions and removals. This also holds for changed next-hops in the forwarding table. ORTC is fast and can be used to quickly compute a new minimum-size prefix-tree. The changed nodes result in prefixes that need to be removed or added to the wildcard-compressed forwarding table. Adding prefixes is simple as they do not even need to be compressed; further optimization is possible.

We illustrate a procedure to remove a prefix p from a set of wildcard expressions $\mathcal{W}$

by an example. Consider that the prefix $p = `10011*'$ of length 5 has to be removed from a set of wildcard expression $\mathcal{W}$. Let $w = `1 * 0 * 1*'$ be the only expression of priority 5 that matches p. We split w into a set of smaller wildcards $\mathcal{W}' = \{110 * 1*, 10001*, 10011*\}$ and remove the prefix p from this set. The updated set of wildcard expressions is then $\mathcal{W}^* = \mathcal{W} \setminus w \cup (\mathcal{W}' \setminus p)$.

The resulting sets of wildcard expressions are not minimal but did not require time-consuming Espresso minimization. Improved results can be obtained after another Espresso minimization $E(\mathcal{W}^*)$.

3.3 Results

In this section we apply the presented compression algorithms to realistic data sets. We evaluate and compare their compression ratios, analyze the structure of the compressed data, and study the algorithm runtime depending on the threshold parameter.

3.3.1 Investigated Data Set

As a base for our study, we use a RIB with 500,495 IP prefixes from the year 2013 obtained from the Route Views project [114]. The RIB is converted into a pseudo-FIB by assigning a next-hop to each prefix. To that end, we assume up to n_{hops}^{next} different next-hops. Each prefix is assigned one of these hops with equal probability. We compress that FIB using the ORTC algorithm which yields a minimum size prefix-tree for further wildcard compression. For each number of next-hops n_{hops}^{next} we generate 10 random FIBs and report mean values for presented results. We also investigated data sets from prior years (2009 – 2012) but we do not show those results as they do not add further insights.

3.3.2 Compression Ratio

We first quantify the compression potential of FIBs in carrier networks through wildcard compression by considering the compression ratios achieved by the heuristic approaches. Figure 3.3 compares the number of FIB entries generated by ORTC with the number of FIB entries after further wildcard compression for different numbers of next-hops n_{hops}^{next}. The number of FIB entries obtained after pure prefix compression by ORTC increases with increasing number of next-hops from 50,000 to 350,000. For a single next-hop, there are about 50,000 FIB entries instead of a single default route. The reason for that is the assumption of a full routing table so that packets need to be dropped in the absence of a matching entry. The figure also provides values for wildcard compression with Espresso with maximum set thresholds of $T = 100$ and $T = \infty$. We observe that this further compression reduces the number of FIB entries by 30,000 – 40,000 in the presence of at least $n_{hops}^{next} = 2$ next hops. Furthermore, Espresso yields hardly more

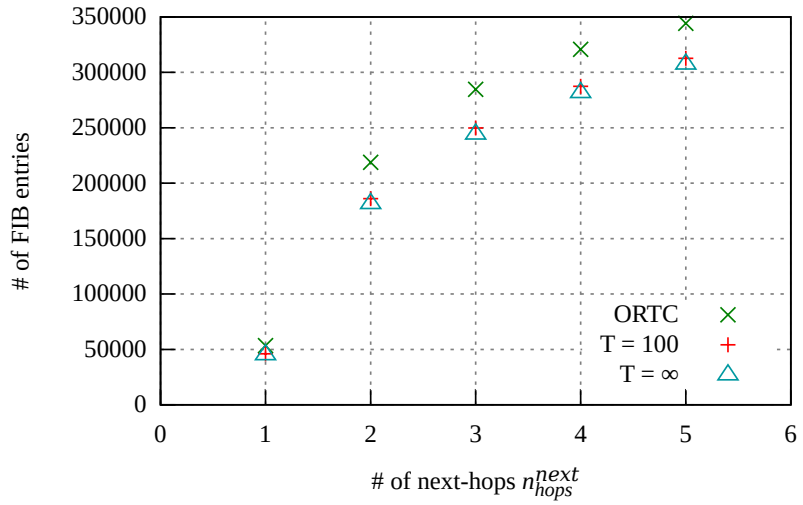


Figure 3.3: Number of FIB entries after pure prefix compression by ORTC and for further wildcard compression by Espresso with and without maximum set threshold T.

FIB entries when applying a low maximum set threshold of $T = 100$ than without such a threshold ($T = \infty$).

As the loss of compression due to the maximum set threshold T is hardly visible in Figure 3.3, we present the compressed fraction ρ in Figure 3.4, i.e., the fraction by which further wildcard compression can reduce the size of the minimum prefix-based flow table. The compressed fraction ranges between 9% and 17% and depends both on the number of next-hops n_{hops}^{next} and the applied maximum set threshold T. The compressed fraction decreases with decreasing threshold, but the loss in compression is at most 2% even for a very low maximum set threshold of $T = 100$. A threshold of $T = 500$ leads only to 1% loss in compression.

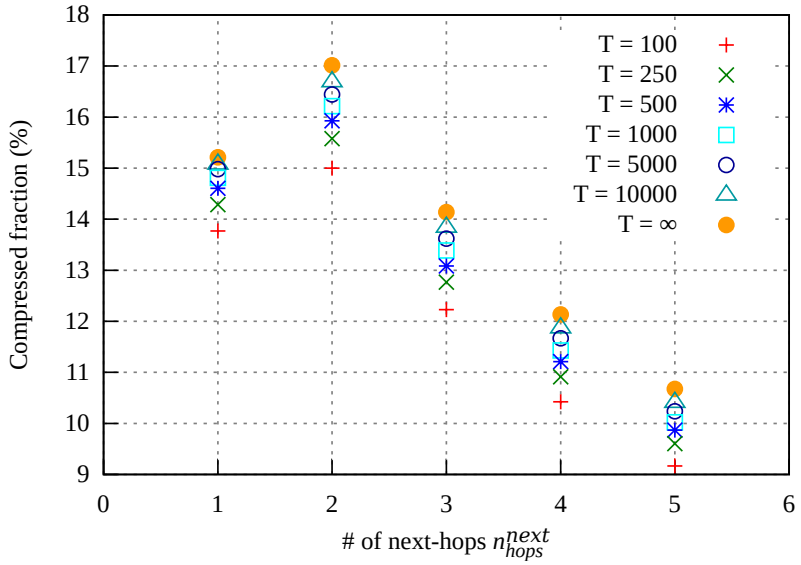


Figure 3.4: Fraction of FIB entries compressed by Espresso depending on the maximum set threshold T.

3.3.3 Analysis per Prefix Length

Figure 3.5 shows the number of FIB entries generated by ORTC sorted by prefix length. Most prefixes have length 24. Smaller prefixes down to prefix length 17 appear with decreasing frequency. Prefix length 16 is about as frequent as prefix length 18 and again, smaller prefixes down to prefix length 14 appear with decreasing frequency. All other prefix lengths hardly occur. The reason for this phenomenon is that standard prefix lengths are /16 and /24 which are already combined to shorter prefixes by Classless Inter-Domain Routing (CIDR) in the BGP RIB or by the ORTC through prefix aggregation.

The figure also shows the number of FIB entries per prefix length after further wildcard compression and the number of FIB entries containing additional wildcards. The number of more general wildcard expressions is rather low and most of them contain only a single wildcard. So the compression potential is rather moderate.

We applied the Espresso heuristic to equal-priority sets of prefixes with equal length. Figure 3.6 shows the compressed fraction per prefix length. For prefix length between 8

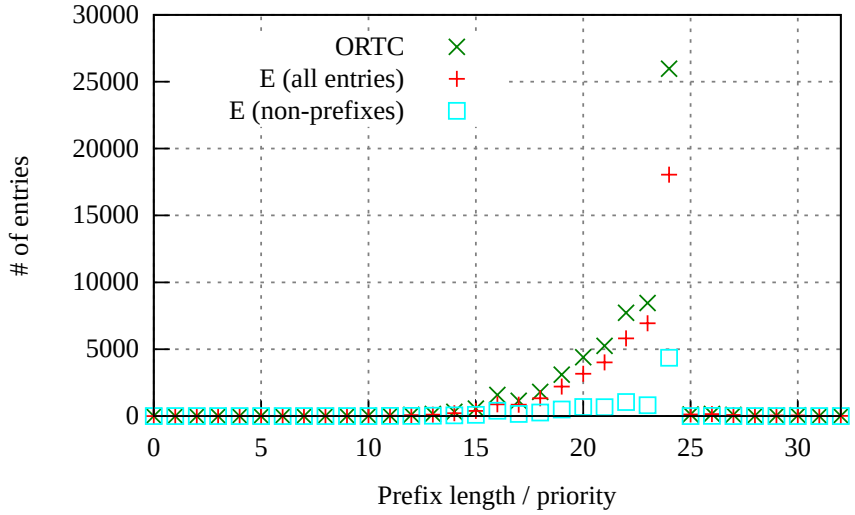


Figure 3.5: Number of FIB entries per prefix length for ORTC and after wildcard compression by Espresso without threshold ($T = \infty$). For Espresso, the number of FIB entries with wildcards is also given.

and 24 it is mostly in the order between 10% and 15%. Smaller and larger prefixes are extremely rare so that their overall fraction amounts to less than 0.1%.

This suggests that Espresso can compress efficiently only if the set FIB entries to be compressed is sufficiently large so that there is a chance for similar entries. We have also studied other approaches to create equal-priority sets that do not reveal equal-length prefixes, but the achieved compression ratio was lower. Apparently, the compression potential is larger for sets of equal-length prefixes.

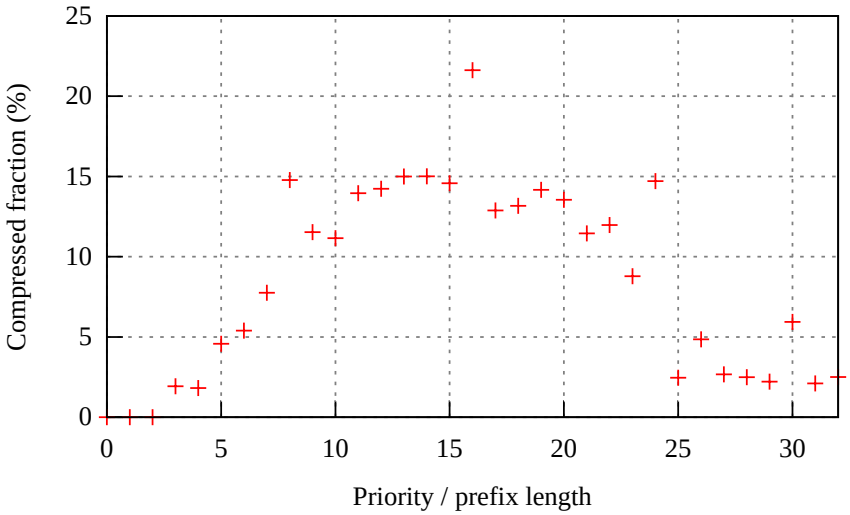


Figure 3.6: Compressed fraction per prefix length for wildcard compression with Espresso without threshold ($T = \infty$).

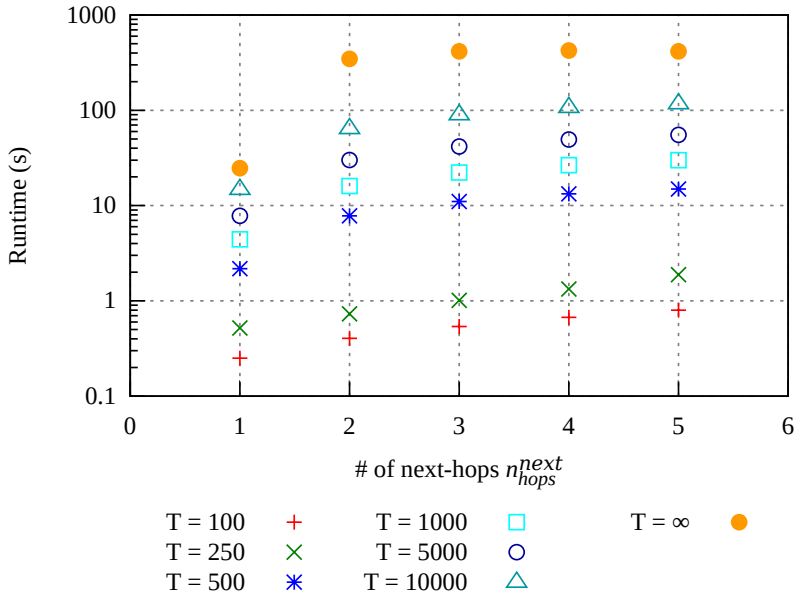


Figure 3.7: Runtime of Espresso-based compression for complete FIBs depending on the maximum set threshold T.

3.3.4 Compression Time

We run Espresso on a computer with 8GB DDR3-RAM and an Intel CPU i5-2500K with 4 cores, 6 MB cache and 3.3 GHz. We implemented the compression in a single threaded C++ program that only uses one core. We measure the runtime for the Espresso-based FIB compression only, i.e., the partitioning work of the input FIB in smaller subsets is provided by a different program.

Figure 3.7 shows the measured runtime of Espresso for various numbers of next-hops n_{hops}^{next} and for various maximum set thresholds T. Without such a threshold ($T = \infty$), the compression for a single next hop takes about 22 s, but for more next-hops the runtime increases from 341 s to 436 s. The runtime is greatly reduced by applying maximum set thresholds. For moderate thresholds of $T = 500$ the Espresso runtime decreases to a range between 7 s and 11 s. Low thresholds of $T = 100$ and $T = 250$ require even for $n_{hops}^{next} = 5$ a maximum compression time of less than 1 s or 2 s. Thus, the application of maximum set thresholds reduces the compression time by orders of magnitude while degrading the compressed fraction by 1% – 2%.

3.4 Summary

We presented a concept for the use of OpenFlow-based SDN in carrier networks. In some design variants, OpenFlow switches need to accommodate the full inter-domain routing information which is rather challenging as flow table size is limited. As OpenFlow supports match fields with general wildcards, we propose to compress prefix-based FIB entries into FIB entries using general wildcards using the Espresso heuristic. This is feasible with SDN as the computation-intensive compression can be performed by a network application on a server instead by the switch.

We showed that our proposed method reduces FIB sizes based on a BGP RIB of 2013 with about 500,000 entries by up to 17%. As the runtime of Espresso is exponential with regard to input size, we propose a maximum set threshold to trade runtime against compression rate. Thereby, the compression time could be reduced by orders of magnitude down to less than 1 s or 2 s while sacrificing 1% – 2% compression ratio. Routing updates can be added incrementally.

This compression potential is certainly not large enough to fit the large amount of routing information into the TCAM of a today's custom OpenFlow switch to enable forwarding without misses of flow table entries. However, the method may be used to compress a rather stable part of the flow table to leave more room for other less frequently used flow table entries.

4 Design and Analysis of Protection Methods for OpenFlow-based SDN

In this chapter, we address the robustness of SDN against network failures. Its content is mainly taken from [4, 6–8] and organized as follows.

We explain the need for protection against network failures in SDNs that are caused by the design of SDN and OpenFlow. We motivate the requirements of protection mechanisms in SDN and outline the differences to decentralized networks. We propose various approaches how such protection against network failures can be realized. First, we adopt LFAs for SDN. We propose a novel loop detection mechanism for LFAs that allows to increase their protection coverage. This approach – called LD-LFAs – only introduces a single forwarding entry per switch but cannot guarantee 100% protection against all single link failures. Thus, we discuss MRTs and MPLS FRR as candidates for FRR in SDN. We propose destination-specific MRTs (dMRTs) and dual sink trees (DSTs); modified variants of MRTs that increase computational complexity but improve the path layout without introducing additional state in the devices. Finally, we propose a resilience mechanism that combines traffic engineering and protection. Load-Dependent Flow Splitting (LDFS) leverages monitoring capabilities in OpenFlow switches to detect traffic overloads and to redirect the overload on the backup paths. We discuss and describe the implementation of all proposed mechanisms in OpenFlow.

We provide three different studies to measure the effectiveness of these mechanisms. We analyze the protection coverage of loop-detecting LFAs (LD-LFAs) compared to LFAs for wide area and datacenter networks. We compare MRTs, dMRTs, DSTs, and MPLS-FRR with regard to path lengths, required network capacities, and forwarding state. We investigate the impact of load-balancing overloads on backup paths and their implications on selected topologies. We conclude this section by discussing the applicability of the proposed mechanism in SDN and if appropriate in decentralized networks. We summarize them based on their strengths and shortcomings and give recommendations based on network requirements.

4.1 Motivation

We described how network failures are handled in communication networks in Chapter 2. Section 2.3 discussed existing IP and MPLS and Section 2.8 SDN-based FRR mechanisms. The principles of network reconfiguration and pre-established backup paths are applied to SDN as well. However, the handling of failures differs for SDN. Forwarding devices do not run decentralized routing protocols and depend on the controller to install appropriate forwarding entries. If a failure occurs, the detecting switch informs the controller which in turn computes new forwarding entries and installs them on the switches. In [87], the authors measured on a real testbed that their controller reacts within 80 – 100 ms to network failures. They also identified that the restoration time depends on the path lengths and number of flows to be restored. Larger networks are expected to have longer restoration times. However, since the control channel may be part of the data plane, reconfiguration may be difficult to perform in a timely manner. FRR support is available for SDN since OpenFlow 1.1. In our opinion, FRR is very important for SDN to mitigate the effects of failures, especially when control elements observe simultaneously high control load and might not be able to repair failures quickly.

The requirements on FRR differ for SDN and decentralized networks. In the latter, network devices individually compute their forwarding entries and require dedicated computational resources which may lead to high costs for large networks. Thus, keeping the complexity of the deployed algorithms as low as possible is desirable. Therefore, algorithms for IP FRR [122] generally strive to minimize computation time. For SDN, algorithms are performed on controllers that generally provide more computation power allowing for more computational intensive approaches. Additionally, OpenFlow requires TCAM to enable wildcard matching on header fields. Because TCAM is expensive and has a large footprint, OpenFlow switches are expected to offer little space for backup path entries. Forwarding state is also of concern in decentralized networks but the lack of space seems to be a more severe issue for TCAM-based forwarding devices.

Adopting existing resilience mechanisms to SDN-based networks can be advantageous. Standardized mechanisms are often tested in real networks and are known with regard to operational complexity. Usually they are already implemented in existing hardware. This increases the probability that similar hardware acceleration may be introduced in SDN devices, too. In hybrid SDN networks, a resilience scheme that is supported in the existing network allows for a common resilient forwarding plane. This allows to introduce SDN devices more easily in existing networks where resilience is an important factor. Moreover, additional computational resources in the SDN controller can be used to improve existing mechanisms.

With these reasons in mind, we propose to adopt LFAs [19] from IP networks as a resilience mechanism for SDN. LFAs do not require additional forwarding entries but cannot protect against all single link failures. We develop a novel loop detection for LFAs that increases their coverage in Section 4.2. This extension only requires one additional forwarding entry per switch and some bits in the packet header. This approach addresses SDNs that require FRR but cannot hold additional backup entries.

In addition, we propose to leverage Maximally Redundant Trees [100, 123] which are also standardized in the IETF like LFAs. In contrast to LFAs, MRTs provide full protection against single link and single node failures if the topology is redundant enough. To facilitate the protection, MRTs require a number of additional forwarding entries that

scale with the number of nodes in the network. Two additional forwarding entries are required for each default hop. Note that the computation of MRTs is optimized for IP networks by reducing the overall complexity of the algorithm. In an earlier work [27], we found that MRTs can lead to excessively long backup paths. Therefore, we adjust the MRT computation to avoid long backup paths and present this approach in Section 4.3. We denote this approach as destination-specific MRTs. The algorithmic complexity is significantly increased but does not require changes to the forwarding mechanism or state requirements. In addition, we propose dual sink trees (DSTs) that are based on the same principles of dMRTs but remove the default shortest path in the failure-free case. We also address MPLS FRR in this work – another IETF mechanism for failure protection. We do not optimize MPLS but analyze in this chapter if MPLS is suitable for SDNs and its performance compared to (d)MRTs.

Finally, we propose an original protection method that combines traffic engineering with failure protection for OpenFlow-based SDN. This work is motivated by the following two arguments found typically in inter-domain networks. Firstly, in [124], the authors address the duration of network failures. They found that failures are typically short-lived and generally do not persist over long time periods. Secondly, traffic overloads can often be observed through inter-domain routing effects [125] or through temporary events such as live-streaming of popular events. We design a novel forwarding behavior that both protect against network failures and handles temporary overloads called Load-Dependent Flow Splitting (LDFS) in Section 4.4. LDFS requires and combines several OpenFlow features: fast-failover, QoS, traffic monitoring and traffic engineering methods. The basic idea of LDFS is to apply load balancing to a flow or an aggregate based on the current load of this flow. Traffic is generally sent over a single primary path. If the load of the flow exceeds a previously defined threshold, traffic is distributed on multiple paths towards the destination. If a network failure occurs, traffic is sent on the secondary paths.

4.2 Loop-Detecting Loop-Free Alternates (LD-LFAs)

In this section, we explain simple LFAs and propose a novel loop detection mechanism for LFAs. Then, we discuss their implementation in OpenFlow. We conclude this section with an explanation how the loop detection method operates with header of specific sizes.

4.2.1 Computation of LFAs

LFAs [19] were proposed by the IETF for IP FRR. LFAs are simple but 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 Equal-cost multi-path (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 $\mathrm{dist}(u, v)$ to determine whether neighbors fulfill these conditions. No additional computation is required because $\mathrm{dist}(u, v)$ is already required for the primary hop computation. We will explain the conditions by example with the network shown in Figure 4.1. 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

$$\mathrm{dist}(N, D) < \mathrm{dist}(N, S) + \mathrm{dist}(S, D) \tag{4.1}$$

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

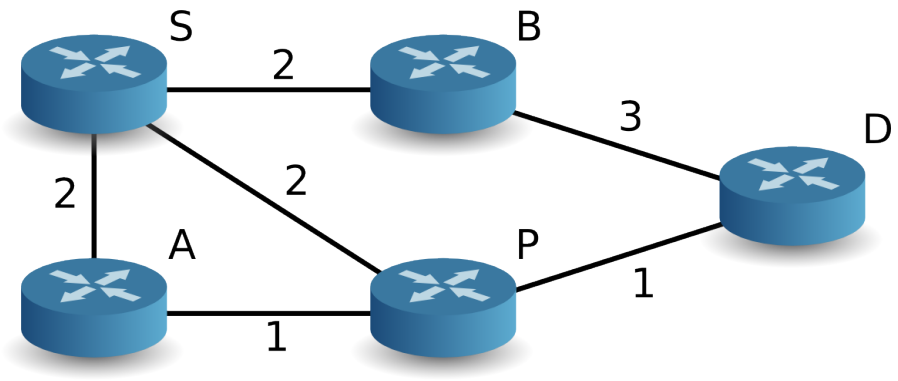


Figure 4.1: Example network for LFA computation.

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

$$\text{dist}(N, D) < \text{dist}(N, P) + \text{dist}(P, D). \tag{4.2}$$

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:

$$\text{dist}(N, D) < \text{dist}(S, D). \tag{4.3}$$

Node B is not downstream ($3 \not< 3$) and packets would loop between B and S when both $S \rightarrow P$ and $B \rightarrow D$ 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 $A \rightarrow P$ 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).

4.2.2 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.

4.2.2.1 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 anLFA available but it is not allowed to be chosen due to the more restricted condition.

OurLFA 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 4.1. Consider that packets are sent from S to D and the links $S \rightarrow P$ and $A \rightarrow P$ have failed. Packets have to be redirected over A because there is noLFA of category (a) and A is anLFA 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 anLFA 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 4.2.2.3 and analyze the impact of various bit lengths in Section 4.5.2.4. We refer LFAs with loop detection to LD-LFAs.

4.2.2.2 Implementation in OpenFlow and State Requirements

The described the basics for resilience mechanisms in OpenFlow in Section 2.5. We illustrate how LFAs are implemented in OpenFlow by the example given in Section 4.2.2.1. 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 to implement 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". We want to emphasize that this loop detection mechanism only requires this one additional flow table entry that detects loops per switch. Therefore, we consider LFAs with loop detection as a lightweight resilience mechanism.

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.

4.2.2.3 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.

Note that, the proposed algorithm can be changed very easily in an SDN environ-

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

  $v \leftarrow \mathrm{argmax}_{v \in \mathcal{U}} (\delta(v))$;

  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**

      $cif[c[u]] \leftarrow cif[c[u]] + \frac{1}{\mathrm{dist}(v,u)}$;

    end

    // assign least interfering color

    $c[v] \leftarrow \mathrm{argmin}_{0 \leq j \leq n_b} (cif[j])$;

  end

  $\mathcal{C} \leftarrow \mathcal{C} \cup \{v\}$;

  $\mathcal{U} \leftarrow \mathcal{U} \setminus \{v\}$;

  $i \leftarrow i + 1$;

end

ment. The computation runs in a logically centralized control plane and, thus, only a few elements must be updated. The In general, the distance function $\mathrm{dist}(u, v)$ is already computed in the control plane for the primary next-hops which enables more complicated than the presented one for large scale networks.

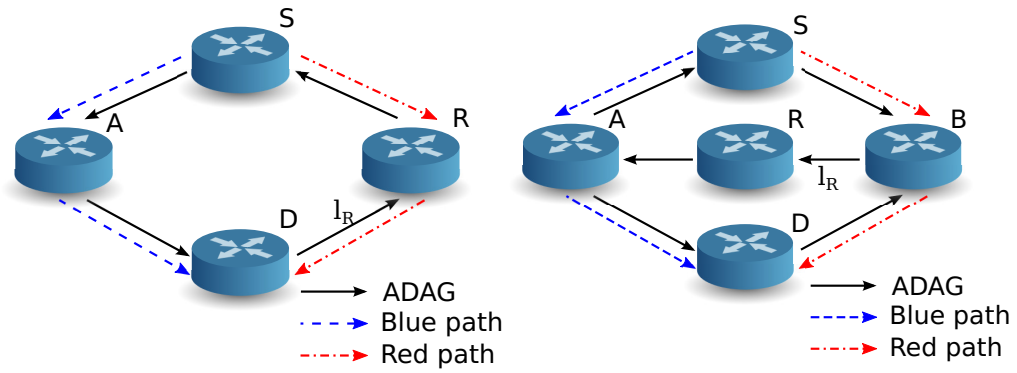
4.3 Destination-based Maximally Redundant Trees (dMRTs) and Dual Sink Trees (DSTs)

In this section, we discuss the adoption of MRTs for protection in SDN networks. We, first explain MRTs and their computational principles as they are standardized in the IETF. We provide insight about the disadvantages of the current form of their computation. We propose dMRTs which leverage an improved path layout for MRTs but does not change the data plane implementation. Finally, we propose DSTs that are based on the same data structure of (d)MRTs but sacrifice path lengths for reduced forwarding state.

4.3.1 Maximally Redundant Trees

Maximally Redundant Trees (MRTs) are standardized in the IETF [100, 123], and can be implemented in IP and MPLS networks to protect against single link and single node failures. The main motivations for MRTs are to achieve 100% failure coverage without introducing too much forwarding state as with previously discussed approaches such as not-via addresses [23] in the IETF. In addition, MRTs should not require much computational complexity since IP routers generally provide limited computational resources.

MRTs achieve these requirements by calculating two routing topologies that are referred to as the red and blue topologies. Packets traversing these additional topologies towards a destination follow maximally redundant paths, i.e., a packet sent on the blue topology from a source S towards destination D traverses different links and nodes than a packet sent on the red topology from S to D if the network topology is redundant enough. If the network topology does not allow for redundant paths, MRTs ensure that only a minimum number of elements are shared by each red and the blue path pair. The backup paths are generally implemented in such a way that only a single additional entry is required in a node per destination and per routing topology. Thus, MRTs scale linear with the number of nodes in the network and require 200% additional forwarding entries which is a desired property of MRTs in the IETF.



(a) Source and destination are part of a single cycle in the ADAG structure.

(b) Source and destination are on two different cycles in the ADAG structure.

Figure 4.2: Construction of red and blue path using the ADAG structure.

4.3.1.1 Computation of Backup Paths

MRTs leverage a special structure to keep computational complexity low. The structure is called Almost Directed Acyclic Graph (ADAG), which is described in great detail in the IETF draft [100], and is computed in linear time [126]. In the following, we briefly outline the ADAG structure and how all next-hops for the red and blue topologies can be inferred from it. An arbitrary node of the network will be the root node R of the ADAG. All (bidirectional) links of the network topology are directed in such a way that every link is part of at least one cycle that contains the root R. Moreover, all cycles must enter the root node through a single link l_R. Removing the link l_R removes all cycles from the structure converting the ADAG in an directed acyclic graph (DAG), hence its name.

The ADAG structure is used to compute two disjoint backup paths. Consider a PLR S and a destination D. Figure 4.2a illustrates the case that both S and D are located on the same cycle. The blue backup path follows the ADAG direction and the red backup path follows the reverse direction. If several cycles exist where S and D are part of, the shortest cycle in terms of link costs is considered for backup path construction. Figure 4.2b shows the case when S and D are not part of the same cycle. Then, they must be part of two different cycles that intersect at least in the root node R and link l_R due to the ADAG properties. The blue backup path first goes against the ADAG direction

towards an intersection node with the cycle that D is part of. From this intersection node, it follows the ADAG towards D. The red backup path is constructed using the reverse direction. At first, it follows the ADAG to a (different) intersection node and from there it goes against the ADAG towards D. These two backup paths for the PLR S are computed using two modified shortest path first (SPF) computations rooted at S. This may not lead to the overall shortest backup paths but red and blue hops can be inferred for all destinations using both cases shown in Figure 4.2.

4.3.1.2 Selection of Backup Paths in Failure Cases

Traffic is normally forwarded on the primary path. When a failure occurs, the PLR determines whether the red or blue backup avoids the failure and sends it along the corresponding path. The computation of this decision is based on the information gathered during backup path construction. The ADAG provides a partial topological order of the nodes and the SPF computations for the backup paths contain information about reachability of nodes by red or blue paths. [100, Figure 25] shows that the construction method utilizing an ADAG ensures that at least one of the backup paths cannot contain the failed element. This path is selected and the traffic is sent to the appropriate color (Section 4.3.4) using tunneling mechanisms.

4.3.2 Destination-specific MRTs

In a previous work [27], we found MRTs can cause excessive path lengths, i.e., up to 16 hops in a 16 node network, depending on the network topology and the selected root node of the ADAG. Therefore, appropriate root node selection is highly significant for MRTs. We think that the reason for long backup paths with MRTs is caused by the way the paths are constructed. We illustrate the reason of long backup paths in Figure 4.3. The shown topology is a two-vertex biconnected network from the Topology Zoo [127]. We select node 2 as root node for MRT computation. In the example, traffic is sent from node 7 towards node 0 on the shortest path and the last hop fails. The shortest cycle that contains the PLR, the destination and the root node is illustrated as the backup path. Since the PLR and the destination are quite distant to the root node, the backup path

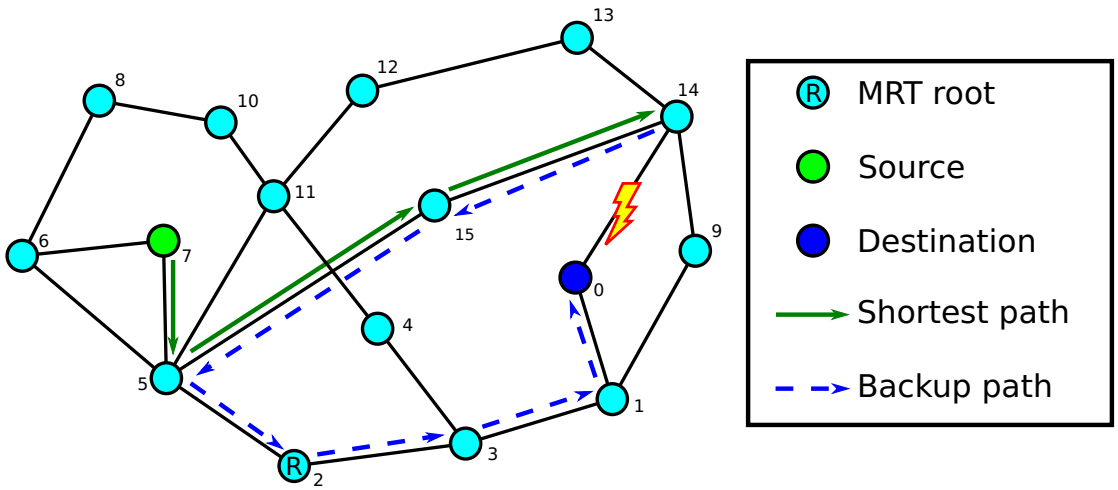


Figure 4.3: MRT backup paths can be long when the ADAG root node is quite distant to the PLR and destination because the backup path often contains the distant root node.

is significantly longer than necessary. This example illustrates why MRTs lead to long paths when PLR and D are on a single cycle (Figure 4.2a). The second case (Figure 4.2b) follows a similar reasoning.

Therefore, we propose to use for each destination D a separate ADAG rooted at D. Then, the PLR and D always are part of the same cycle due to the ADAG construction. Figure 4.4 illustrates the same setup for dMRTs. The shortest cycle of the three nodes (PLR, D, and ADAG root) is significantly shorter than for MRTs since the ADAG root node is the destination. This optimization of MRTs seems kind of obvious to make, yet it has significant implications on the computational complexity on the algorithm. For each of the n possible destinations an ADAG and two backup paths have to be calculated because the root node and thus the structure of the ADAG changes. Note that a high number of SPF calculations is very undesirable for IP networks because each router has to calculate the routes and computing resources are more scarce on each device. However, we think that SDN justifies higher algorithmic complexity in order to reduce backup path length. The computational resources are significantly higher in SDN context since the control plane generally is computed on separate high-performance

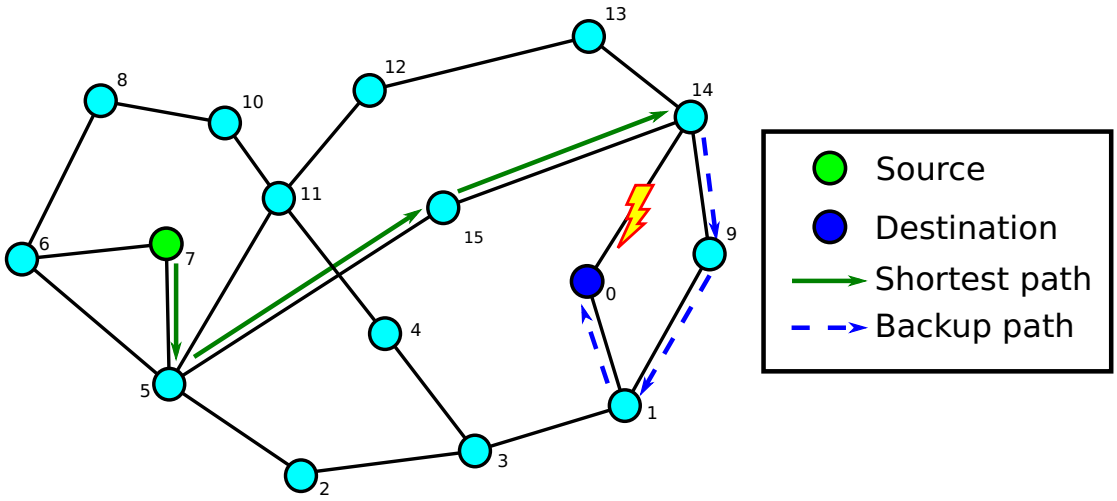


Figure 4.4: dMRTs leverage the shortest cycle of the ADAG rooted at the destination. Backup paths can be significantly shorter than with MRTs.

computers. We discuss this aspect in more detail in Section 4.3.4 below.

4.3.3 Dual Sink Trees

MRTs require three next-hops per destination in every switch. The first next-hop is along the shortest path towards the destination. In addition, a blue and a red next-hop are required for the backup paths. We propose to remove the default shortest path hop to reduce the state required in the network. The remaining red and blue hops are redundant by design of the MRTs and the ADAG structure and form a pair of forwarding trees. We denote this mechanism as dual sink trees (DSTs). A similar mechanism was proposed under the name of Independent Directed Acyclic Graphs (IDAGs) in [95]. It leverages a similar structure to the ADAG to compute the redundant paths. The authors do not propose their approach to SDN or explain how it should be implemented in an OpenFlow-based network. The performance evaluation in [95] does not reflect important aspects with regard to SDN networks. Moreover, they do not compare their approach to MRTs and, thus, the performance difference is not clear in their work. We provide further insight about the performance between MRTs and DSTs in Section 4.6.

Figure 4.5 illustrates the effects of removing the default shortest path. The ADAG

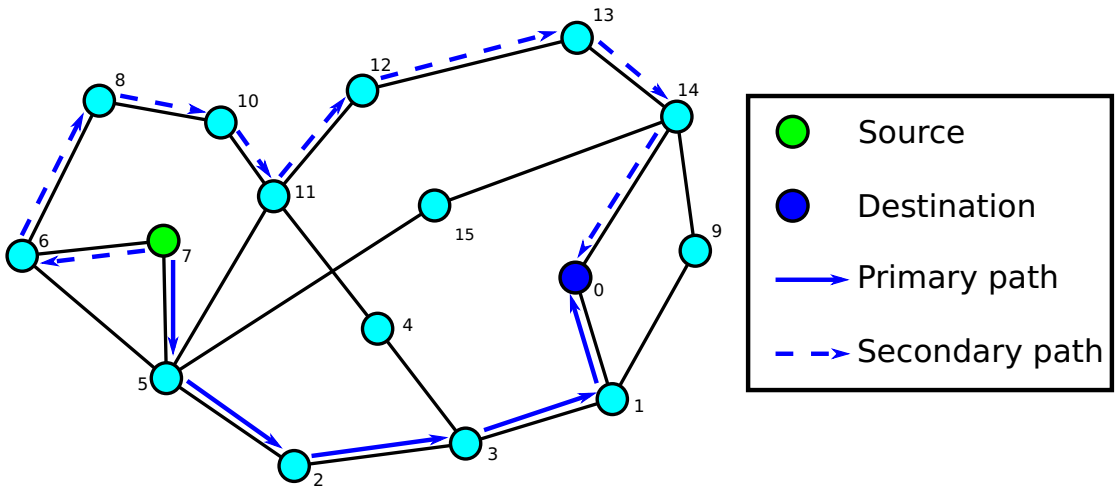


Figure 4.5: The primary and secondary paths for DSTs form a vertex-redundant circle containing the source and the destination. The shorter path is selected as primary path. In this example the primary path is one hop longer than the shortest path.

rooted at the destination is used to find the shortest vertex-disjoint path pair from the source to the destination. It selects the shorter of the two as the primary path that carries the traffic in the failure-free case. This can be either the blue or the red path. In the example, the primary path has length 5 but the shortest path (Figure 4.4) only length 4. We investigate the impact of the path prolongation in Section 4.6.5.

Figure 4.6 shows how failures are handled with DSTs. Packets are forwarded on the primary path to the PLR. The PLR switches the color of the packet. The packet is then sent on the shortest circle containing the PLR and destination. In the example, the backup path is as short as for dMRTs. Because dMRTs can either select the red or the blue path, dMRTs backup paths may be shorter than for DSTs. However, switching packets from one color to another can cause looping when multiple failures occur. Therefore, we require a mechanism that tracks the number of redirects and drops the packet to avoid the potential forwarding loops of multiple redirects. We describe this mechanism and its realization in OpenFlow in the following section.

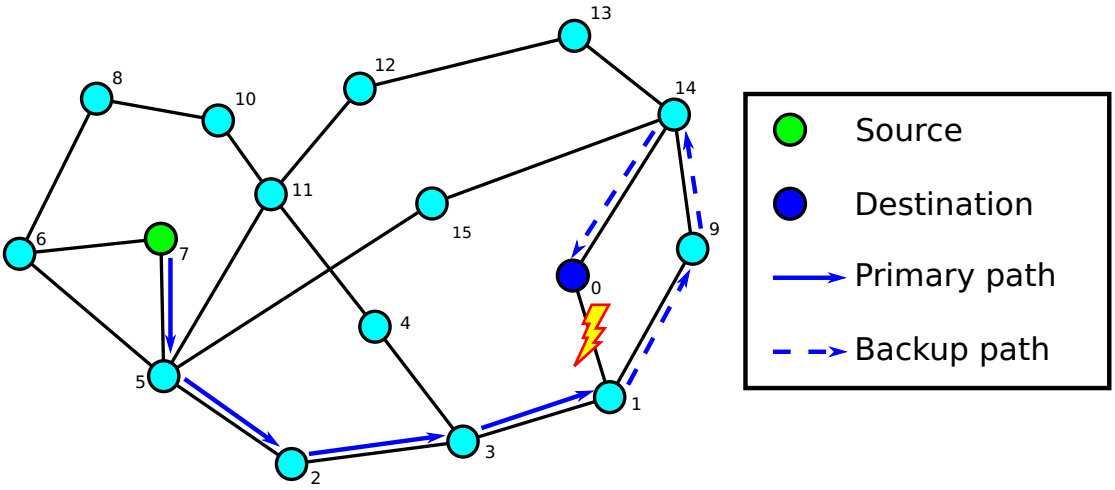


Figure 4.6: DSTs leverage the shortest cycle that contains the PLR and the destination. It requires that the forwarding color changes which may increase the backup path length compared to dMRTs.

4.3.4 Application and Implementation in IP and SDN-based Networks

In this section, we discuss the applicability of (d)MRTs and DSTs in IP and SDN-based networks. We first address the computational complexity of the proposed solutions. As presented in Chapter 2, every IP router has to calculate its primary and backup NH to each destination. In an SDN-based network, the computation of forwarding entries for the network devices is performed in one or more control elements. We assume a single centralized controller to estimate the algorithm complexity for an SDN approach. Note that a distributed controller approach can reduce the overall computation of a single controller, e.g., by dividing the network in disjoint parts.

We also address how (d)MRTs and DSTs can be implemented in OpenFlow. We first present how MRTs are implemented in IP networks. We do not propose an implementation of DSTs for IP networks. Then, we present the implementation of MRTs and DSTs in OpenFlow. The latter requires a more complicated implementation to avoid forwarding loops when multiple failures occur.

Computational Complexity for MRTs We first outline the complexity for simple MRTs to provide comparability to dMRTs. An IP node has to compute the default hops using a SPF computation. The ADAG computation is assumed here to be in linear time [126] but may be higher for advanced approaches. Backup path calculation requires two modified (Figure 4.2) SPF computations as discussed in Section 4.3.1.1. The backup path selection (Section 4.3.1.2) is done in linear time leveraging information gathered by the ADAG construction and blue and red hop generation [100, 126]. A SDN controller computes the primary NH for all nodes in the network: n SPF computations. The controller needs to compute only one ADAG as all paths follow the single ADAG structure. The controller has to compute for each node the backup paths; this results in $2n$ modified SPF computations.

Computational Complexity for dMRTs For dMRTs the computation of backup paths in IP nodes has to change significantly. The node has to compute n ADAGs, which changes the linear computation in a polynomial one. The red and blue hops must be computed for each destination and its corresponding ADAG resulting in $2n$ SPF computations. Backup path selection can still be performed linear. For SDN, the computational overhead compared to simple MRTs does not change too much. The only difference is that the controller must also compute n ADAGs. Since computation of the controller was already polynomial, only a small overhead is expected for dMRTs.

Computational Complexity for DSTs The computation of the blue and red next-hops is identical to dMRTs. DSTs do not require a shortest path for the failure-free case and, thus, do not compute it. This saves a SPF computation for each destination but the major complexity is caused by the computation of the destination-based ADAGs.

Implementation of MRTs in IP and MPLS Networks The existing forwarding techniques are not changed bydMRTs since only the path layout is changed compared to simple MRTs. We briefly outline how MRTs are implemented in current networks and leave the details to the IETF draft [123]. Signaling of backup paths is implemented using tunneling: IP-in-IP tunnels and MPLS tunnels are supported currently. Each node

announces its address or label of the primary path as well as the blue and red backup paths in the network. In failure cases, the PLR encapsulates the traffic using the address of the appropriate color of the destination.

Implementation of (d)MRTs in OpenFlow We propose to implement (d)MRTs in OpenFlow analogously to the implementation existing for IP networks. OpenFlow 1.1 or higher is required to support the implementation of (d)MRTs since it introduces both MPLS support and group tables to implement backup mechanisms.

We explain the structure of the OpenFlow pipeline and how the controller configures the switches for (d)MRTs. The OpenFlow pipeline consists of one or more flow tables. The controller installs one entry for each destination that points to a group table entry of type fast-failover. An entry consists of a list of action buckets which are executed consecutively until the first bucket succeeds. Thus, the controller installs the forwarding next-hop in the first bucket and the encapsulation to the appropriate colored destination in the second bucket.

MPLS is an optional feature for OpenFlow 1.1, and may not be available on all OpenFlow switches. If MPLS is available, the controller simply implements (d)MRTs by pushing the appropriate MPLS label on the packet as action in the group table entry. We recommend MPLS tunneling since it is more lightweight than IP-in-IP tunneling. OpenFlow can also be configured for IP-in-IP tunneling but lacks actions for encapsulation with an additional IP header. We suggest two workarounds for this deficiency. The first one is to create additional interfaces on the switch that perform the appropriate encapsulation. One interface is required for each color and destination pair that the switch may address. The OF-CONFIG protocol [59] can be utilized by the controller to configure the tunnel interfaces on the switches. The second workaround is to extend the OpenFlow protocol by actions for IP encapsulation and decapsulation. This approach was implemented in [128]. Then, the controller can set up the IP tunnels analogously to the MPLS approach.

The controller also installs the blue and red next-hops in the forwarding tables. Such an entry does not point to a group table but simply contains the actions to forward the packet or to decapsulate the packet. This prevents additional redirects in case of

multiple failures which may result in forwarding loops. This is the behavior suggested in the MRT drafts.

Implementation of DSTs in OpenFlow The implementation for DSTs leverages the same features as for dMRTs. The main difference to dMRTs is that the controller does not install default shortest paths that are used in the failure-free case. The controller installs only forwarding entries for the red and blue backup paths. The red and blue hops leverage group table entries with fast-failover. Blue paths are switched to the red path and vice versa. To avoid forwarding loops, each redirection towards a blue path or red path must be encoded in the packet header, e.g., in the differentiated services code point (DSCP) field. Each switch requires an additional forwarding entry that drops packets that are redirected twice. This terminates forwarding loops when multiple failures occur and is similar to the loop detection mechanism proposed in Section 4.2.

4.4 Load-Dependent Flow Splitting (LDFS)

In this section we propose LDFS OpenFlow networks. LDFS provides both resilience and Traffic Engineering (TE) for OpenFlow-based SDN. It requires the availability of a primary and one or more backup paths. The traffic is normally sent over the primary path. When network failures occur and the primary path fails, traffic is redirected on the backup path. If the primary path is available but the load of a flow exceeds a specified threshold, it is load balanced between the primary and the secondary paths.

4.4.1 Requirements for LDFS

LDFS provides both resilience and TE for OpenFlow-based SDNs. Therefore, we require that OpenFlow switches support group tables and the fast-failover mechanism. The SDN controller computes a primary path for each traffic flow. The flow may be either an fine-grained flow such a TCP flow or a traffic aggregate, e.g., an IP prefix. Moreover, the controller computes one or more secondary paths that should be disjoint to the primary path. The switch must be able to measure the liveness of a the primary and secondary paths. This is can be done using the port status for the next-hop or using a BFD measuring the liveness of a whole end-to-end path.

The second requirement for LDFS is load-balancing capability of the OpenFlow switch. With OpenFlow load-balancing is performed using group tables and the group type *select*. A *select* group table entry selects one bucket for each packet. For example, if a group should load balance through port 1 and port 4, two action buckets have to be defined; the buckets contain the actions “forward to port 1” and “forward to port 4”, respectively. The OpenFlow standard specifies that the algorithm which decides the action bucket to select is not part of the OpenFlow standard. Potential algorithms can be based on hash values over match fields or a simple round robin approach. Note that, we require that the selection algorithm only select *live* action buckets to implement LDFS.

The third and final requirement for LDFS is the capability of OpenFlow switches to monitor the traffic rates of flows and groups. As discussed in Section 2.5, various counters for the main components of a switch exist. These counters have to be updated when the packets enter the OpenFlow pipeline of the switch.

Flow table and group table entries contain counters that measure the entry's duration, received packets, and received bytes. Note that many of the statistics are optional. Thus, we require OpenFlow switches that support the necessary counters to implement LDFS. However, statistic generation using counters is difficult in practice due to hardware restrictions. This problem is described for OpenFlow switches in [129]. The authors provided a solution that is able to generate statistics more efficiently.

OpenFlow's monitoring capabilities were notably enhanced in OpenFlow 1.3. Meter tables enable OpenFlow to implement simple QoS operations. In combination with per-port queues, complex QoS frameworks such as DiffServ can be implemented. Meter table entries contain a list of meter bands. Each meter band has a specified rate on which it applies. The meter with the highest configured rate that is lower than the current measured rate is chosen. Thus, meters also have the potential to implement LDFS.

4.4.2 LDFS Policies

The LDFS algorithm operates on either on fine-granular flows or on macro flows such as IP prefixes. Macro flows can be easily defined using wildcard-based OpenFlow matching. Individual flows can be handled as an aggregate by using the same group for each flow. The *group* instruction provides selection of group table entries. The rate of an aggregate is should be measured using the group counters or by a meter that is attached to the corresponding flow table entries.

LDFS requires a new group type. This new type should cause the group to behave similarly to the *select* type. The group needs to store a threshold rate value T that distinguishes normal traffic from overload traffic. As long as the threshold rate is not exceeded, the first action bucket is executed. Otherwise, the switch performs load-balancing on the remaining action buckets.

In addition, each bucket requires an associated *liveness* condition as if needed for

fast-failover groups. Note that only *live* action buckets can be selected. The controller is responsible to compute the paths in such a way that the secondary action buckets form disjoint paths to the primary path. At least one secondary bucket is required.

We propose three different load-balancing variants which behave differently on overloads. In the following, we define the rate of an aggregate g as $r(g)$ and the excess traffic rate as $r_e(g) = r(g) - T$.

Balance All Traffic (BAT) When the threshold is exceeded, the whole aggregate is load-balanced over the secondary paths. Thus, the same rate is sent over all paths. Note that the BAT method has the same forwarding behavior as the multipath routing on traffic overloads. Otherwise, the primary path is selected.

Balance Excess Traffic (BET) The BET method is similar to the BAT method. However, we only split the excess traffic over the secondary paths. This is achieved by configure the load-balancing in such a way that there is a load of $T + \frac{r_e(g)}{2}$ on the primary and $\frac{r_e(g)}{2}$ on the secondary path.

Redirect Excess Traffic (RET) The RET method sends all excess traffic on the secondary paths. A load balancing algorithm that sends a rate of T on the primary and a rate of $\frac{r_e(g)}{n}$ on n secondary paths is a potential implementation of RET.

4.5 Coverage Analysis of LD-LFAs

In this section, we present our evaluation methodology for the coverage analysis of LD-LFAs. Then, we show the results for carrier-grade networks. Finally, we discuss the protection effectiveness of LFAs and LD-LFAs in datacenter topologies.

4.5.1 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.

4.5.1.1 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 (SLF) $\mathcal{S}^{1,0}$ and all single node failures (SNF) 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 [130].

4.5.1.2 Coverage

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,

	$\lvert\mathcal{T}\rvert$	$\lvert\mathcal{V}\rvert$	$avg(\lvert\mathcal{V}\rvert)$	$\lvert\mathcal{E}\rvert$	$avg(\lvert\mathcal{E}\rvert)$	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 4.1: 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.

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.

4.5.2 Coverage 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.

4.5.2.1 Networks under Study

We evaluate the fast reroute mechanisms for various networks from the Topology Zoo [127]. We classify these topologies into three categories: star topologies $\mathcal{T}_S$, ring topologies $\mathcal{T}_R$, and mesh topologies $\mathcal{T}_M$. Table 4.1 provides an overview of the selected networks. We omit $\mathcal{T}_S$ in our analysis due to the lack of alternate neighbors.

4.5.2.2 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.

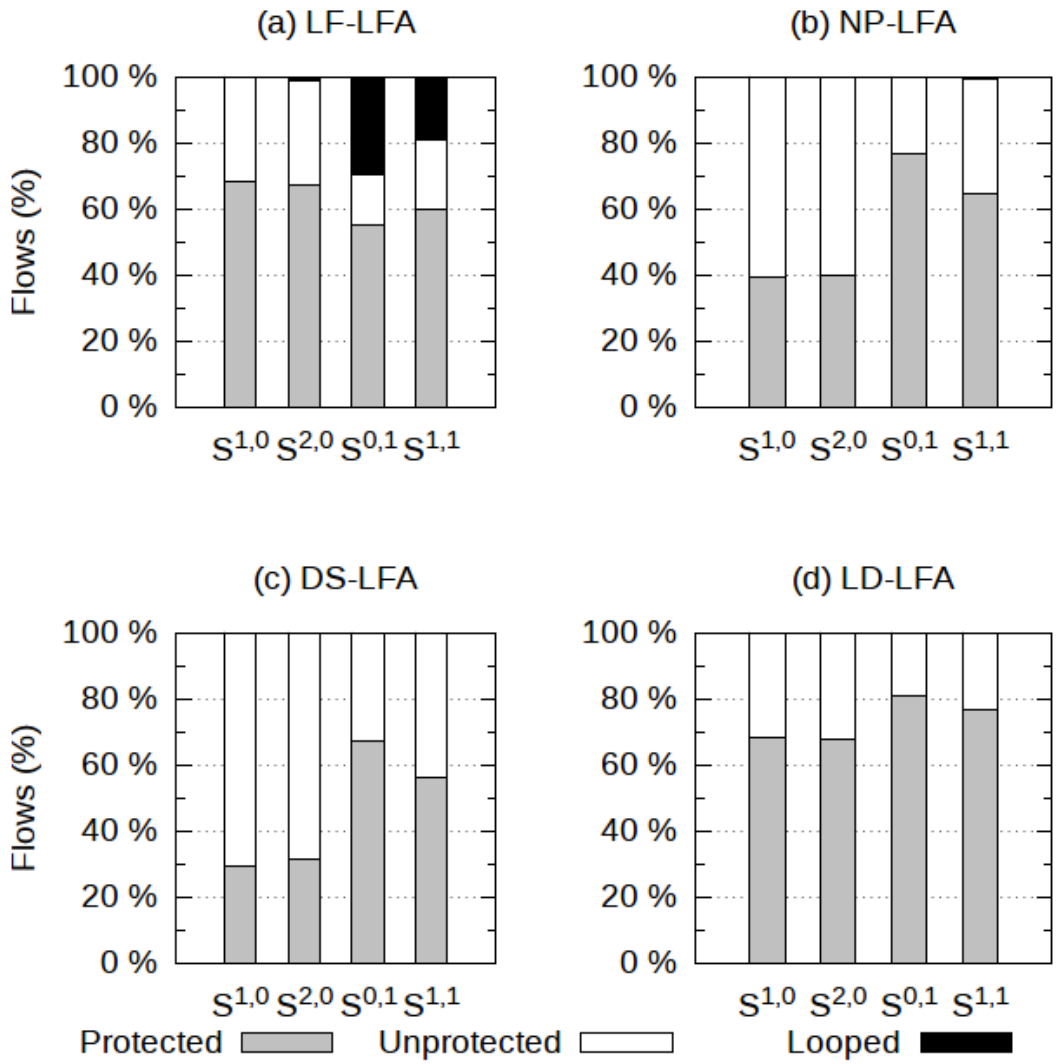


Figure 4.7: Percentage of protected flows in mesh topologies.

Figures 4.7 (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 4.8 (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 4.7 (b) and Figure 4.8 (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.

4.5.2.3 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

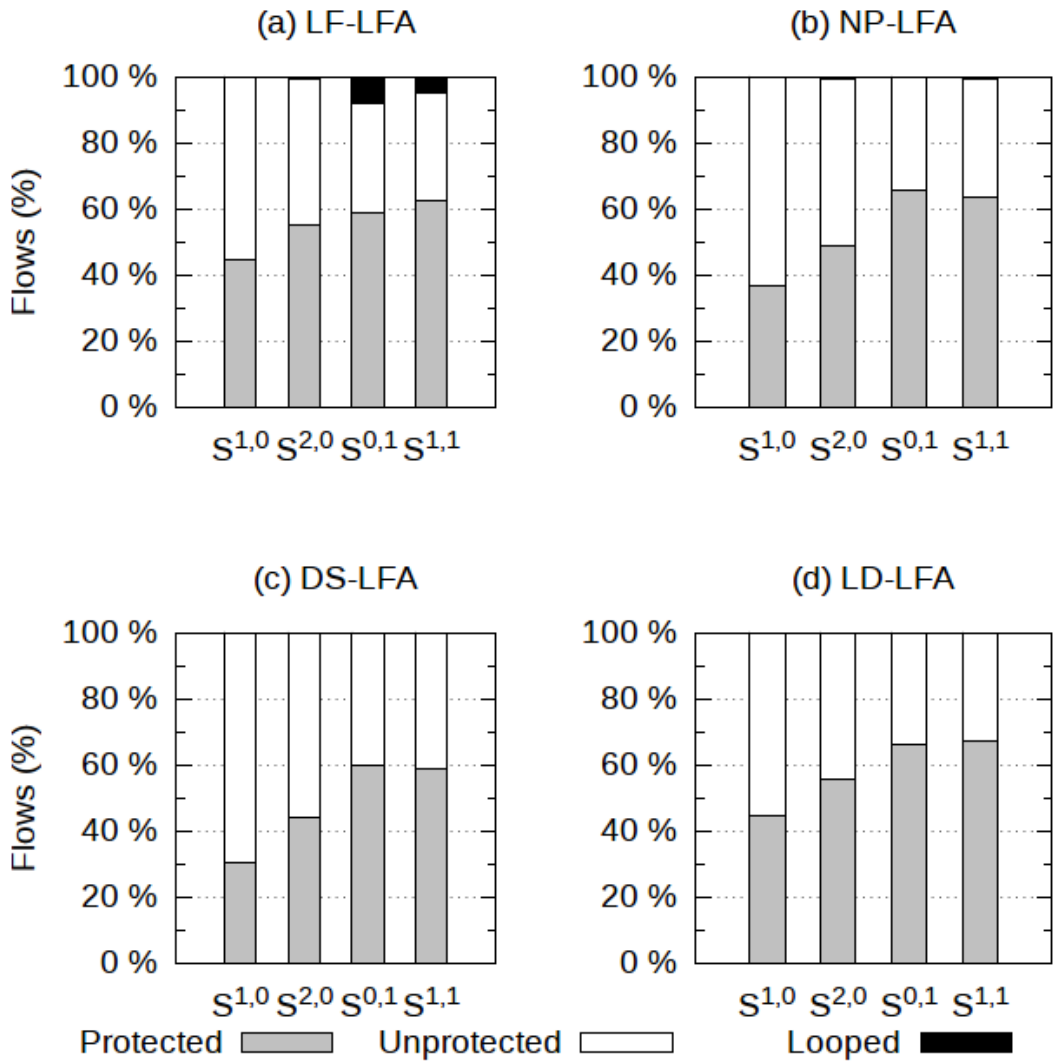


Figure 4.8: Percentage of protected flows in ring topologies.

LD-LFAs are shown in Figure 4.7 (d) and Figure 4.8 (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 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.

4.5.2.4 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 4.9. 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.

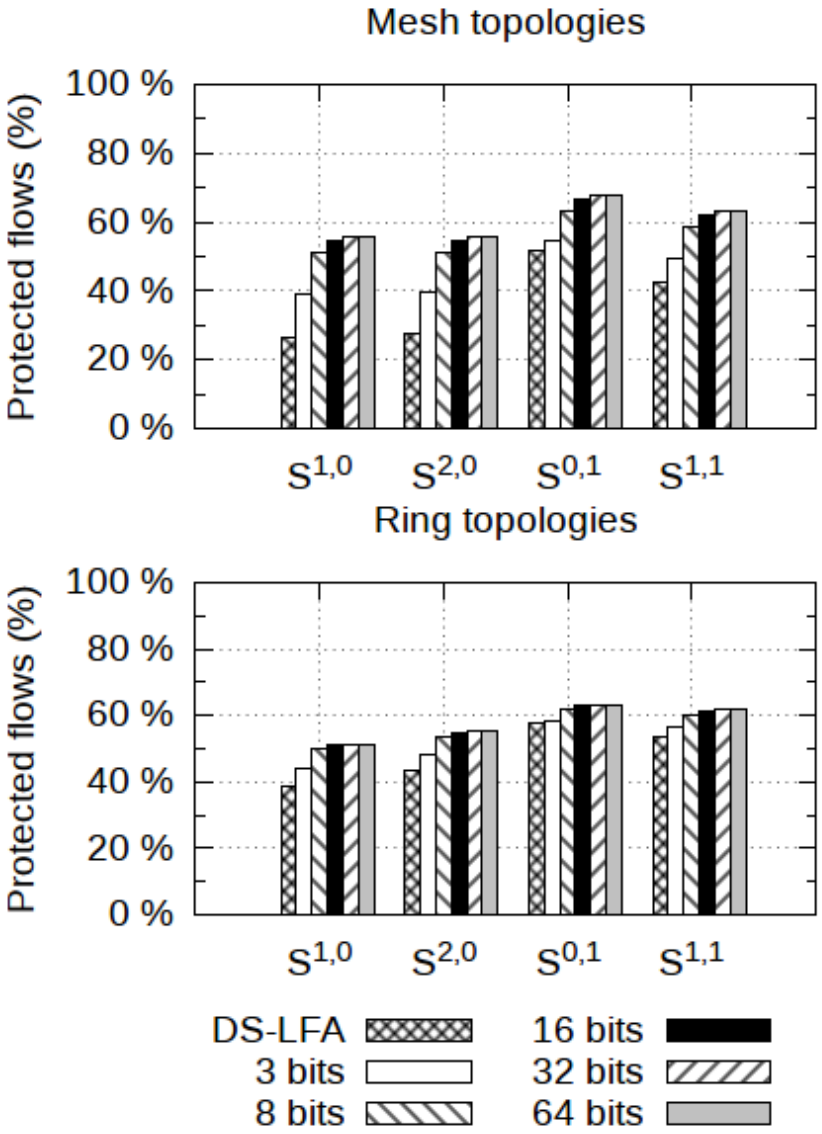


Figure 4.9: Percentage of protected flows in large mesh and ring networks for DS-LFAs and LD-LFAs with varying bit lengths. Bit lengths of 8 or 16 are sufficient and their protection is comparable to unlimited ID lengths in large networks.

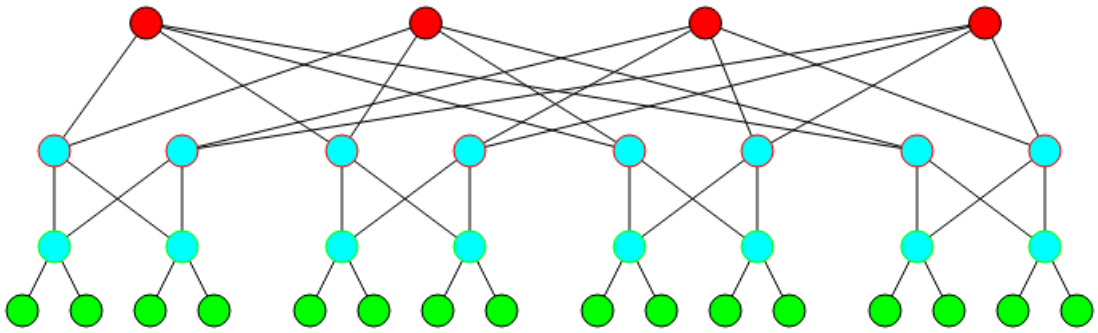


Figure 4.10: The fat-tree topology with $k = 4$ consists of four pods and supports up to 16 servers (green nodes).

4.5.3 Results for Datacenter Networks

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

4.5.3.1 Fat-Tree Networks

Fat-tree topologies [131] 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 4.10 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 switches but in such a way

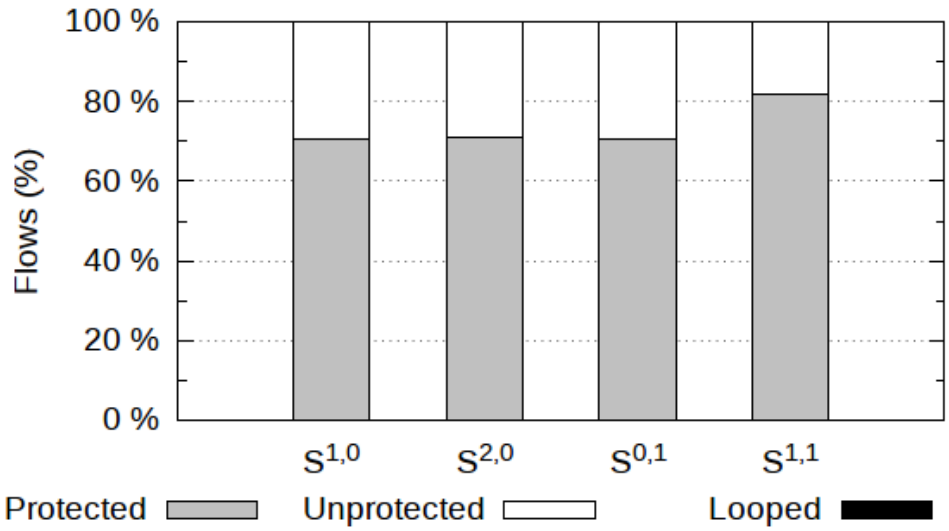


Figure 4.11: Percentage of protected flows in fat-tree with $k = 6$. All LFA variants (LF, NP, DS, LD) provide identical protection.

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 $\frac{k^3}{4}$ 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 datacenters 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 4.11. We observed two key findings. Firstly, we obtain the same coverage results for all LFA variants. Secondly, LFAs do not generate loops in

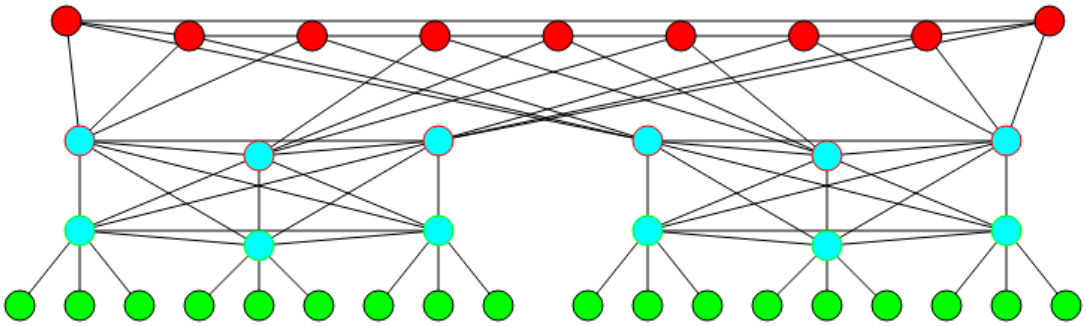


Figure 4.12: Augmented fat-tree topology with $k = 8$ and two pods.

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.

4.5.3.2 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 4.12 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 4.5.3.1. We generate only two pods instead of the possible six pods.

The results are shown in Figure 4.13. 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 protect 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 4.12. Consider that packets are sent from the first server in the first pod to the first server of the second pod in Figure 4.12. 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 only

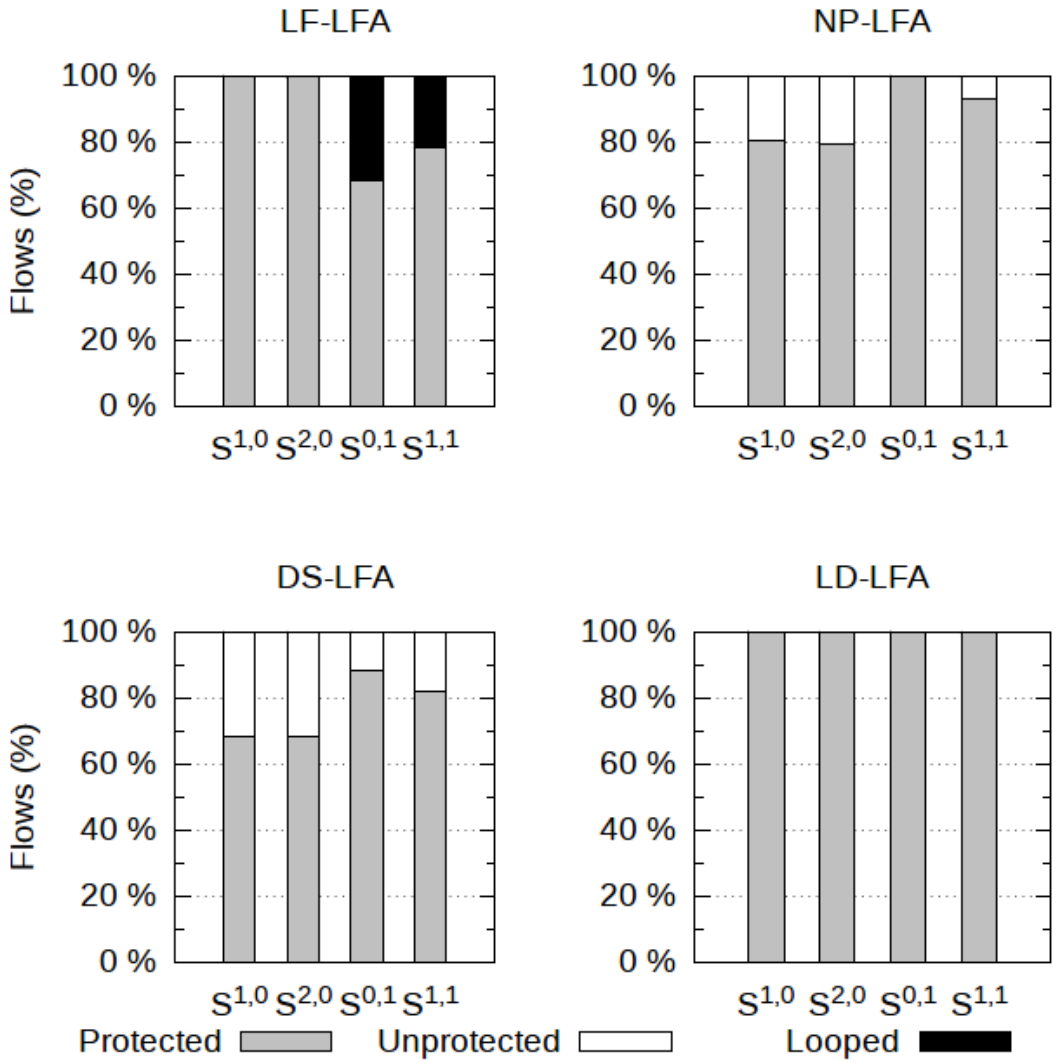


Figure 4.13: Percentage of protected flows for the augmented fat-tree topology with $k = 8$. In a very few multiple failure cases ($\mathcal{S}^{2,0}$, $\mathcal{S}^{1,1}$), LD-LFAs cannot protect less than 0.3% of the traffic, which is not visible in the figure.

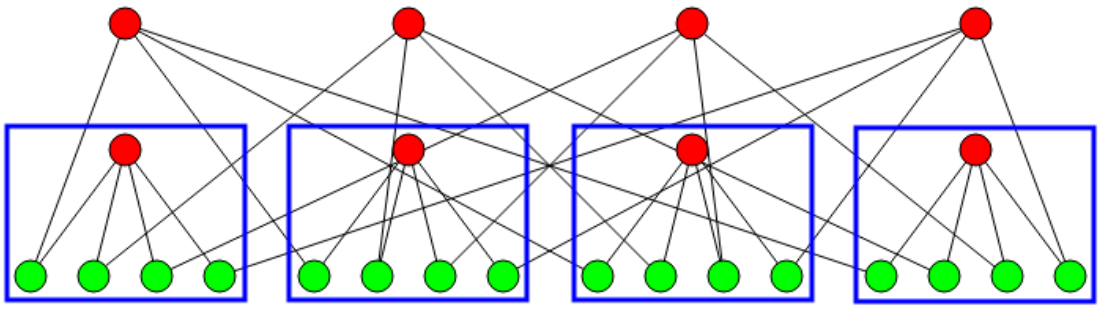


Figure 4.14: BCube_1 network with $n = 4$ and $k = 1$ consists of 4 BCube_0 cells. The cells are interconnected using an additional layer of switches.

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 datacenter are often placed as close together as possible, i.e., in the same server or in the same pod [132], 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.

4.5.3.3 BCube Networks

BCube is a specifically designed datacenter topology intended for shipping-container based, modular datacenters [133]. 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 4.14 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

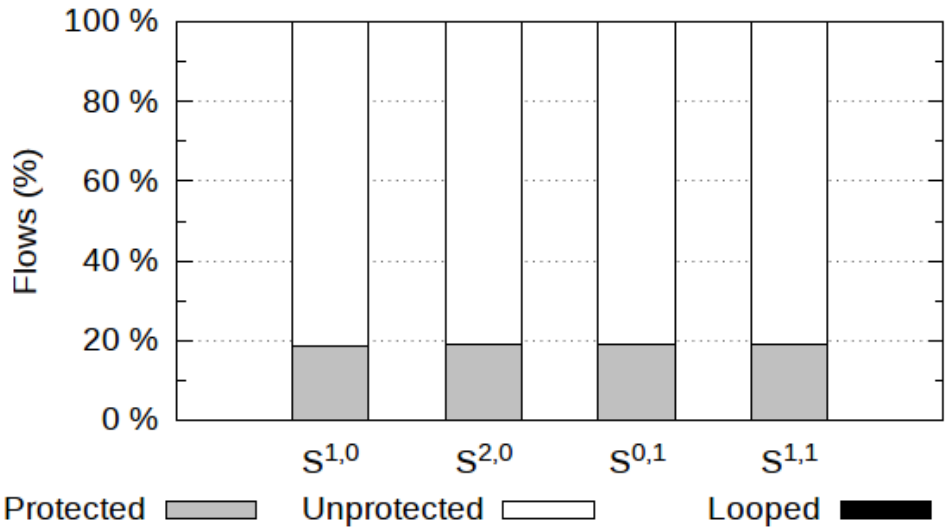


Figure 4.15: Percentage of protected flows in BCube with $n = 4$ and $k = 1$. AllLFA variants (LF, NP, DS, LD) provide identical protection.

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 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 [133].

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) AllLFA variants provide exactly the same protection. (2) There are no forwarding loops for all considered scenarios including mul-

tiple 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 4.14. 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.

4.5.3.4 DCell Networks

DCell networks are proposed in [134] 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

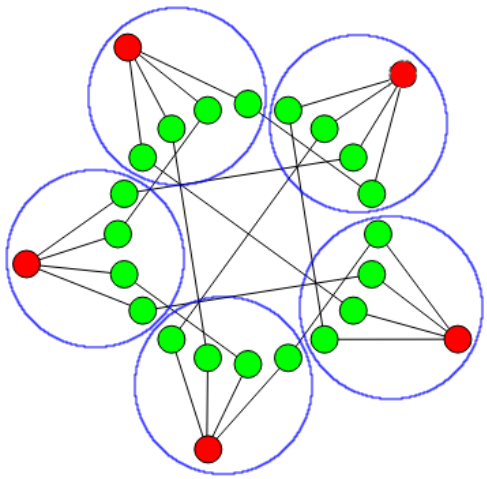


Figure 4.16: DCell_1 with $n = 4$ and $k = 1$ consists of 5 DCell_0 cells (blue). Each switch (red node) is part of one DCell_0. Traffic from one cell to another is forwarded using servers (green nodes).

$k = 1$ is shown in Figure 4.16. 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 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 [134].

We present the results for the DCell network with $n = 4$ and $k = 1$ in Figure 4.17.

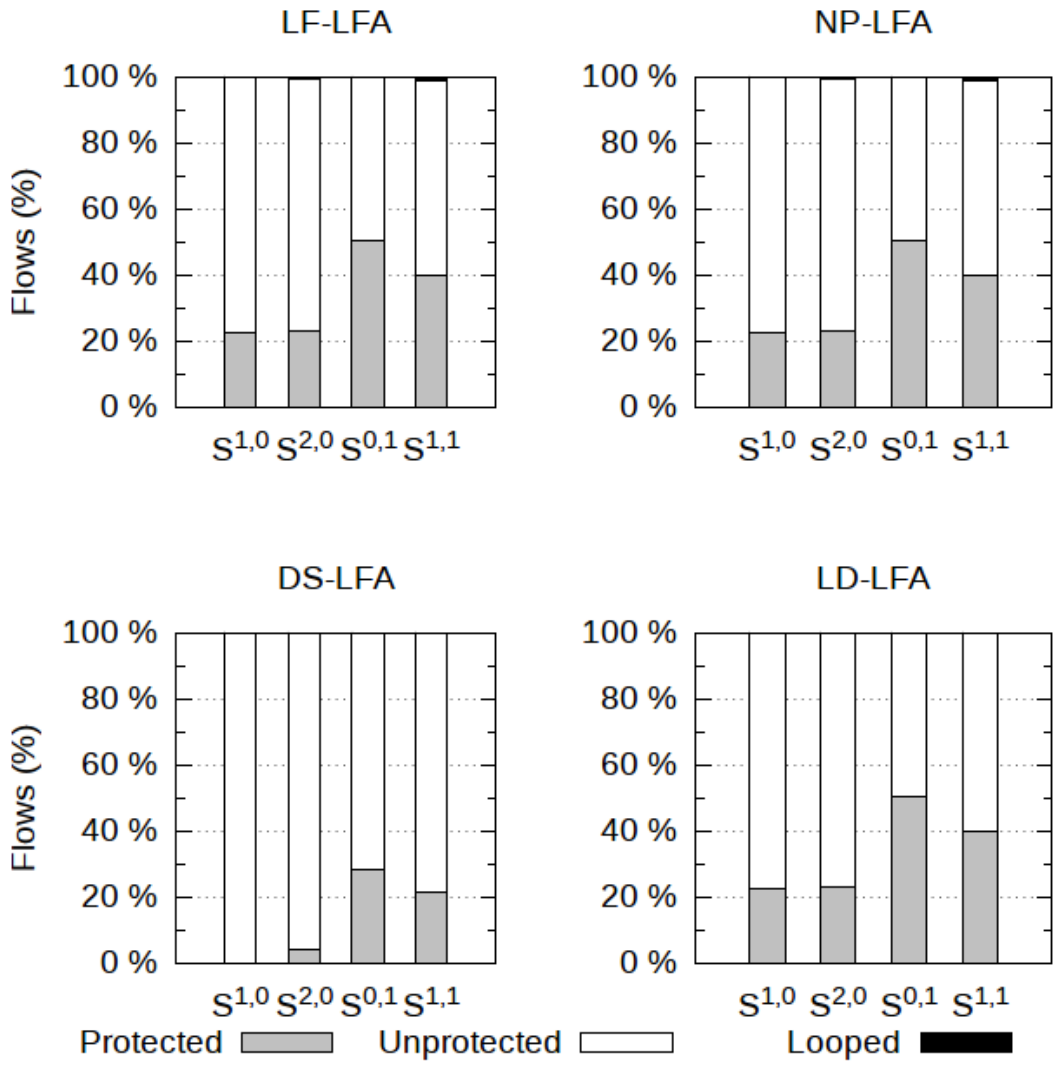


Figure 4.17: Percentage of protected flows for the DCell topology with $n = 4$ and $k = 1$.

The results are similar for larger networks and additional results do not provide further insights. We found that LF-LFAs and NP-LFAs protect equally and there are only loops for less than 1% of the flows for multiple failures. 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.

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.

4.5.4 Summary

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 pro-

tected 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 datacenter 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.

4.6 Comparison of Full Protection Proposals

In this section, we discuss the considered failure scenarios, traffic assumptions, and considered metrics. We compare dMRTs and DSTs with MPLS FRR and MRTs. We provide two MRTs variants that are configured in such a way that path lengths and network capacities are minimized.

4.6.1 Methodology

We evaluate the flow layout for the different methods in the failure-free case (FF) and in single link failure scenarios (SLF). In FF, (d)MRTs and MPLS are routed both using the shortest path principle. DSTs leverage the shorter of the two paths.

We use unit link costs as input for the routing mechanisms due to the lack of information specified in our networks under study described below. In failure cases, packets are rerouted using the pre-established backup paths of the selected routing mechanism.

We consider networks from the Topology Zoo [127] that compromises various commercial and research networks. Nodes that are connected by only a single vertex cannot be protected against a failure of the connecting link. Thus, we modify the topologies by removing such nodes. Furthermore, topologies with a vertex count greater than 200 are omitted. The data set still contains 150 different topologies of various sizes which have 4 to 166 vertices, with an average of 27.6. The number of edges varies from 6 to 212 with an average of approximately 38. We think that the data set is large enough to show the generality of the results presented below.

The Topology Zoo does not provide traffic matrices. We generate for each topology a homogeneous traffic matrix that consists of traffic demands of equal load for each source-destination pair.

4.6.2 Metrics

We consider three metrics for our evaluation: path lengths, required relative capacities, and number of forwarding entries needed. We provide a cumulative distribution function

(CDF) for each metric over all networks under study that allows to identify the trends of the different methods with regard to a specific metric on the large network set.

Path length prolongation We measure the path length of a flow in hops in the failure-free case and for each single link failure. The maximum and average path lengths in the failure-free case are $p^{\text{FF}}_{\text{max}}$ and $p^{\text{FF}}_{\text{avg}}$. We count the maximum and average number of hops in SLF of the *rerouted* flows $p^{\text{SLF}}_{\text{max}}$ and $p^{\text{SLF}}_{\text{avg}}$. Finally, we report the average and maximum path prolongations (p_{avg} and p_{max}) for SLF caused by the rerouting process. The prolongations are computed by $p_{\text{avg}} = p^{\text{SLF}}_{\text{avg}}/p^{\text{FF}}_{\text{avg}}$ and $p_{\text{max}} = p^{\text{SLF}}_{\text{max}}/p^{\text{FF}}_{\text{max}}$, respectively. Using normalized path prolongations in this evaluation is especially helpful because the topologies under study differ in size and, thus, allow easy generalization of the results in the large analyzed data set.

Required relative capacities We determine the utilization of each link in the failure-free case and for every single link failure. We compute the sum and the maximum of all links as network capacity and maximum link capacity, respectively. We report the relative network capacity C_n and the relative maximum link capacity C_l relative to the failure-free case, i.e., $C_n = C^{\text{SLF}}_n/C^{\text{FF}}_n$ and $C_l = C^{\text{SLF}}_l/C^{\text{FF}}_l$.

Number of additional forwarding entries To provide connectivity among all nodes in a network with n nodes, $n-1$ forwarding entries are required per node. MPLS and MRTs install additional forwarding entries to implement the backup paths. MRTs require two additional entries per destination. For MPLS facility backup, the number of additional entries varies for each node. We provide both the average and maximum percentage of additional entries for all nodes.

4.6.3 MRT Variants

While dMRTs leverage a single ADAG structure for each destination, MRTs are based on a single ADAG using a single root node. We already showed in [27] that the MRTs performance of path lengths and link utilization heavily depends on the selection of the

root node. Therefore, we compare dMRTs to different MRT root node configurations. We show only MRTs with root nodes in the results that provide the best performance results for a specific metric. We consider MRTs optimized on the maximum path prolongation p_{max} (MRT_p) and relative network capacity C_n (MRT_C). If two root nodes provide the same value for the metric, we consider the root node with the lower secondary metric: average path prolongation and maximum link capacities, respectively. We omit other MRT configurations because we could not find any additional insight.

4.6.4 Required Relative Capacities

We present the required relative capacities in this section. We analyze (d)MRTs for each topology and provide C_n and C_l. The required relative network capacity C_n is shown in Figure 4.18 when single link failures are considered. The figure shows hardly any difference with regard to C_n between MRTs and dMRTs. capacities increased by a factor between 2 and 2.4 for most topologies compared to the failure-free case. MRT_C results in marginally less required capacity than dMRT and MRT_p. DSTs and MPLS_n requires slightly more network capacities than (d)MRTs and have almost the same requirements. MPLS_l performs worst. The difference between MRT_C and MPLS_l is between about 0.15 and 0.34 for 90% of all topologies. This can be explained by the fact that MPLS_l redirects closely to the failed link which may cause local hotspots more easily than for MPLS_n and (d)MRTs.

Figure 4.19 shows the CDF for maximum relative link capacity. We observe the same trends as for the relative network capacity. Rerouting using (d)MRTs increases the required maximum link capacity by a factor between 1 and 2 in all topologies. DSTs and dMRTs result in the least maximum link capacity but the difference compared to MRTs is only marginal. The difference between dMRTs and MPLS_l is about 0.36 and 0.5 for 90% of all topologies.

We conclude that dMRTs, DSTs, and MRTs do not differentiate much concerning required capacities caused by rerouting. (d)MRTs require approximately 2 – 2.4 more capacity to protect against failures compared to the failure-free case. (d)MRTs require less capacities than MPLS. The difference is greater for MPLS_l than MPLS_n.

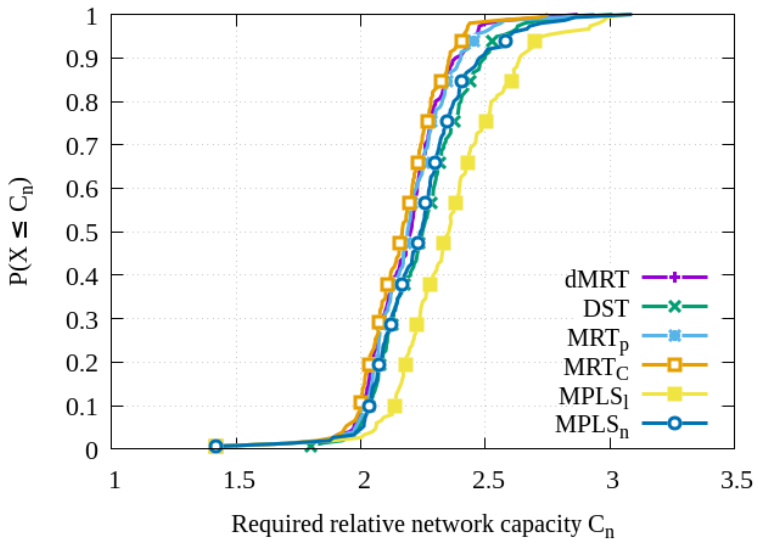


Figure 4.18: CDF of required relative network capacity C_n.

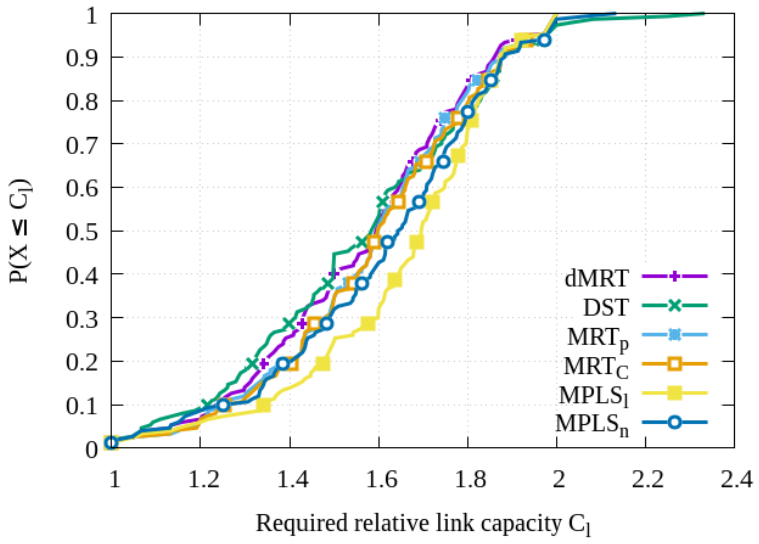


Figure 4.19: CDF of required relative maximum link capacity C_l.

4.6.5 Path Length Prolongation

In this section, we present the path length prolongation caused by the rerouting mechanisms. Figure 4.20 shows the average path length prolongation for all considered topologies as a CDF. MPLS FRR and dMRTs cause the shortest and similarly long backup paths with an average path prolongation of approximately 1.75 – 2.7 for most topologies. MRTs result in longer path prolongations. DSTs and MRT_p are in the range from 1.8 to 3 and notably longer than MPLS or dMRTs. DSTs are visibly shorter than MRT_p. MRT_C causes the longest average path lengths with a p_{avg} between 2 up to 3.5 for about 90% of the topologies. The maximum path prolongation p_{max} is illustrated in Figure 4.21. Notable differences for p_{max} are observed for all methods. MPLS_l has for 87% of the networks a $p_{\text{max}} \leq 2$ but MPLS_n only for 73%. Both can result in a factor up to 2.75. dMRTs and MRT_p have 60% and 46% of the networks with $p_{\text{max}} \leq 2$ and in the worst case $p_{\text{max}} \approx 3.2$. DSTs are notably worse with a $p_{\text{max}} \leq 2$ for 62% of the considered topologies. The maximum factor is around 3.5. Finally, for MRT_C, only 26% of the networks result in $p_{\text{max}} \leq 2$ and a maximum factor of 3.7.

dMRTs clearly lead to shorter backup paths than MRTs even if the root node is optimized on backup path length. DSTs provide shorter backup path lengths than MRT_p on average but causes longer maximum backup paths. We observed that dMRTs have on average similarly long backup paths as MPLS FRR. The longest backup paths for dMRTs are clearly longer than for MPLS FRR, especially for MPLS_l. Thus, we expect dMRTs only have longer backup paths for few failure scenarios and be competitive to MPLS with regard to path lengths in general.

Figure 4.22 shows the performance of DSTs in the failure-free case with regard to path lengths. We provide the relative path prolongation of the DST primary path compared to the shortest path leveraged by (d)MRTs in the failure-free case. We found no path prolongation for the longest paths for 45% of the analyzed topologies. However, the longest paths may be increased by a factor up to 1.7. In 15% of the topologies, we found no path prolongation at all. The average path length prolongation in the remaining 85% increases up to a factor of 1.3. Therefore, we clearly identified a notable trade-off for DSTs compared to dMRTs because most traffic is not affected by failures in general.

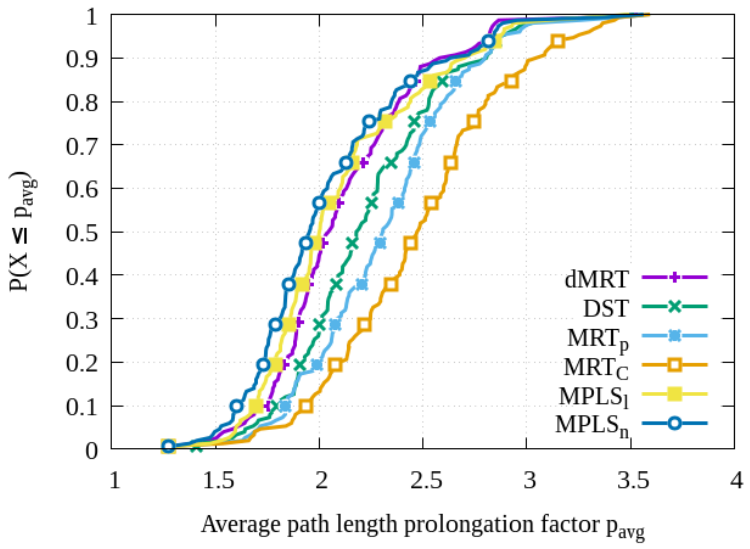


Figure 4.20: CDF of average path length prolongation p_{avg}.

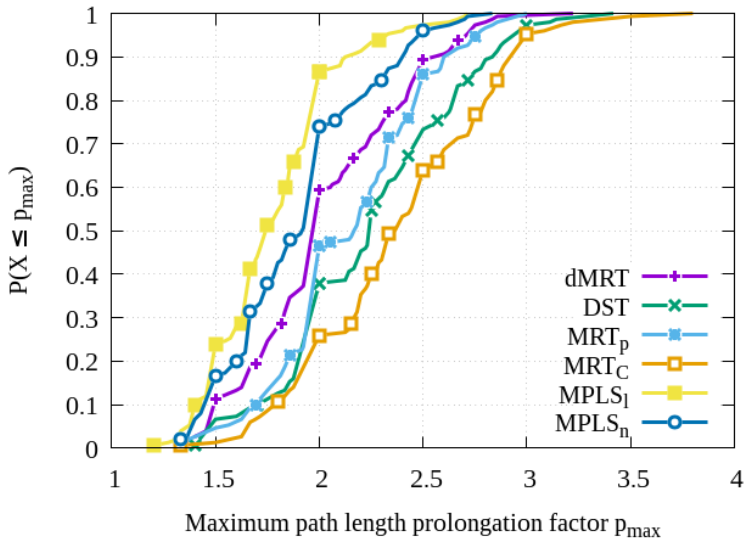


Figure 4.21: CDF of maximum path length prolongation p_{max}.

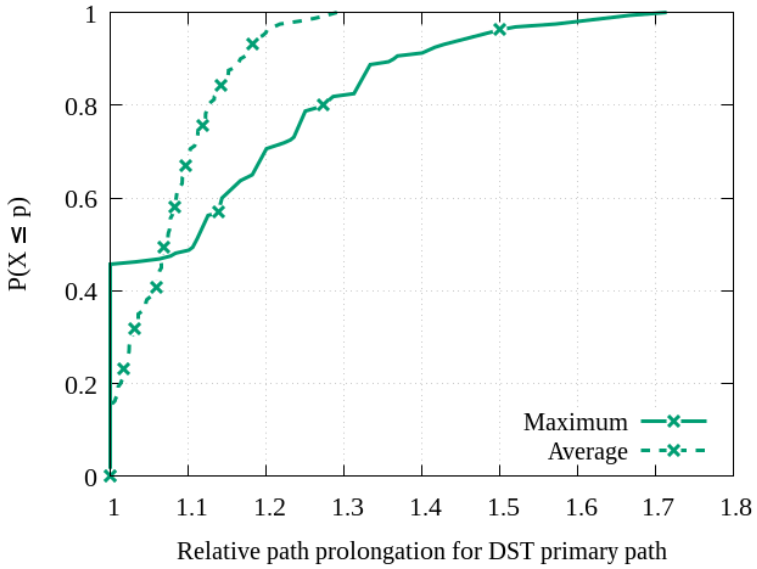


Figure 4.22: CDF of primary path prolongation for DSTs.

4.6.6 Number of Additional Forwarding Entries

Figure 4.23 illustrates the required state for MPLS compared to MRTs as a CDF. The number of additional forwarding entries for MRT and dMRT are fixed by 200% and DSTs by 100%. For MPLS, we differentiate between link and node protection and illustrate the average and the maximum number of forwarding entries. The average number of entries for link-protecting MPLS is below the number for MRTs. For approximately 25% of all networks, the maximum number is larger than 200%. More than 95% networks result in less than 300% and in few cases the number of maximum forwarding entries reaches about 500%. Node-protecting MPLS requires significantly more forwarding entries. For 22% of the networks, the average number of forwarding entries is between 200% and 400%. For few networks the number increases up to 900%. We found that only 8% of the networks require less than 200% maximum forwarding entries for MPLS. For about 80% of the networks the number is between 200% and 600%. The remaining 10% of the networks require over 600% up to 1800% additional forwarding entries.

We conclude that (d)MRTs clearly outperform MPLS with regard to state require-

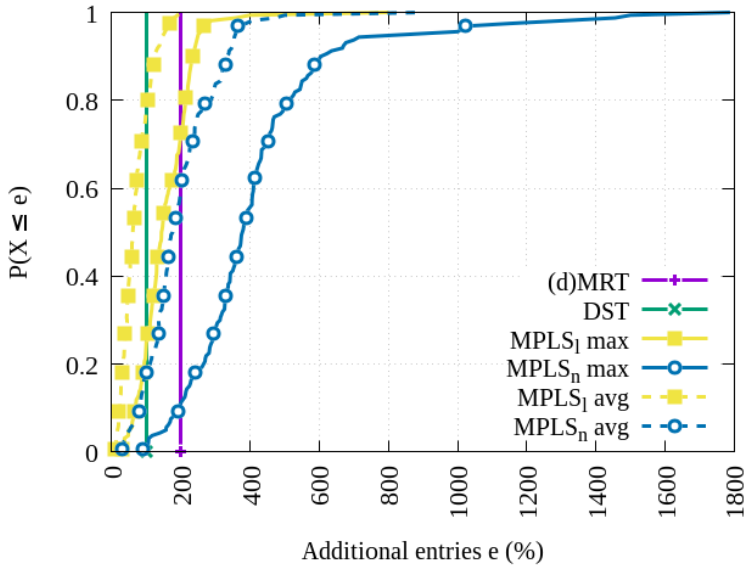


Figure 4.23: CDF of additional forwarding entries.

ments on the basis of two main facts. MRTs do protect against both link and node failures and require significantly less state to achieve node protection. Although link-protecting MPLS requires for about 60% of the networks less state, MRTs are very close to them but achieve higher protection.

4.6.7 Summary

We evaluated dMRTs and DSTs for various networks. The results show that both lead to very similar bandwidth requirements when single link failures are considered. We analyzed the path prolongation that is caused when packets are rerouted. We found that dMRTs lead to the shorter backup paths than MRTs and DSTs. On average dMRTs perform similar to MPLS FRR. When MRTs are optimized on path lengths (MRT$_p$), we observe notably longer backup paths on average and slightly longer for the worst cases compared to dMRTs. Both MRT$_p$ and dMRTs provide significantly shorter backup paths compared to MRTs optimized on bandwidth requirements. DSTs are shorter on average to MRT$_p$ but causes longer backup paths in the worst case. We identified notable performance degradation with regard to path lengths for DSTs for traffic that is not

affected by failures due to the lack of shortest paths in the failure-free case.

Considering additional forwarding entries, (d)MRTs produce clearly better results than MPLS. Although $MPLS_l$ requires slightly fewer entries it only protects against single link failures. (d)MRTs cover every single component failure and the comparable $MPLS_n$ mechanism relies on significantly more forwarding entries for the same degree of protection. DSTs clearly require the least forwarding entries.

4.7 Performance Analysis of LDFS

In this section, we present the performance evaluation of our proposed method. First, we show the networks under study and explain our metrics. We present our results and discuss the impact of load-dependent flow splitting compared to single-shortest path and multipath routing.

4.7.1 Methodology

In the following, we describe and explain the traffic models that we consider for our performance evaluation. We present the network scenarios that model the combination of network failures and traffic overloads and discuss the computation of required link capacities depending on a routing mechanism. Finally, we discuss the paths that are used for multipath routing and for LDFS.

4.7.1.1 Notation

We denote a network graph as $G = (V, E)$. V is the set of nodes and E is the set of edges or links. We refer to the set of aggregates as $\mathcal{G}$. Each aggregate $g \in \mathcal{G}$ is characterized by its source and destination ($g = (s, d)$) and has a rate $r(g)$.

4.7.1.2 Network Scenarios

We consider network failures and overloads. We model network failures with link failures and traffic overloads with the models described in Section 4.7.1.3. We refer to the combination of link failures and traffic overloads as network scenario. The scenario specifies all failed links and the traffic matrix containing the rates of all aggregates.

We define various sets of network scenarios $S^{i,j}$. This set contains all scenarios that consist up to i simultaneous link failures in combination with up to j simultaneous overloads. For example, $S^{0,0}$ contains one scenario: the failure-free case with no overloads, $S^{1,0}$ contains all single link failures without overloads, etc.

We compute the required capacity for a routing mechanism and a set of network scenarios S by measuring the traffic rate for the routing mechanism with regard to the

link failures and the traffic matrix. We compute the required link capacity $c(l, s)$ for each link in each scenario $s \in S$. The required capacity for a link is $c(l) = \max_{s \in S} c(l, s)$.

4.7.1.3 Traffic Models

We evaluate the LDFS performance on various networks. The networks provide information about the topology but do not provide any information on the traffic that traverses the network. Therefore, we generate traffic matrices for each network using the gravity model [135] for the performance analysis. The gravity model is based on a population function $\pi(n)$ for nodes $n \in V$. The overall traffic C_{tot} is distributed in the network in such a way that more populated nodes send and receive more traffic than less populated ones. The aggregate rates $r(g)$ are determined with

$$r(g = (v, w)) = \begin{cases} \frac{\pi(v) \cdot \pi(w) \cdot C_{tot}}{\sum_{x,y \in V, x \neq y} (\pi(x) \cdot \pi(y))} & \text{if } s \neq d \\ 0 & \text{if } s = d \end{cases}$$

Hot Spot Model The hot spot model [136] shifts traffic locally to hot spots. The overall traffic C_{tot} in the network is constant and, thus, the hot spot model is considered conservative. When an overload occurs, flows attached to a hot spot have increased load while other flows experience reduced load. The model applies an overload factor f_o to the population function of a hotspot:

$$\pi^{v}_{hotspot}(w) = \begin{cases} \pi(w) & \text{if } v \neq w \\ f_o \cdot \pi(w) & \text{if } v = w \end{cases}$$

Node Overload Model The node overload model increases the overall traffic in the network. This model is inspired by traffic overloads caused by inter-domain routing [125, 136]. All flows that are attached to an overloaded node carry a multiple of the original load. The rate of such a flow is multiplied with the overload factor f_o.

Aggregate Overload Model The aggregate overload model shares the same basic idea as the node overload model. In contrast, an aggregate instead of a node can be overloaded having a rate multiplied with the overload factor f_o.

4.7.1.4 Path Layout for Multipath Routing and LDFS

We compute two paths for each aggregate. The algorithm computes for each source and destination a pair of paths that are maximally disjoint, i.e., share the least number of links possible.

primary pat and a secondary path for LDFS We compare single-shortest path routing, Multipath Routing (MPR), and the LDFS variants (BAT, BET, RET). MPR and LDFS both use the 2-shortest paths to highlight the impact of the dynamic load balancing approach.

For each source and destination, two shortest paths are computed that are maximally disjoint, . The paths are computed with a k-disjoint shortest path algorithm [137]. The shorter path is the primary and the longer the secondary path. Note that this primary path can be longer than the shortest path from the source to the destination.

4.7.2 Networks under Study

We provide results for the Agis, Abilene, and GÉANT networks that are publicly available at the Topology Zooo [127]. The Agis and Abilene networks are long-haul backbone networks in the USA. The former one consists of 25 nodes and 30 bidirectional links and the latter one consists of 11 nodes and 14 links. The GÉANT network is a European research network with 40 nodes and 61 links. We selected the human population to generate the traffic matrices for the population function [138, 139].

4.7.3 Metrics

We denote required link capacities for a set of network scenarios S and a routing method m as $c(S, m, l)$. The computation of these capacities is described in Section 4.7.1.2.

The required network capacity is the sum of all link capacities. The maximum link capacity can be an interesting metric because higher rate interfaces are generally more expensive than lower rate interfaces.

We use values relative to single-shortest path routing (SPR) to simplify the comparison of multipath routing and LDFS. Thus, we define the relative required network capacity as

$$C_{net}(S, m) = \frac{\sum_{l \in E} c(S, m, l)}{\sum_{l \in E} c(S, \mathrm{SPR}, l)}$$

and the relative maximum link capacity as

$$C_{link}^{max}(S, m) = \frac{\max_{l \in E} c(S, m, l)}{\max_{l \in E} c(S, \mathrm{SPR}, l)}$$

4.7.4 Impact on Required Network Capacity

We present the results for the required network capacity for the GÉANT network, the hot spot model, and an overload factor of $f_o = 3$. The results of the other networks, models, and overload factors are very similar and additional figures do not provide further insight. However, differences are explicitly stated in the text.

Figure 4.24 shows the relative required network capacity in the GÉANT network for various network scenarios. The failure-free case without overloads $(S^{0,0})$ leads to approximately the same required network capacity for LDFS and SPR because LDFS only uses the primary paths. MPR requires additional 20% capacity which is caused by the usage of longer paths. In single link failure scenarios $(S^{1,0})$, both MPR and LDFS results in 8% less capacity compared to SPR.

In overload scenarios without failures ($S^{0,1}$, $S^{0,2}$), BET leads to less required network capacity than SPR and significantly less than MPR, BAT, and RET. In the Agis network, we reduced the required network capacity by approximately 13% for BET compared to SPR. RET performs worse than the other routing methods. When the overload factor is

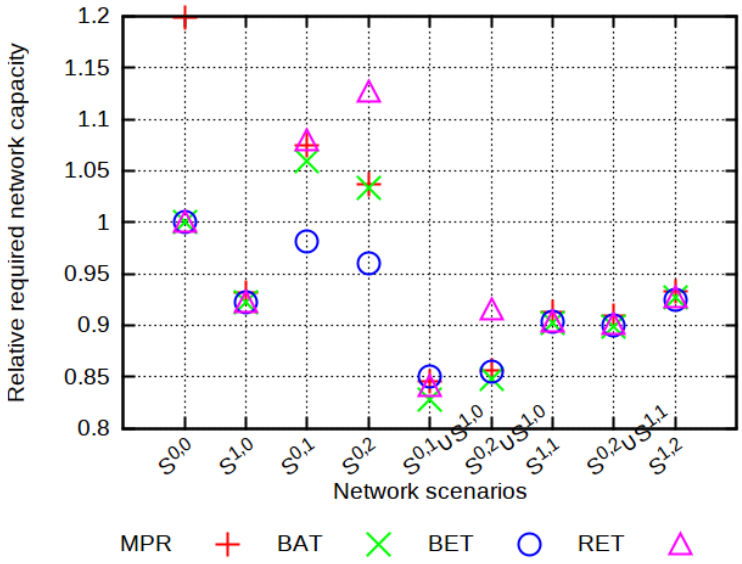


Figure 4.24: The relative required network capacity in the GÉANT network for various network scenarios. The traffic overload is modeled by hot spots with a factor of 3.

increased, the required network capacity for RET increases significantly. This is caused by sending a lot of traffic over the longer secondary path when the load significantly exceeds the threshold rate.

For non-simultaneous overloads and failures scenarios $(S^{0,1} \cup S^{1,0}, S^{0,1} \cup S^{1,0})$ we observe additional savings of 10% compared to SPR. Yet, MPR and LDFS do not differ so clearly in the GÉANT network. The BET method performs slightly better than the other mechanisms in the Agis and Abilene networks.

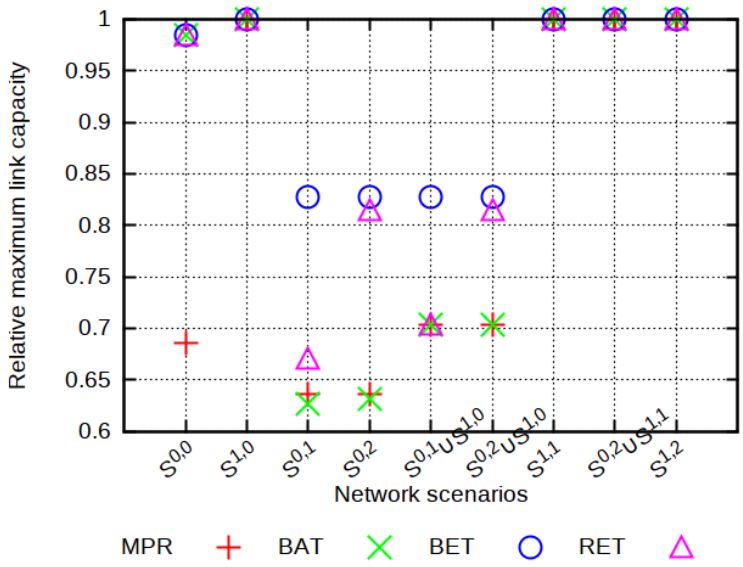


Figure 4.25: The relative maximum link capacity in the GÉANT network for various network scenarios. The traffic overload is modeled by hot spots with a factor of 3.

Almost no differences can be observed for the different routing methods in network scenarios with simultaneous link failures and traffic overloads $(S^{1,1}, S^{2,0} \cup S^{1,1}, S^{1,2})$. In these network scenarios, it is possible that one of the paths of an overloaded flow is affected by a link failure. Then, there is no path choice for both MPR and LDFS. Note that the RET method provides notably higher required network capacities for high overload factors than the other methods in these network scenarios.

4.7.5 Impact on Maximum Link Capacity

Figure 4.25 shows the relative maximum link capacity in the GÉANT network for various network scenarios. In $S^{0,0}$, LDFS methods are close to SPR and MPR leads to almost 30% less maximum link capacity. However, for the Agis network we observe significantly higher maximum link capacities for the LDFS methods. Surprisingly, MPR performs worse in the Abilene network which may be caused by the topology.

The maximum link capacities are equal for the routing methods in network scenarios

with failures $(S^{1,0}, S^{1,1}, S^{2,0} \cup S^{1,1}, S^{1,2})$. We again explain this fact by the lack of choice for MPR and LDFS in certain failure scenarios with simultaneous link failures and overloads. In the GÉANT network, the maximum link load is between 5 – 10% lower compared to SPR. In the Agis and Abilene network it is close to SPR.

The maximum link capacities differ significantly in network scenarios with non-simultaneous link failures and overloads. MPR and BAT often perform quite similar and requires up to 37% less capacity compared to SPR. MPR and BAT behave equal in overload scenarios which explains the similarity with regard to maximum link capacities. The BET method performs better than SPR but approximately 12 – 20% worse than MPR. In the Agis and Abilene networks, MPR and LDFS only perform better than SPR in $S^{0,1}$ and $S^{0,2}$. In addition, the maximum link capacity for BET is similar to MPR in the Abilene network.

4.7.6 Summary

In this section, we evaluated three different load balancing policies (BAT, BET, and RET). We compared the three LDFS variants with traffic-agnostic single-shortest-path and 2-shortest-paths routing (SPR, MPR) with regard to required network capacity and maximum link capacity. We found that the BET variant is in most scenarios similar or better to the other LDFS variants and MPR. We found that LDFS requires the least capacity when networks are expected to accommodate traffic for non-simultaneous link failures and traffic overloads, but MPR requires least link capacities. The capacity saving by LDFS diminish when networks were provisioned for simultaneous failures and overloads.

4.8 Discussion and Summary

We proposed various protection mechanisms for SDN-based networks. They all provide different strengths and weaknesses. We briefly review their key performance indicators and give recommendations for their usage.

Mechanism	OpenFlow Version	Complexity	Coverage	State	Path lengths	Capacity
LDFS	OF 1.3 + extensions	Configuration + management	Full	✖		✓
LD-LFA	OF 1.1	Loop detection	Partial	✓✓✓		
MRT	OF 1.1	Simple	Full	✓	✖	
dMRT	OF 1.1	Simple	Full	✓	✓✓	
DST	OF 1.1	Loop prevention	Full	✓✓	✖✖	
MPLS	OF 1.1	Simple	Full	✖	✓✓	✖

Table 4.2: Overview of analyzed resilience mechanisms.

Table 4.2 summarizes the strengths and weaknesses of the analyzed resilience mechanisms. LDFS is the least promising approach for SDN resilience. It does provide capacity savings but introduces high complexity due to OpenFlow extensions and configuration of individual flows. Disjoint source-destination paths require additional state within the network. Therefore, we do not recommend LDFS in general. LD-LFAs require the least state of all considered approaches but cannot guarantee full protection. Only one additional forwarding entry is required in each switch. We do not measure path lengths or network capacities to compare them with full coverage mechanisms due to LD-LFAs partial coverage. This prevents fair comparison due to the different set of successfully rerouted flows. LD-LFAs should be used in augmented fat-tree networks since they can protect all single link failures. We also recommend them in carrier-grade networks with a mesh topology when forwarding state is more important than protection against all failures.

MPLS FRR, (d)MRTs, and DSTs provide full coverage against all single link and single node failures. DSTs require the least state with 100% additional forwarding entries. 200% are needed with (d)MRTs. MPLS state requirements depend on the topology but we observed significant state overhead for most networks compared to DSTs and (d)MRTs. With regard to backup path lengths, we observed different results. MPLS results in the shortest backup paths and dMRTs are slightly longer. DSTs can be shorter

or longer than simple MRTs. However, DSTs cause increased path lengths for traffic that is not affected by failures. Therefore, we recommend the use of DSTs when minimizing forwarding state has highest priority while guaranteeing full protection. The usage of dMRTs is recommended when 200% additional forwarding state is tolerable for the considered network due to significant shorter paths compared to DSTs. When path lengths are most important and the network tolerates the state overhead of MPLS, it is a suitable option to keep the paths short.

5 Scalable and Resilient P4-based Software-Defined Multicast

We present the Software-Defined Multicast (SDM) architecture that supports multicast traffic in such a way that only topology-specific forwarding entries must be hold inside SDN switches. This ensures that each switch only has to hold a fixed number of entries in the forwarding tables to forward an arbitrary number of multicast flows. We leverage the Bit Indexed Explicit Replication (BIER) and Traffic Engineering for BIER (BIER-TE) architectures that are currently standardized in the IETF. We address resilience aspects by proposing 1+1 protection for BIER and different resilience methods for BIER-TE.

In addition, we discuss the implementation of this architecture. First, we discuss the data plane requirements for BIER in SDN. We consider OpenFlow and the P4 language as southbound interface and present why P4 is a more suitable option. We present and explain the implementation of the SDM prototype.

The content of this chapter is mainly taken from [10, 11] and organized as follows. We first discuss the issues that occur with IP multicast and why BIER and BIER-TE are needed for SDM. We present the BIER architecture and propose a 1+1 protection solution in Section 5.2. Then, we explain the BIER-TE architecture and explain different FRR methods. Section 5.4 contains a study that compares BIER and BIER-TE options with regard to different performance metrics. The SDM architecture and its data plane requirements are presented in Section 5.2.1. Then, we describe the demonstration and the prototype implementation in Section 5.6. We summarize our findings of SDM in Section 5.7.

5.1 Scalability Issues of IP Multicast

IP multicast allows efficient data transmission from one sender to many receivers. In general, it is desirable that traffic is optimally forwarded in such a way that the least packet duplication and load in the network occurs. The research of the past decades improved many areas for multicast, e.g., traffic-engineered multicast deployment. However, network operators today still face several operational problems when applying current multicast protocols, e.g., Protocol Independent Multicast (PIM), to common use cases such as "Layer 3 VPNs", IPTV, and over-the-top services [140, 141]. The protocols generally rely on explicit tree building mechanisms. The trees have to be installed in the routers and increase their overall state. For some use cases, too much state is required in the routers so that they are unable to optimally forward the multicast traffic. E.g., operators often flood packets of some multicast flows to all potential egress nodes regardless of a subscription. This wastes bandwidth to save state in routers and operators have to find the right tradeoff. Multicast state in routers also causes operational issues if the network is reconfigured or subscribers are added or removed. The control plane requires many routers to participate in the process, compute new trees, and install new rules, which may result in convergence times of multiple minutes in large multicast deployments.

The IETF currently works on BIER to address the issues mentioned above. With BIER, multicast traffic can be forwarded without per-multicast-flow state by encoding the egress nodes of a multicast flow in a new BIER header [142]. Ingress nodes add this BIER header to multicast packet, transit nodes only forward them without multicast-flow-specific information, and egress nodes remove the BIER header. Thus, BIER supports a simple multicast overlay which is easy to operate and requires minimal state in routers. In addition, BIER-TE [13] is proposed in the same working group. It encodes the multicast tree in the BIER header and allows for traffic-engineered multicast trees with minimal state overhead.

5.2 Bit Indexed Explicit Replication (BIER)

5.2.1 Architecture and Forwarding Procedure

BIER [142] provides a multicast overlay without any states for multicast tree on core routers. A BIER domain consists of so-called bit forwarding routers (BFRs). Ingress BFRs add a BIER header to incoming multicast packets. The BIER header indicates all BFRs that should receive a copy of the packet, so-called egress BFRs. To that end, each BFR in the network is represented by one bit position in the BIER header which is set if the BFR is an egress BFR for the packet. BFRs forward BIER packets solely based on their BIER header and a Bit Indexed Forwarding Table (BIFT) whose entries depend on the routing underlay, which may be an IGP such as OSPF or IS-IS.

The forwarding procedure works as follows. A BFR essentially forwards BIER packets towards all egress BFRs indicated in the BIER header over the routing underlay. However, it sends at most one copy towards each next-hop and clears all egress BFRs in the BIER header that are not reachable by itself over the respective next-hop. The Bit Indexed Forwarding Table (BIFT) contains information that supports a BFR to perform these operations efficiently for fast packet processing. When a packet reaches an egress BFR, the BIER header is decapsulated and the packet is forwarded as usual. A salient feature of BIER is that only ingress BFRs need to know multicast groups including egress BFRs in order to add appropriate BIER headers to packets. All other BFRs in the BIER domain do not need to know these groups for multicast forwarding. This makes BIER scalable and requires only reconfiguration of ingress BFRs if multicast group membership of egress BFRs changes. Essentially, BIER removes the multicast state of conventional multicast routing protocols from core routers by encoding the multicast egress nodes into the packet headers.

BFRs may support bit strings (BIER headers) between 64 and 4096 bits, the support for a bit string length 256 is mandatory. Amongst other protocol information, it mainly contains the bits for potential egress BFRs. If the bitstring is too small to accommodate all BFRs, the set of potential egress BFRs can be broken down into several sets [142]. They basically receive traffic over egress-BFR-disjoint but possibly overlap-

ping multicast trees which increases the traffic load in the network. Thereby, very large topologies can be supported. BIER supports different subdomains for which different routing underlays may be used. Thereby, they can be leveraged to forward multicast packets differently, e.g., for traffic engineering purposes. If multiple subdomains are in use, the ingress BFR chooses the appropriate one for an incoming packet and indicates it in the BIER header.

5.2.2 Multicast only Fast Reroute for BIER

BIER does not provide a protection mechanism but rather relies on the restoration process of the routing underlay which may be slow and cause packet loss. We propose to combine Multicast only Fast Reroute (MoFRR) [143] with BIER to provide fast protection. MoFRR is based on 1+1 protection: the traffic is duplicated at the source and sent over redundant paths to the destination. The destination node continuously measures the quality of the streams from both paths, selects the one with highest quality for forwarding, and discards the other. In case of multicast, two redundant trees are required. The ingress node duplicates the traffic, sends it over both trees, and egress nodes forward packets from only one of them. If one tree fails, the egress node is likely to still receive the traffic from the other tree.

The BIER architecture may be upgraded as follows to implement MoFRR. At least two subdomains with different routing underlays are needed to allow for redundant multicast trees. The ingress BFR copies incoming BIER packets to two subdomains that provide redundant multicast trees. The egress BFRs read from the two streams and forward only a single copy.

A challenge is the provision of two routing underlays such that they yield redundant trees. We propose to leverage MRTs for that purpose which have been standardized by the IETF [123]. This idea has already been proposed for other multicast mechanisms in [144], but the draft has been abandoned. MRTs calculate dual routing topologies in a distributed way. In the absence of failures, a packet can be delivered of both of them to all destinations. The resulting paths in the two routing topologies do not necessarily form a pair of trees but the paths are node-redundant in the sense that the packet reaches

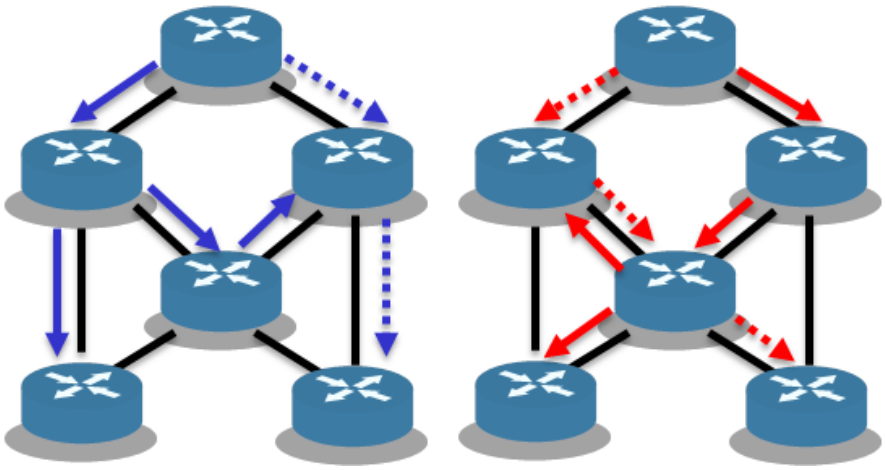


Figure 5.1: The blue and red MRT routing topologies forward traffic over node-redundant paths.

any destination over at least one routing topology in case of any single link failure. Figure 5.1 illustrates how a packet is forwarded over two redundant topologies (blue and red) from one source to all other nodes depending on its destination address. The straight lines represent a tree along which a packet may be delivered to all intermediate nodes and leaves. The dashed lines show how the packet reaches a single remaining node over an additional path whose intermediate nodes receive packets over the tree. This example helps to understand some evaluation results in Section 3.3. However, traffic is mostly carried over real trees and it was in fact difficult to find this small counterexample.

To protect unicast forwarding, MRTs are applied as follows. In the failure-free case, traffic is forwarded over shortest paths instead of one of the two routing topologies. In case of a failure, it is locally rerouted over a working routing topology. This may lead to very long backup paths [27].

We propose to use the MRT routing topologies for multicast forwarding although they do not form a tree. Nevertheless, we denote them as multicast trees when used in that context. For this purpose, multicast traffic is distributed along the two routing topologies to all required destinations. In case of a failure on one topology, the traffic is dropped instead of being switched to the other. In the context of BIER, the two MRT routing topologies must be maintained by routing underlays in the two different subdomains and

ingress BFRs copy incoming multicast packets to both of them. Packets are delivered only once to egress BFRs over a routing topology because BFRs modify the set of egress BFRs on forwarding in the packet header. This mechanism prevents that egress BFRs accidentally obtain separate copies over a solid and the dashed path within a routing topology and ensures that BIER packets are forwarded only when needed.

Standardized algorithms for calculation of the routing topologies are available in [100]. They can be computed using a few spanning tree operations in a very fast manner which is often desired in ISP networks. However, the path layout is not always optimal. Therefore, this MRT-based MoFRR solution is appealing for the deployment of BIER in large networks. The improvement of a node-redundant path layout for routing underlays with little state is an open research question.

5.3 Traffic Engineering for BIER (BIER-TE)

5.3.1 The BIER-TE Architecture

The BIER-TE architecture [13] is based on a segment routing [145] approach that is similar to SDN in the sense that the path layout for each flow can be explicitly configured by a controller. BIER-TE leverages the BIER header defined in the BIER architecture [142]. There are two main differences between BIER and BIER-TE. First, BIER-TE encodes both the links and the egress nodes of a multicast tree in the BIER header while BIER only encodes the latter. Second, unlike BIER, BIER-TE does not necessarily require an IGP control network or a routing underlay.

The BIER-TE architecture consists of a BIER controller and BFRs. The controller computes traffic-engineered multicast trees and instructs ingress BFRs to apply appropriate BIER headers to incoming multicast traffic. These BIER headers contains forwarding information and reflects the multicast structure. Furthermore, the controller installs forwarding rules in the forwarding tables of the BFRs which are called BIFT. Their contents is independent of existing multicast flows. A link between two BFRs in the BIER overlay is called an adjacency. An adjacency may be a physical link directly connected to a BFR neighbor or a tunnel provided by the routing underlay (remote ad-

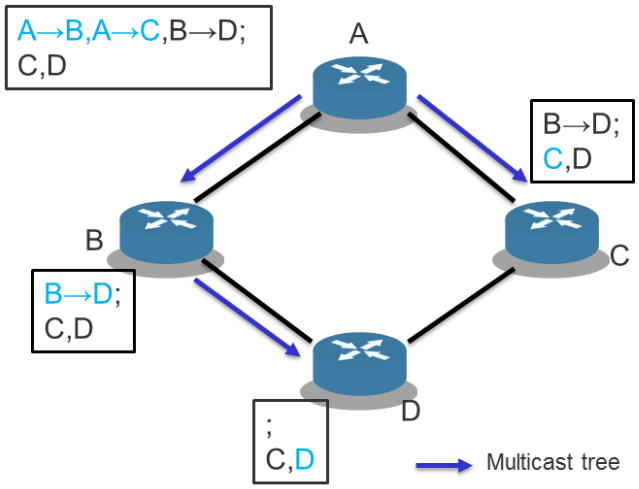


Figure 5.2: The BIER header contains adjacencies (first line) and `local_decap` bits (second line). The BOI of the nodes are highlighted in blue. The BIER header is modified before the packet is forwarded to the NH.

jacency). It is possible to have more than one adjacency between two BFRs when the destination BFR is reachable through different interfaces.

5.3.2 BIER-TE Forwarding

We illustrate the main forwarding procedure in Figure 5.2. Packets are sent from A to C and D. The initial header contains the links A→C, A→B, B→D and the egress nodes C and D. The egress node bits are called `local_decap` bits. The adjacencies and the `local_decap` bit of a BFR are called Bits of Interest (BoI) and are highlighted in blue in the headers. There are two BOI for A, A→C and A→B. A clears its BoI from the header and sends the packets to C and B. The packet arriving at C only has C set. The BIER header is decapsulated and the packet leaves the BIER overlay. Note that the BIER header is not empty at C and still contains bits with regard to the other subtree of the multicast tree. This is important for one of the FRR schemes presented in Section 5.3.3. The packets at subtree at B are processed in the same way as at A.

5.3.3 Fast Reroute for BIER-TE

The reachability of neighboring BFRs over a specific adjacency can be controlled by a BFD component so that a BFR can detect that a neighbor is no longer reachable and locally reroute affected traffic, i.e., the BFR acts as PLR.

If a BFR detects that a BIER packet needs to be forwarded over a failed adjacency, the BFR uses information in the BIER header to consult the BIER-TE Adjacency Forwarding Table (BTAFT). This yields a backup path including its encoding that the BFR applies to the packet header. Then, the BFR forwards the packet over another adjacency according to the modified header. The state information in the BTAFT of a BFR depends only on the number of its neighbors but not on traversing multicast flows.

To protect a link failure, the PLR forwards an affected packet to its next-hop over a backup path that bypasses the failed link. To protect a node failure, the PLR forwards an affected packets to all downstream next-next-hops (DS-NNH) over possibly several backup paths that bypass the failed node. A DS-NNH is a NNH of a PLR that receives the packet over the failed subtree. This concept is illustrated in Figure 5.3. A BFR can efficiently determine DS-NNHs using the BIER header and the information in the BTAFT. When link and node protection is combined, the PLR forwards an affected packet to all DS-NNHs and to the next-hop only if the next-hop is a destination of the packet. In general, the BFR cannot differentiate between link and node failures. Therefore, the controller configures the BTAFT of all BFRs such that link protection, node protection, or both are supported.

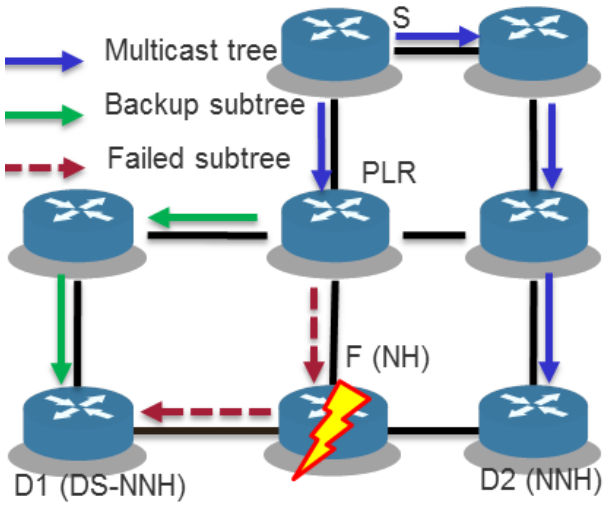


Figure 5.3: With node protection, the PLR reroutes traffic to all DS-NNHs.

5.3.4 Three Implementation Options for BIER-TE FRR

We propose three different implementation options for BIER-TE FRR. For more technical details, we refer to our specification in [14].

Point-to-Point Tunneling (PPT)

With PPT, the PLR reroutes BIER packets by tunneling them to appropriate next-hops and DS-NNHs over unicast tunnels. They are provided by a routing underlay and bypass the failed links and nodes, respectively. E.g., MPLS [146] may be used as a routing underlay. The provision of the tunnels possibly complicates the operation of the routing underlay and increases its state information. Moreover, each tunnel represents an additional adjacency and requires a separate bit in the BIER header. If a PLR reroutes a BIER packet over several unicast tunnels, some of them may share common links, which unnecessarily increases the traffic load on these links compared to the use of point-to-multipoint tunnels.

BIER-in-BIER Encapsulation (BBE)

With BBE, the PLR identifies the set of next-hop and DS-NNHs to which a BIER packet needs to be forwarded. The BTAFT helps to create a BIER-TE header towards these nodes avoiding the failed link or node, respectively. The BIER packet is encapsulated with that additional BIER-TE header and sent over the point-to-multipoint structure, which avoids unnecessary traffic increase on some links. The BIER packet is decapsulated at the egress nodes of this multipath.

Header Modification (HM)

With HM, the backup path is encoded in the existing BIER header through application of an AddBitmask and a ResetBitmask. Due to the forwarding mechanism of BIER-TE, this may cause duplicate packets for some multicast leaves. Therefore, some bits have to be cleared to avoid such duplicates by applying a ResetBitmask. We explain the occurrence of duplicates by the example shown in Figure 5.4. There are two multicast trees: (1) A sends to C and D, (2) B sends to C and D. If the link B→C fails, packets have to be rerouted by B over A and D towards C. Thus, B→A, A→D and D→C are added to the header. If we do so without clearing additional bit, the BIER header for multicast tree (1) still contains at node B the `local_decap`(D). As a consequence, the packet will be delivered to C and D. However, the packet is also directly delivered from the source A to D. Thus, `local_decap`(D) should be cleared in the header of the rerouted packet. This is different for multicast tree (2) because BIER-TE sends only a single packet over each interface and must, therefore, encode the backup path in the BIER header. If the `local_decap`(D) is set before transmission at B towards A, the packet will be delivered to D, otherwise it will not be delivered so that D loses the packet although its reachability is not affected by the failure.

Thus, after the backup path is added to the packet header through the AddBitmask, a ResetBitmask must be applied before sending the packet to avoid duplicates which, however, may cause packet loss for other destination. The ResetBitmask contains both the `local_decap` bits of the nodes on the backup path and their outgoing adjacencies. The latter are needed to ensure that duplicates are not propagated into other multicast

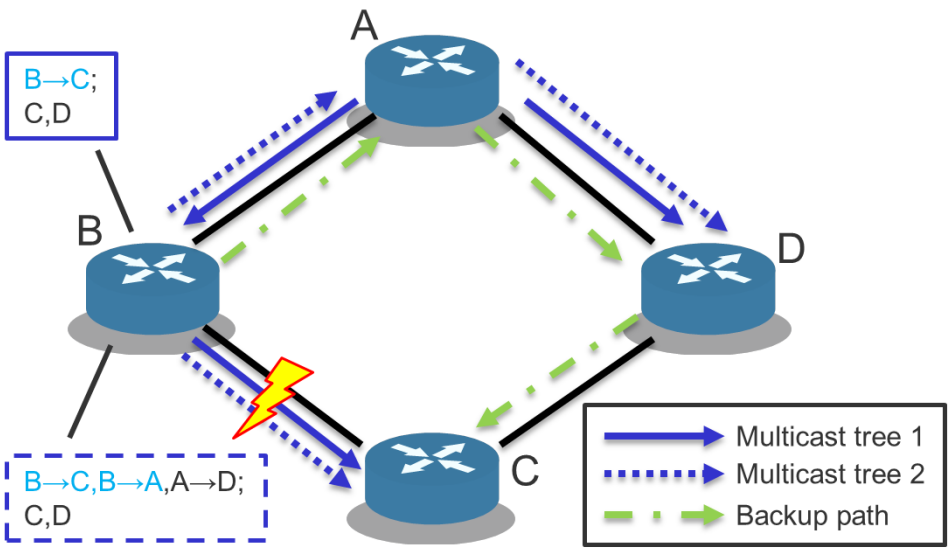


Figure 5.4: If A→D fails, clearing the `local_decap`(D) bit at B prevents a duplicate packet at D for multicast tree (1) but causes packet loss at D for multicast tree (2).

subtree if the backup path traverses them.

Although the HM method may lose some packets in case of failures, it is of interest because it avoids the overhead of an encapsulation header, it does not extend the BIER header by adding further adjacencies, and does not require support from a routing underlay.

Notation

Link and node protection {L,N} can be implemented with any of the presented protection methods for BIER-TE. We denote them by $\{\text{HM, PPT, BBE}\}_{\{\text{L,N}\}}$.

5.4 Performance Comparison of BIER and BIER-TE

In this section we evaluate key performance metrics of the discussed FRR mechanisms for BIER and BIER-TE. We first present the networks under study and explain our evaluation methodology. We quantify the effectiveness of the HM protection variants for BIER-TE. For the other protection methods we compare path lengths, consider resource requirements, and discuss state and header overheads.

5.4.1 Networks under Study

For our study, we leverage networks from the Topology Zoo [127] which contains research and commercial wide area and Internet service provider networks from mainly North America and Europe. We simplify the networks by consecutively removing all vertices that are attached to the network with only a single edge. We consider a network to have a ring structure if at least 60% of its nodes have node degree two, otherwise it has a mesh structure. This definition categorizes our 220 considered networks into 118 ring topologies T_R and 102 mesh topologies T_M.

The networks vary in size and their average numbers of nodes are 17.7 and 21.2 for T_R and T_M, respectively, the average numbers of unidirectional links are 22.9 and 33.4. Unlike BIER, BIER-TE encodes not only nodes but also links in the header. To that end, BIER-TE requires in ring topologies on average 40.7 bits in the header and at most 154 bits. In mesh topologies, BIER-TE requires on average 54.7 bits and at most also 143 bits.

5.4.2 Methodology

We analyze the forwarding behavior for all BIER variants in all 220 considered topologies. A topology is represented as a graph $G = (V, E)$. We investigate three different sets of failure scenario. First, the failure-free (FF) scenario. Second, the set of single link failures (SLF). It contains all bidirectional single link failures and, thus, consists of $|E|$ scenarios. And third, the set of single node failures (SNF).

The Topology Zoo does not provide traffic models or matrices. Therefore, we define a traffic model that is suitable for a systematic evaluation of multicast protection mechanisms. Every node in the network is sender of a multicast tree which has all other nodes as receivers, and any node sends the same unit rate. There is no other traffic in the network.

We apply shortest path routing to construct multicast trees for BIER-TE, shortest path around links for link protection, and shortest path around nodes for node protection methods. The path layout for MoFRR for BIER leverages MRT routing topologies that are computed according to the lowpoint algorithm in [100].

5.4.3 Efficiency of BIER-TE Protection with Header Modification

As outlined in the previous section, BIER-TE FRR with HM may lose traffic in order to avoid duplicate packets. In the following, we quantify the average fraction of traffic that may get lost for HM in all SLF and SNF, respectively. Also the perfect FRR schemes PPT and BBE lose traffic under some conditions:

- The network is not two-connected. If a critical link or node fails, the network is disected so that senders in one part of the network cannot reach the senders in the other part of the network.
- In case of node failures, traffic from or to a failed (ingress/egress) node is lost.
- If link protection is applied, traffic cannot be protected in case of a node failure.

Therefore, we consider the traffic loss occurred for PPT_N as lower bound on avoidable traffic loss.

Figure 5.5 shows the CDF of the percentage of lost traffic over all networks. Every data point on a curve corresponds to one network. Note that the curves consist of many more data points than markers which only improve their readability.

In case of SLF, all traffic can be protected by perfect FRR methods in more than 90% of the networks. The remaining networks are not two-connected with regard to links

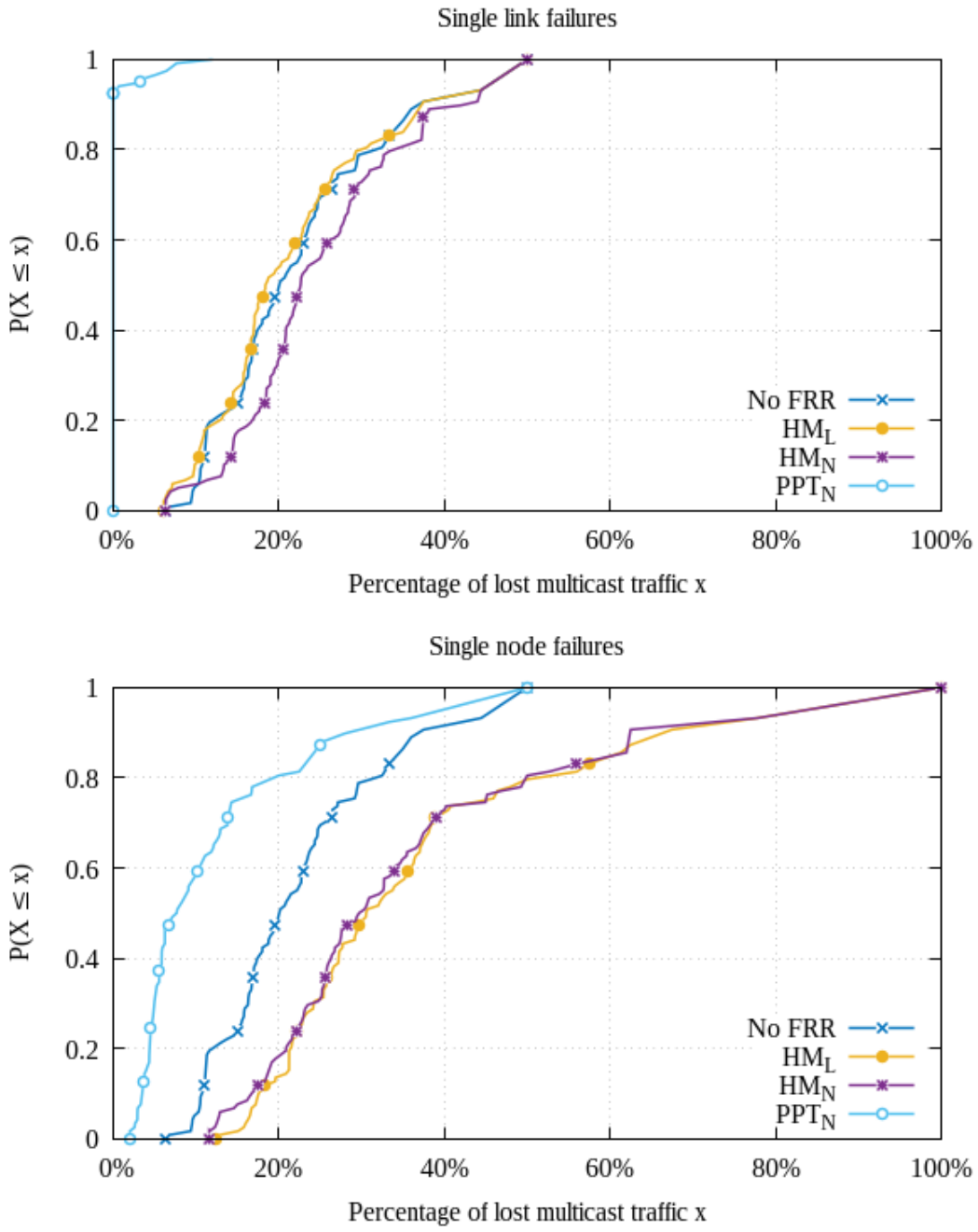


Figure 5.5: CDF of the percentage of lost multicast traffic in all networks for SLF and SNF, respectively.

so that the failure of a specific link disects the network. Another curve illustrates that between 7% and 50% of the traffic is lost without protection (No FRR). Surprisingly, HM methods cause about the same amount of lost traffic due to subtree pruning. When HM is configured to protect against node failures, it loses a bit more traffic than without protection while when it is configured to protect against link failures, it yields slightly less traffic loss than without protection.

In case of SNF, also the perfect FRR methods lose between 2% and 50% of the traffic in case of node failures, but at most 10% in 60% of the networks. This happens for the reasons given above and is unavoidable. Without protection, significantly more traffic is lost. The HM methods lead to even clearly more traffic loss than without protection. The reason for that node failures activate more backup paths than link failures so that more traffic is pruned from subtrees.

These results clearly demonstrate that HM is not efficient to protect against failures, yet it can be counterproductive in particular in case of node failures. Therefore, we exclude the HM methods from further discussions.

Nevertheless, HM can fully protect 20% – 40% of all multicast flows against SLF in 80% of the networks. This potential of HM could be further elaborated in future studies, which can be of interest to protect small multicast trees in some networks with only little technological complexity.

5.4.4 Path Lengths

The path layout depends on the applied routing mechanism and impacts path length in failure-free and failure cases. It is the same for PPT and BBE, but depends on link or node protection. From previous work [27] we know that MRTs may lead to excessive path length. Therefore, a comparison of the new resilience mechanisms with regard to path length is important.

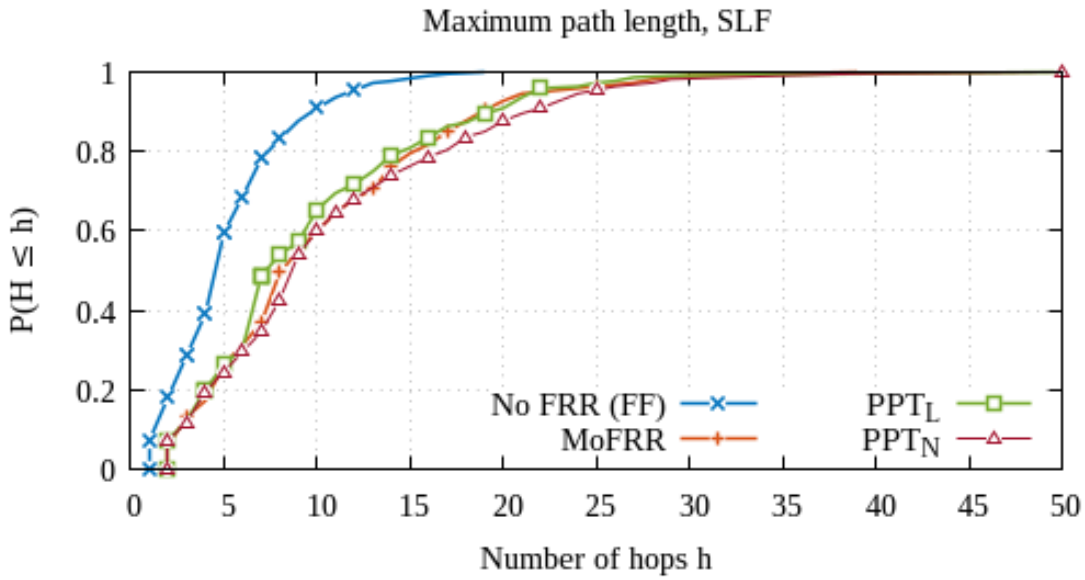


Figure 5.6: CDF of maximum path lengths for shortest path routing in the failure-free case and for BIER variants in SLF.

Figure 5.6 shows the maximum path lengths. We observe that most average path lengths are between 2 and 6 hops in the failure-free case. When we consider only average path lengths of affected multicast flows for all SLF, we mostly observe an average path stretch between 0.5 up to 1 hop for BIER-TE FRR mechanisms that bypass the traffic on shortest paths around the failure location. The values are rather small as the paths to some destinations are not extended. Most interesting is the finding that the path lengths for BIER MoFRR using MRTs are hardly longer than those for BIER-TE FRR although MRTs cause significant path stretch when used for the protection of unicast flows. This can be explained as follows. MRT routing topologies are used differently

for multicast compared to unicast. Unicast flows are carried over shortest paths until a failure occurs and are then switched to a working MRT routing topology. In contrast, multicast flows are transmitted in parallel over both MRT routing topologies without any switching. During failure-free operation, we consider the length of the shortest of both paths, in case of a failure, we consider the length of the working path, which may be shorter than the failed path.

The longest path prolongation in failure cases can be significant. Figure 5.6 shows the CDF of the maximum path length over all networks in the failure-free scenario and for SLF scenarios. The longest paths are mostly twice as long as in the failure-free scenario. Again, BIER with MoFRR and BIER-TE with PPT lead to about the same maximum path lengths. In case of node protection, path lengths are slightly longer than in case of link protection because traffic for NNHs is explicitly bypassed around the NH which may extend the backup path length.

Thus, MRT-based MoFRR for BIER does not cause excessive path length compared to shortest path routing, but backup paths of any FRR mechanism can lead to significant path stretch – not on average, but in the worst case.

5.4.5 Difference in Path Lengths for MoFRR

With MoFRR, egress routers measure the quality of the traffic stream received over the two independent paths and choose the traffic from the one with highest quality. In case of a failure, they detect the failure by the fact that the signal from one path is lost and choose the signal from the remaining path. The detection is faster and the switch-over smoother if packets from both paths are received simultaneously by the egress node or with only little delay difference. Delay difference may result from different path lengths. We quantify different path lengths by the ratio R of the longer and shorter length of the two redundant paths over the two MRT routing topologies.

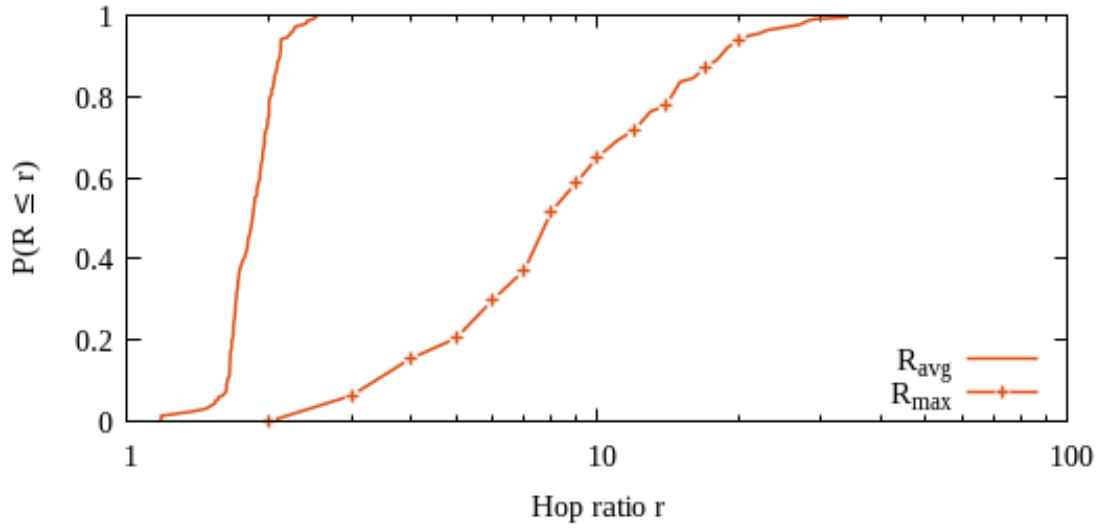


Figure 5.7: CDF of the average and maximum ratios of lengths of redundant paths for MoFRR in all networks.

Figure 5.7 shows the CDF of the average ratio $\mathrm{R}_{\mathrm{avg}}$and the maximum ratio $\mathrm{R}_{\mathrm{max}}$over all considered network topologies. The average ratio is about 2 for most topologies, i.e., mostly one of the path is twice as long as the other path. Maximum ratios are significantly larger. For 60% of the topologies, the maximum ratio is between 2 and 10, and for the other networks we observe maximum ratios between 10 and 33. As a result, the delay difference of both redundant paths for MoFRR can be significant for some leaves of some multicast trees. This makes failure detection more difficult than for simultaneously received signals, requires more buffer, and may cause more jitter or packet loss in case of a switch-over.

5.4.6 Load and Capacity Analysis

MoFRR for BIER duplicates all traffic at the source. $\mathrm{PPT_N}$ for BIER-TE sends traffic over multiple unicast tunnels to DS-NNHs in case of node protection while $\mathrm{BBE_N}$ for BIER-TE uses multicast for that purpose. This observation calls for an analysis of traffic loads and required transmission capacities.

We first consider the average load in the network, i.e., we summarize the rates of

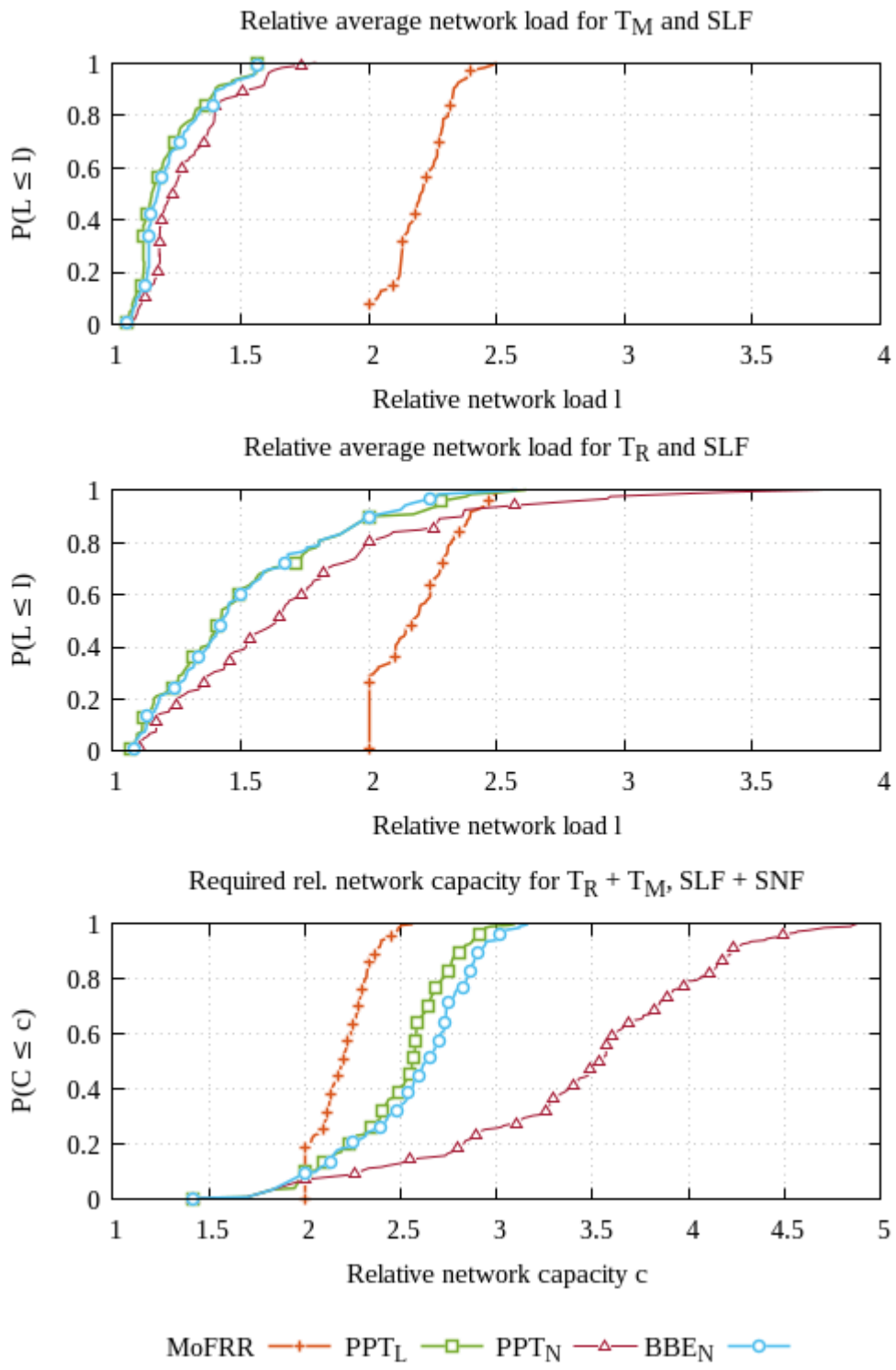


Figure 5.8: CDFs of relative average network loads (for different network topologies) and relative network capacities.

all links in the network in particular failure scenarios and average over all of them. To report load values from differently large networks in one figure, we normalize the average network load by the average network load in the failure-free case for shortest path routing which represents a lower bound. This yields an average network load relative to shortest path routing without failures that we call relative average network load. Figure 5.8 shows CDFs of the relative average network loads for SLF over all networks for BIER with MoFRR, PPT for BIER-TE with link and node protection, respectively, and for BBE for BIER-TE with node protection. One figure provides results for mesh topologies and another for ring topologies. BIER-TE protection methods cause in most mesh topologies an average load increase of up to 50%, but in most ring topologies a load increase of even up to 100% because more traffic is affected by failures and backup paths are longer in ring topologies than in mesh topologies. PPT with node protection causes more network load than with link protection because the PLR replicates the traffic and sends it over separate point-to-point tunnels to the DS-NNHs. This is unlike for BBE with node protection which leads to hardly more network load than PPT or BBE with link protection. With MoFRR and BIER we observe at least twice the relative average network load as in failure-free scenarios, and in some networks slightly larger values. The latter may be surprising with the notion of two redundant MRT-based multicast trees for MoFRR in mind. But this believe is wrong as MRT routing topologies do not necessarily form trees in some networks (cf. Section 5.2.2).

Second, we investigate the required network capacity to cover all SLF and SNF in a network. We compute the maximum traffic load per link over all considered failure scenarios and summarize these maximum rates of all links in a network. For easier comparison, we normalize the required network capacity by the network load in the failure-free scenario with shortest path routing and obtain a relative required network capacity. The third chart of Figure 5.8 illustrates the CDF of the relative required network capacity in all networks. Surprisingly, BIER with MoFRR requires by far the least capacity in most networks. BIER-TE variants require more capacity because traffic may be routed on very different paths depending on the specific failure so that high traffic rates can occur on many links. This is different with BIER with MoFRR: in case of a failure, traffic is

not rerouted but dropped so that the required capacity for a network equals its traffic load in the failure-free scenario. BIER-TE with PPT and node protection requires very large amounts of transmission resources, mostly 70% more than other BIER-TE variants and 2 – 3 times as much as BIER with MoFRR. The reason is that with PPT and node protection a PLR deviates traffic towards multiple DS-NNHs by unicast instead of multicast. This causes very large rates on backup paths for which capacity must be provided.

5.4.7 State and Header Overhead Considerations

BIER and BIER-TE require additional routing tables whose contents depends only on the topology and the routing in the underlay but not on supported multicast traffic which is good for scalability.

BIER without FRR requires only a single routing plane, e.g., shortest path routing. In contrast, BIER with MoFRR based on MRT routing topologies requires two additional routings planes whose state information scales with the number of nodes in the network. MRTs are adopted in the IETF and provide acceptable scaling in ISP networks [100]. They increase the routing state of normal IP routing by approximately 200%. Also the state information for BIER routing tables increases linearly with the number of nodes. The BIER header requires one bit for each node in the network. The default size of the BIER header is 256 bits (32 bytes) and was sufficient for all considered networks. Larger headers of up to 4096 bits are possible.

BIER-TE without FRR requires one bit per node in the network and one bit per link, which causes larger header overhead than BIER. Moreover, a controller is needed that constructs the multicast trees and encodes them in the packet headers. The FRR methods BBE and PPT have different scaling properties. BBE requires two BIER headers which results into an additional 32 bytes header overhead in our study because all networks were small enough to encode their links and nodes in a 256 bits header. PPT leverages point-to-point tunnels provided by a routing underlay. These tunnels need to be encoded as forward-adjacencies in the BIER header so that even more bits are needed. Moreover, the tunnels need to be provided by the routing underlay, increasing its complexity. To

protect against link failures, every unidirectional link in the network has to be protected so that the number of additional bits equals the number of unidirectional links. The networks under study had 56 links on average and at most 176 links. To protect against node failures, every node needs to be protected by additional tunnels. In the networks under study, 156 tunnels were needed on average and at most 716 tunnels. Thus, for the largest network, the BIER header requires for PPT 892 additional bits (112 bytes) for unicast tunnels. This causes more overhead than a small 32 bytes encapsulation header for BBE. Moreover, tunneling over the routing underlay may also add some header overhead. Therefore, BBE may be the most efficient FRR melthod for BIER-TE in terms of complexity and header overhead.

As BIER represents only nodes in the headers while BIER-TE represents both links and nodes, BIER-TE leads to larger header overhead than BIER which may be relevant for very large networks so that the resulting header size may be technically still feasible but not efficient. When fast protection is required, the complexity of the routing underlays may be problematic for BIER MoFRR because it requires three different routing planes whereas BIER-TE with BBE does not even depend on a routing underlay.

5.5 Software-Defined Multicast Architecture

In this section, we present the SDM architecture. We discuss the data plane requirements of this architecture.

5.5.1 Architecture Description

The generic architecture is illustrated in Figure 5.9. It is based on the typical SDN structure as discussed in Chapter 2. The data plane consists of P4 switches that are controlled by one or more controllers. The choice of the southbound interface is motivated in Section 5.5.2. The control plane configures the switches using the API that is generated by the P4 compiler of the P4 source code. The API is specific to the P4 program and in this case contains for example operations for the BIFT.

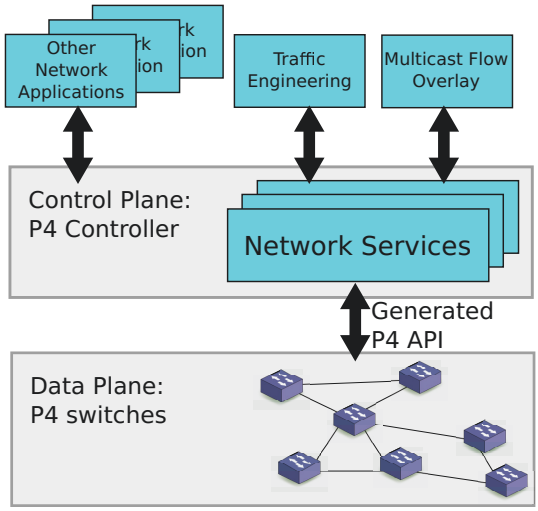


Figure 5.9: Software-Defined Multicast (SDM) architecture with P4 as southbound interface.

The northbound interface requires several specific components. The controllers have to interact with the so called Multicast Flow Overlay (MFO) as discussed in [142] that manages multicast subscriptions and can be provided by approaches such as defined in the RFCs 6513 or 6514. In addition, we propose the use of a “Traffic Engineering”

component that provides optimized paths for BIER-TE. FRR can be installed if traffic should be protected. The controller will install either traditional IP multicast, BIER, or BIER-TE depending on the specific use case and its requirements given by the MFO, TE, or other network applications.

5.5.2 Data Plane Requirements for SDN

The implementation of BIER is currently standardized for MPLS [147], OSPF [15] and IS-IS [16] to integrate BIER in current IP/MPLS networks. The MPLS encapsulation proposes the BIER header format as TLV.

There are two main options to implement BIER and BIER-TE in SDN: OpenFlow and the P4 language. Since version 1.2, OpenFlow supports different protocol headers, e.g. BIER, using the OXM. However, BIER is currently no standard header and must be supplied by vendors as an extension using OXM. Both multicast and FRR require group tables in OpenFlow. However, different group table types (*all* and *fast-failover*) are required and cannot be mixed. This increases the difficulty to implement BIER-TE with FRR. In addition, we think that the pipeline prior OpenFlow 1.5 is too restricted to support BIER and BIER-TE. OpenFlow 1.5 is not implemented in current OpenFlow switches and BIER will most likely require these optional features that may not be supported by all OpenFlow switches in the future.

Therefore, we consider P4 [31] as southbound interface for an SDN-BIER approach. P4 allows both the programmability of the data plane with regard packet format and to packet control through the switch's pipeline. New packet headers can be defined and P4 programs allow for complex pipeline operations. We illustrate the necessary features that needed to implement IP multicast, BIER and BIER-TE forwarding, and implement BIER-in-BIER encapsulation (BBE) as protection method for BIER-TE. We provide information about difficulties and missing flexibilities of the P4 language to implement our architecture.

5.6 Prototype and Testbed Demonstration

In this section, we present the testbed and illustrate various demo scenarios that can be executed simultaneously. We provide an introduction into the P4 language. Finally, we describe and explain the P4 implementation of SDM.

5.6.1 Testbed Description

We provide a P4 implementation of BIER and BIER-TE and release the P4 source code as open source under the Apache License. The software is available online [12] and explained in detail in Section 5.6.4. We use the same BIER header proposed for BIER MPLS encapsulation [147] for our implementation but embed BIER and BIER-TE into an Ethernet infrastructure instead of MPLS. We use the P4 reference software switch called "behavioral model" (*bmv2*) [148]. The reference switch has similar importance to P4 as OpenVSwitch to OpenFlow. Our testbed runs in a virtual machine with Ubuntu 14.04. We emulate various network scenarios in Mininet 2.3. The demo scenarios of our prototype are validated using multicast test tools developed by University of Virginia's Multimedia Networks Group [149].

We use a static network topology in our demonstration which is illustrated and controlled by a GUI written in Python and Qt. The GUI installs forwarding entries in the P4 switches using the Command-Line Interface (CLI) which is automatically generated of the P4 source code. The GUI acts as SDN controller but lacks some features such as topology detection, etc. Figure 5.10 illustrates the GUI. On the left side the network is shown. The links that currently transport traffic are highlighted in green. On the right side, there are four xterms-windows that contain the outputs of the mtools (msend and mreceive) used for the demo scenario. In the middle of the screen, a table dump of a BTAFT is given.

5.6.2 Demo Scenarios

In this section we describe the demo scenarios. All use cases can be simultaneously executed in the demonstration.

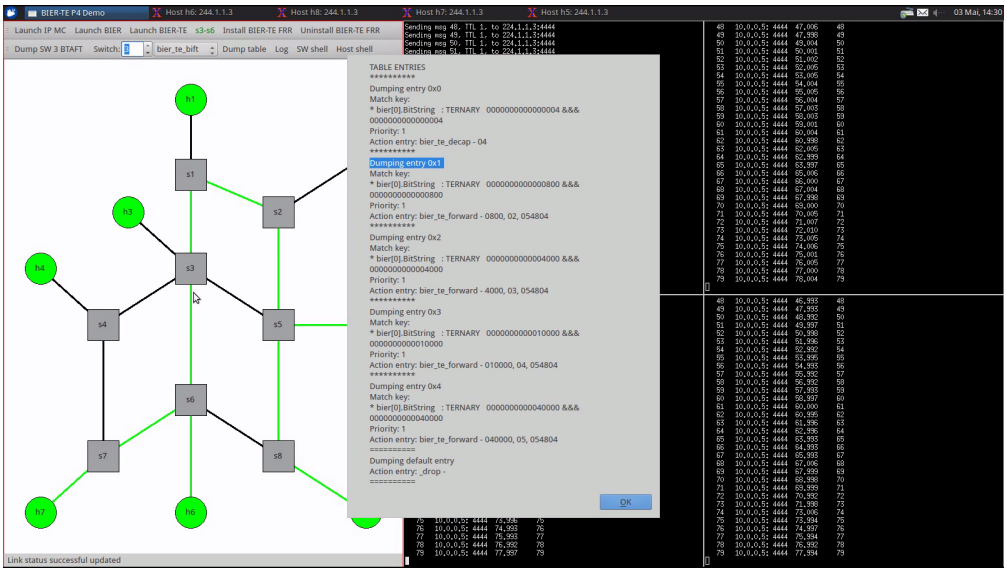


Figure 5.10: GUI of the prototype implementation. It allows to start all supported demo scenarios, query the table contents, and illustrates the network and the current traffic.

Traditional IP Multicast

We implement traditional IP multicast in the P4 switches. Each switch contains a mapping of multicast addresses to corresponding outgoing ports. Every switch that is part of the multicast tree requires a specific IP multicast table entry. Packets are duplicated correctly at branching points.

BIER Multicast

We configure the BIFT in all P4 switches according to the BIER IETF RFC [142]. For each multicast tree, one entry is installed at the ingress switch. The entry contains an action that encapsulates the BIER header which contains the egress nodes of the multicast tree. The multicast packet is forwarded along the shortest path tree towards the destinations using the BIFT entries. The BIER header is removed at the egress switches and the original packet is forwarded to the receiving host.

Explicit Paths Using BIER-TE

We configure the BIER-TE tables similarly to BIER multicast according to the BIER-TE draft. The ingress nodes encapsulate BIER headers that contains the links to traverse and the egress nodes. Packets do not have to follow the shortest paths. We leverage the EtherType field to distinguish IP, BIER, and BIER-TE multicast traffic to match packets against the corresponding forwarding tables in the P4 program.

Reliable Multicast Using BIER-TE FRR

BIER-TE FRR is based on rewriting the forwarding information contained in the BIER header in such a way that the packets are locally rerouted to the next-hops in case of failures. There are three different mechanisms proposed to implement the backup paths. We implemented BBE in our prototype for local reroute. We omit the two other variants because they are generally outperformed by BBE [10].

We configure the BTAFT to protect links in the network. We show that different multicast trees are rerouted correctly using the same BTAFT entries when a link failure occurs.

5.6.3 P4 Language

We provide an overview of the P4 language in this section. We focus on the P4 features required to implement SDM. This section explains the concepts of P4 which will be illustrated in great detail in Section 5.6.4 on the prototype implementation. We refer to the $P4_{14}$ specification from May 2017 [150]. A P4 program looks similar to a normal general-purpose program. It contains header and table definitions and describes how packets are processed. A P4 program is compiled to a so called *target*. A target is a device that performs the packet processing. Each target implements features specified in the standard and may provide target-specific functionalities. Not every P4 program will compile on all possible targets. We compile our P4 program to the P4 reference switch that provides a multicast feature which is not part of the standard. We implement IP multicast using this feature. The compilation process also generates an API that allows controllers to configure the device with appropriate forwarding entries.

A significant difference of P4 compared to OpenFlow is that the data plane can be easily extended. New headers can be simply defined similar to a records definition (e.g. `structs` in C++) known from general-purpose languages. Similarly, the programmer can specify metadata that act as helper variables during the execution of P4 program. Metadata is specific to a packet that is processed by the switch, it is initialized empty (value zero), and is independent of metadata of other packets. Header and metadata must be *instantiated* in the P4 program, i.e., the programmer specifies how many headers of a certain kind are accessible in the P4 program. For example, the P4 program for the SDM architecture specifies two headers for BIER and, thus, supports only one BBE encapsulation.

The packet processing is defined in the P4 program by specifying the *ingress* and *egress* control flows. The responsibilities of the ingress control flow are mainly to identify how a packet should be handled and on which egress the packet should leave the device. The egress pipeline is responsible for final packet processing after the egress was specified in the ingress.

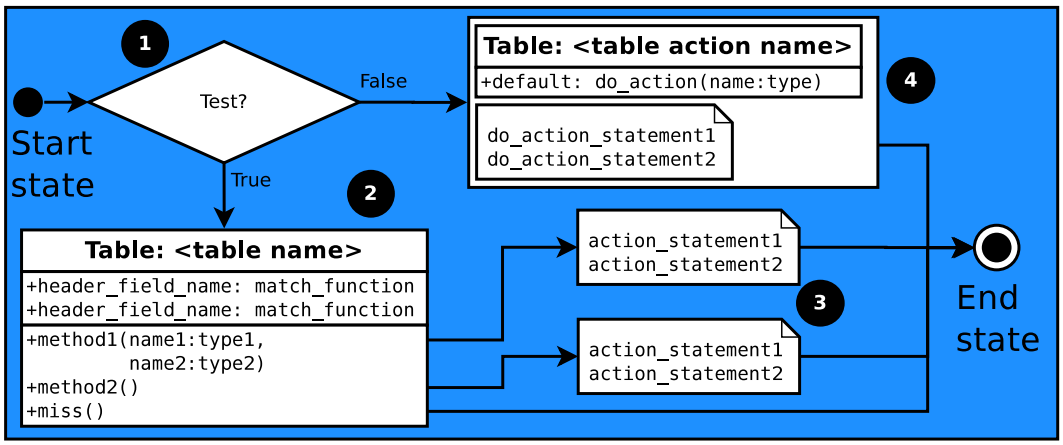


Figure 5.11: The control flow chart illustrates the most important control flow capabilities in P4 for the SDM implementation.

There are various operations possible in the control flow. Figure 5.11 illustrates a simple control flow diagram that contains the most important capabilities. Each chart contains a start state and at least one final state. This allows to separate a P4 program in smaller parts and focus on some aspects of the implementation, e.g., a control flow chart for IP forwarding as depicted in Figure 5.12 below.

(1) `if`-constructs allow to select different control flow actions dependent on the header or metadata fields, e.g., to test if a specific header field is empty. We illustrate them as diamonds as in typical flow charts. Depending on the outcome of the test, packet processing continues.

The most important control flow operation is to match header or metadata fields against a lookup table (2). We illustrate tables in a block that contains a name, a list of header fields, and a list of potential actions. The header fields may be matched exactly or as wildcard match. Each entry in the table is associated with an action that accepts zero or more arguments. Note that a P4 program allows to specify the following control flow depending on table hits, misses, or specific actions which is depicted as outgoing arrow of the actions. The arrows point to action blocks or other control flow elements. Each action block (3) contains one or more action statements. A P4 program cannot directly call actions in the control flow and relies on tables and associated actions. Therefore, we

introduce “table actions” (4) that do not contain match fields and only allow a default action to be assigned. We illustrate them by a single block that contains the table and an action block. Another important restriction in P4 programs is that a table can only be matched *once* per run. It prevents multiple matches in a BIFT for a single packet which is necessary for BIER and BIER-TE. We describe a solution based on packet cloning below.

Finally, we heavily rely on packet cloning, recirculation, and resubmission of packets to implement BIER an BIER-TE. Note that these operations are not intended to implement multicast as stated in the specification. However, we could not find another way to implement BIER in a scalable way, i.e., provide topology-specific and not flow-specific forwarding entries. In the following, we describe the operation and semantics of these operations and the necessary workarounds to implement SDM using the P4 reference switch *bmv2*. The specification states that packet cloning is possible from the ingress to the egress but *bmv2* only supports cloning to the egress at the time of this writing.

Packets can be cloned from the ingress to the egress by calling the action `clone_pkt_ingress_to_egress` anywhere during the ingress control flow. The egress pipeline will be executed twice. The first egress run uses the packet with the header and metadata state equal to the end of the ingress pipeline, i.e., all changes made to the packet are visible in the header fields. The second run operates on a packet clone of the initial header fields before the ingress has started. It is possible to keep the metadata from the end of the ingress pipeline.

Packet recirculation allows to begin the switch pipeline again with the resulting packet of the egress pipeline. The action `recirculate` may be called in the egress control flow. The packet will be parsed again and the ingress control flow is started. Metadata can be kept by specifying the appropriate fields. Packet resubmission the analogous operation for the ingress control flow. Note that both recirculation and resubmission of packets cancel any pending operations of the packet, i.e., packet clones or transmission of packets through a port are not performed.

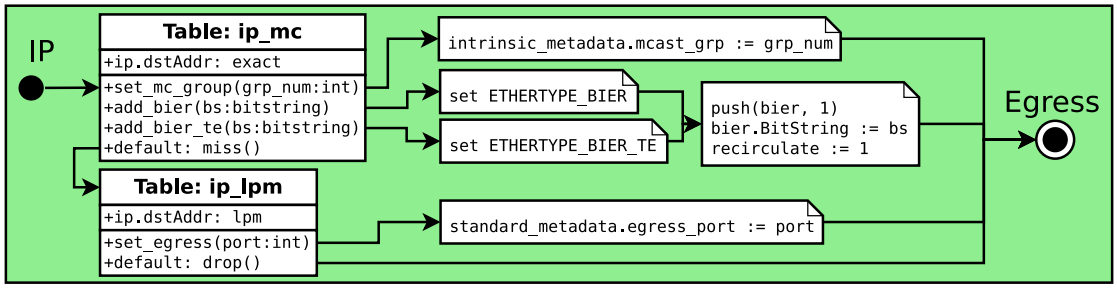


Figure 5.12: The IP control flow is applied to packets that do not contain a BIER header.

5.6.4 P4 Prototype Description

We describe the ingress and egress control flow of the P4 program. We omit the header and metadata definitions of the program but the header fields are implicitly given in the control flow. We illustrate the program in a control flow chart as discussed in Section 5.6.3.

The ingress control flow is separated in the *IP control flow*, *BIER control flow*, *BIER-TE control flow*, and *BIER-TE FRR control flow*. The responsibility of the ingress control flow is to identify the corresponding actions, assigning outgoing ports, and preparing the egress control flow. The egress control flow has three major responsibilities. First, it recirculates packet clones that must be matched again at the BIFT. Second, it makes changes made during the pipeline processing permanent for subsequent runs that start anew at the ingress. Finally, it sends packets out through the specified port if decided in the ingress ports.

In the following sections, we describe each control flow. Note that the ingress control flows are selected depending on the EtherType of the packet. Plain IP packets are processed to the IP control flow, and packets with EtherType BIER and BIER-TE are processed by the appropriate control flow.

5.6.4.1 IP Control Flow

IP packets are supported in the SDM architecture. Normal IP packets are simply forwarded and multicast packets can be either sent using IP multicast, BIER, and BIER-TE. Plain IP packets can be identified by the EtherType for IP packets. Figure 5.12 illustrates

the control flow for plain IP packets. The first step is to identify whether the packet is a multicast packet. This is done by matching the packet against the `ip_mc`-table. If the packet contains a multicast address that matches in the table, there are three possible actions. The first action (`set_mc_group`) is applied when it is an IP multicast packet. We leverage the P4 reference switch internal multicast feature by setting the appropriate switch-specific metadata field `intrinsic_metadata.mcast_grp` field. The metadata field corresponds to an entry of the switch internal multicast table which can be filled using the switch API. In the table, each multicast group is assigned a list of ports. The egress pipeline is started multiple times, one time for each port specified in the list. The egress port is set each time by the switch accordingly and the packet is sent through the correct port.

The second action is applied when the IP address is associated with BIER forwarding. The EtherType of the packet is set to BIER. Then, the BIER header is pushed and the bitstring is set appropriately. We also set a metadata bit (`recirculate`) that indicates recirculation of the packet in the egress pipeline. Recirculation in the egress pipeline will insert the BIER encapsulated packet in the ingress pipeline. Moreover, only recirculation is the only method we found to permanently store changes to the packet in subsequent flows through the pipeline, which is necessary for BIER forwarding and described in detail in the next section.

The third option is to encapsulate with BIER-TE. The procedure is identical but leverages a different EtherType to distinguish BIER and BIER-TE.

If the packet is not a multicast packet, we match the IP destination address against the `ip_lpm` lookup table. LPM is used to determine the egress port and the packet handled by the action `set_egress`.

5.6.4.2 BIER Control Flow

Figure 5.13 illustrates how BIER packets are processed in the pipeline. The packet in the ingress shows the contents of the BIER header. It contains bits (blue) matching a specific BIFT entry and bits that do not match the BIFT entry (red). The remaining bits (white) are unset in the header. The first step is to keep the matching bits, remove the

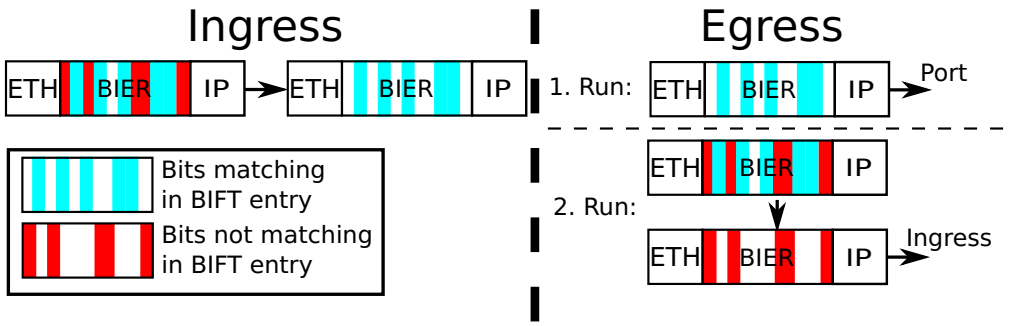


Figure 5.13: BIER packet processing.

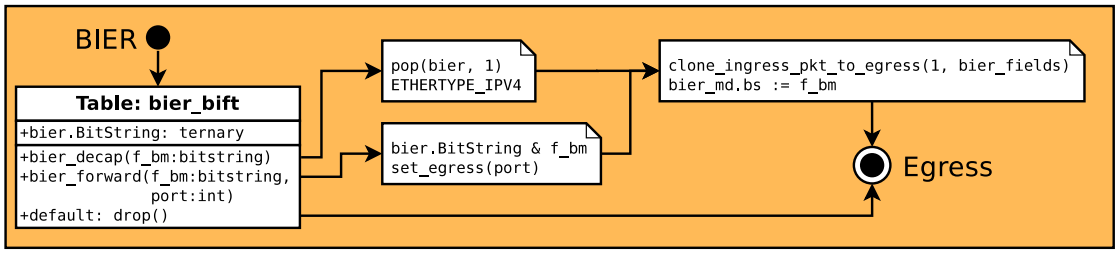


Figure 5.14: BIER control flow is applied to packets that are BIER encapsulated. The new EtherType BIER distinguishes BIER packets from plain IP and BIER-TE packets.

remaining bits from the header, and set the appropriate egress port. In addition, we use the `clone_ingress_to_egress` action to duplicate the packet. This action causes the egress pipeline to be started twice: the first run starts with the modified packet. The packet does not require any further processing and is immediately sent through the port. The second egress pipeline run starts with the unmodified packet of the ingress pipeline due to the semantics of the clone operation. Therefore, we have to remove the bits that matched in the ingress pipeline from the packet clone. The modified packet is recirculated to the ingress, which also makes the removal of the blue bits permanent in the subsequent iterations of the packet. The packet is recirculated until all bits are cleared and is then dropped.

Figure 5.14 shows the structure of the P4 program for BIER packet handling in the ingress control flow. The packet is matched against the BIFT. Two actions can be set for each BIFT entry. The action `bier_decap` is used for packets that leave the BIER

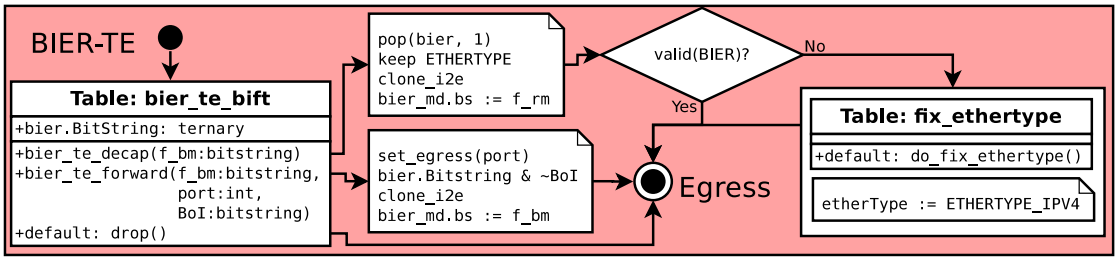


Figure 5.15: The BIER-TE control flow is structured similarly to the BIER control flow. It is called by the BIER-TE FRR control flow shown in Figure 5.16.

domain. The BIER header is popped and the IPv4 EtherType is set. The second action is `bier_forward` and corresponds to the BIER packet processing described above. The egress port is set and the matching IDs are updated using the forward bitmask (`f_bm`) [142]. The `f_bm` corresponds to the blue bits in Figure 5.13. We also save `f_bm` in a metadata variable that will be kept for the packet clone created by `clone_ingress_pkt_to_egress`. The bitmask will be used to extract the red IDs and recirculation to the ingress as illustrated in Figure 5.13. This is discussed in detail in Section 5.6.4.4 below.

5.6.4.3 BIER-TE Control Flow

In this section we explain BIER-TE and FRR for BIER-TE. We implement only the BBE method (Section 5.3.3). We first explain the BIER-TE forwarding procedure (Section 5.3.2) and afters the FRR procedure. Note that the FRR procedure is applied first and, which in turn executes the forwarding procedure.

The BIER-TE BIFT contains up to $n+1$ entries for a switch with n ports. One entry is established for packets leaving the BIER-TE domain causing the BIER-TE header to be removed. The remaining n entries are associated with each port of the switch. BIER-TE packets are forwarded by removing all Bits of Interest (BoI) before sending it through the port.

Figure 5.15 illustrates the BIER-TE forwarding procedure described above. The control flow is very similar to the BIER control flow and starts by matching the bitstring against the BIER-TE BIFT. The `bier_te_forward` action is almost analogous to

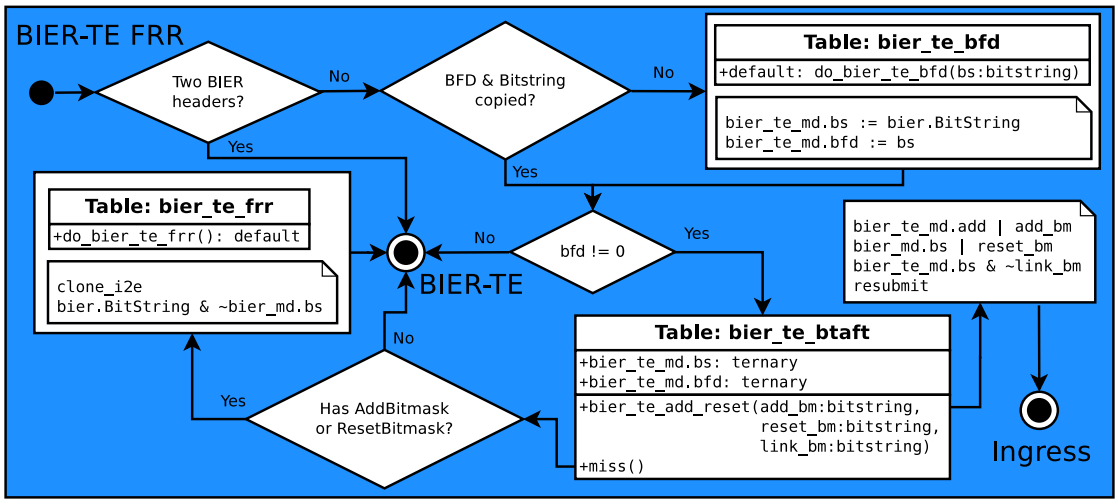


Figure 5.16: BIER-TE FRR control flow. It is applied if the packet is BIER encapsulated with EtherType BIER-TE.

the BIER one. The difference is that the sent packet has its BoI removed instead of the matching entry. Instead, the matching bit has to be removed from the packet clone to enable recursive processing of the packet which is performed in the egress control flow. The `bier_te_decap` action needs to respect the possibility of two BIER headers due to BBE FRR. We first pop the BIER header and prepare the packet clone and do not change the EtherType. Then, we check if there is still a BIER header in the packet using the `if`-construct and the `valid` function which checks if a specific header is present in the packet. If there is still a header, we transition to the egress control flow. Otherwise, we need to fix the EtherType to IPv4. P4 does not allow to call actions directly within the control flow. Thus, we provided a workaround of an empty table `fix_ethertype` that contains a default action that sets the EtherType to IPv4. Since the table does not contain any entries, the action is always applied. Finally, the egress control flow starts.

Figure 5.16 illustrates the P4 program part that performs FRR for BIER-TE. This part is directly applied in the ingress when the EtherType is BIER-TE and runs before the normal BIER-TE forwarding described above. To apply FRR, the P4 program requires access to the status of the output ports. However, the port status cannot be accessed using the switches metadata. Therefore, we implemented a BFD component that measures the

status of all ports and updates a table (`bier_te_bift`) in such a way that it contains the port status in form of a bitstring.

The first step in the FRR control flow is to check if two BIER headers are in the packet. If there are two BIER headers we skip the FRR part and start with the normal forwarding procedure since the approach only supports one BIER-in-BIER encapsulation. Otherwise, we copy the BFD content (`bfd`) and the bitstring (`bs`) into metadata. If there are failed adjacencies (`bfd != 0`), we match on the copied bitstring `bs` in the BTAFT. Each match in the BTAFT corresponds to a backup path – a single backup path to the NH for single link failures or a part of the backup path towards a single DS-NNH for node failures. The backup paths are merged in the Add- and ResetBitmasks and the failed adjacency removed from the copied bitstring `bs`. The packet is resubmitted to the ingress keeping the metadata. These steps are repeated until there are no failed adjacencies left in the copied bitstring.

When an Add- or ResetBitmask was computed through the BTAFT, we create a packet clone that will be encapsulated with the contents of the AddBitmask. Since actions cannot be called directly, we require an empty table (`bier_te_frr`) with a default action. The encapsulation with a second BIER header will be performed in the egress control flow. In addition, we remove the bits collected in the ResetBitmask. Then, normal BIER-TE forwarding is executed.

5.6.4.4 Egress Control Flow

The egress control flow is responsible for three main cases which are illustrated in Figure 5.17. The first case addresses packet clones. As described in the sections above, we create packet clones when a match in the BIER or BIER-TE BIFT occurred, or when a ResetBitmask has to be applied. The packet clone should be inserted in the ingress but must have the matched bits removed or the ResetBitmask applied. The corresponding bits are saved in the metadata `bier_md.bs`. The packet clone is then recirculated to the ingress. Since the control flow does not allow to call actions directly, we leverage the workaround with an additional empty table again.

The second case is related to BBE FRR. We push a new BIER header with the con-

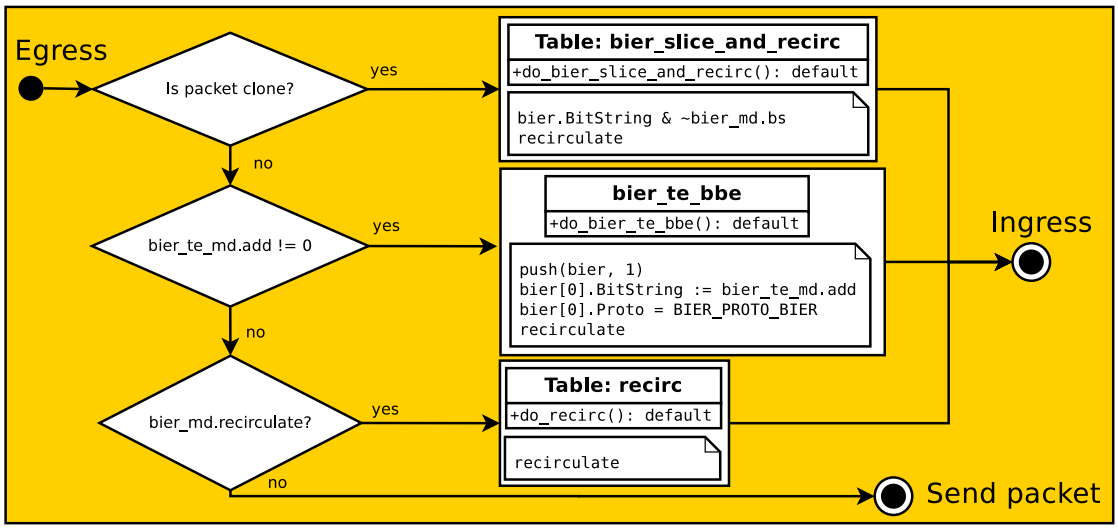


Figure 5.17: Egress control flow.

tents of the AddBitmask. The packet is recirculated to the ingress and will be forwarded through the normal BIER-TE forwarding procedure.

The third and final case handles the encapsulation of IP packets to BIER packets. The packet has to be recirculated to the ingress to make the BIER encapsulation permanent in the packet. The recirculation is necessary to workaround the effects of packet clones. Consider the following P4 program when a multicast packet is encapsulated to a BIER packet in the IP control flow. If the BIER control flow is directly called without recirculating the packet to the ingress, a packet clone will be created using `clone_ingress_pkt_to_egress`. The packet clone will be inserted in the egress pipeline with the header fields equal to the start of the pipeline: the BIER header is not present in the packet clone. Recirculation simplifies the P4 program significantly by ensuring the correct header state for packet clones.

If none of the cases apply, the packet does not need further processing and will be sent through the assigned port.

5.7 Summary

In this chapter, we discussed how scalable and resilient multicast can be implemented in SDN. We leveraged the BIER and BIER-TE architectures to achieve scalable multicast. We proposed various resilience mechanisms and compared them on 220 network topologies. We suggested to implement MoFRR for BIER leveraging the calculation of MRT topologies to obtain redundant multicast forwarding structures. For BIER-TE FRR we suggested three different methods: header modification (HM), unicast tunneling (PPT), BIER-in-BIER encapsulation (BBE).

BIER MoFRR implements 1+1 protection and, therefore, causes at least double traffic load during operation compared to without protection. However, it requires mostly less additional capacity than FRR for BIER-TE because it does not reroute traffic in failure cases. Although the original MRT method is known for possibly long backup paths, path lengths resulting from BIER with MoFRR do not exhibit significantly more path stretch than with FRR for BIER-TE. An implementation challenge may be a possibly large difference in length of the two redundant paths. Below the line, MoFRR leveraging MRT calculation seems an effective FRR option for BIER.

HM is the simplest method for BIER-TE FRR in terms of technology. However, it is not effective because it loses about the same amount of traffic as without protection just to avoid duplicate packets. In contrast, PPT and BBE can protect against all failures if topologically possible. PPT configured for node protection may lead to excessive capacity requirements, large header overhead, and complicates the routing underlay. As BBE avoids these drawbacks, it requires only an additional BIER header of moderate size, and does not need support from the routing underlay, we recommend BBE as preferred protection method for BIER-TE.

We proposed the SDM architecture and discussed its data plane requirements. We provided a demonstration testbed and a prototype implementation. The prototype is written in the P4 language and described in detail. We showed workarounds that were necessary to implement BIER, BIER-TE, and the BBE method for FRR. The demonstration allows to execute IP multicast, BIER multicast, and BIER-TE multicast simultaneously and, thus, shows the feasibility of our proposed SDM architecture.

6 Conclusion

The SDN architecture has driven network innovation forward and provides the opportunity to evolve communication networks faster and more efficient than before. While SDN currently is already used in datacenters, applying SDN to general communication and ISP networks imposes new challenges with regard to scalability and resilience. The SDN architecture bases on the separation of the data plane and the control plane. Network failures require a logically-centralized control plane to reconfigure the network. The separation and the design choices may impose challenging circumstances that delay or prevent network reconfiguration which increases the amount of traffic lost in failure cases. Moreover, current OpenFlow switches – that are most commonly used to implement SDN – have limited space in their flow tables. Fine-grained rules required for network applications and large routing tables challenge the memory capacities of OpenFlow switches.

The first part of this dissertation addresses inter-domain routing in OpenFlow-based SDN. We developed a wildcard compression mechanism to reduce the required state in OpenFlow switches to support large routing tables. We presented the architectural design, and considered important practical issues: incremental table updates and compression speed. We analyzed the effectiveness of the approach and found that compression routing tables is challenging and time consuming. We significantly improved compression speed by sacrificing only little compression efficiency. However, we found an overall compression efficiency of approximately 17 – 20% which does not solve the state problem itself. Yet, our method is orthogonal to other already studied approaches and can be combined to increase the network throughput.

The second part of this dissertation discusses resilience in SDN networks and is based on FRR principle. We propose various different FRR mechanism that we as-

sumed are appropriate candidates for SDN networks. We considered several existing FRR mechanism: Loop-free alternates (LFAs), MPLS FRR, and Maximally Redundant Trees (MRTs). We provide novel extensions or adjustments to LFAs and MRTs that are possible leveraging the SDN architecture to enhance their characteristics. LD-LFAs improve the coverage of LFAs significantly and require only one additional rule per switch. We developed destination-specific MRTs (dMRTs) that share the data plane implementation with MRTs but reduce the length of the backup paths. Moreover, we proposed dual sink trees (DSTs) that reuse the underlying structure of MRTs but sacrifice path length for fewer forwarding entries. We did not propose changes to MPLS FRR. Finally, we developed a novel FRR scheme for OpenFlow-based SDN called Load-Dependent Flow Splitting (LDFS) that combines traffic engineering with backup paths.

We analyzed the proposals on various networks obtained from the Topology Zoo that consists of several hundreds of commercial and research wide area networks. We found that LDFS achieves capacity savings when traffic overloads occur individually but not in combinations with failures. We generally do not recommend LDFS as resilience mechanism for SDN due to its complexity and lack of efficiency. LD-LFAs require the least state and provide significantly higher protection compared to simple LFAs. LD-LFAs are suited for networks that cannot provide space for backup forward entries and do not require full coverage. The results for the remaining resilience methods that provide full coverage are as follows. MPLS FRR lead to the backup paths shortest paths but has the highest state requirements. dMRTs clearly outperform simple MRTs and provide a good balance between path lengths and state requirements. Path lengths are on average almost as short as with MPLS but noticeably longer in worst cases. DSTs require the least state of the considered methods but causes longer paths especially for traffic flows that are not affected by any failure. Therefore, we recommend dMRTs as resilience method for SDN in general. DSTs and MPLS should be used when state or path lengths have highest priority, respectively.

The final part of this dissertation addresses scalable and resilient multicast for SDN-based networks. We propose a software-defined multicast architecture that leverages Bit Indexed Explicit Replication (BIER) and Traffic Engineering for BIER (BIER-TE) to

achieve multicast without flow specific state. We propose various resilience methods for BIER and BIER-TE and evaluate and compare them on the Topology Zoo. In addition, we provide a prototype implementation and testbed demonstration that shows the feasibility of our solution.

Acronyms

ACL	Access Control List	12
ACK	Acknowledgment	40
ADAG	Almost Directed Acyclic Graph	71
API	Application Programming Interface	1
APL	Apache Public License	25
BAT	Balance All Traffic	82
BBE	BIER-in-BIER encapsulation	144
BET	Balance Excess Traffic	82
BFD	Bidirectional Forwarding Detection	6
BFR	Bit forwarding router	123
BGP	Border Gateway Protocol	6
BIER-TE	Traffic Engineering for BIER	121
BIER	Bit Indexed Explicit Replication	4
BIFT	Bit Indexed Forwarding Table	123
BoI	Bits of Interest	127
BTAFT	BIER-TE Adjacency Forwarding Table	128
CAPEX	Capital Expenditure	32
CDF	Cumulative distribution function	102
CIDR	Classless Inter-Domain Routing	54
CLI	Command-Line Interface	145
DAG	Directed acyclic graph	71
DCN	Datacenter network	38
dMRT	Destination-specific MRT	59
DS-NNH	Downstream next-next-hops	128

DSCP Differentiated services code point 79
DSC Downstream condition 64
DST Dual sink tree 59
DS Differentiated Services 17
ECMP Equal-cost multipath 63
EGP Exterior Gateway Protocol 6
ECN Explicit Congestion Notification 17
EPL Eclipse Public License 25
ESP Encrypted security payload 23
FF Failure-free 132
FIB Forwarding Information Base 35
ForCES Forwarding and Control Element Separation 13
FRR Fast Reroute 2
GPL GNU General Public License 24
GUI Graphical User Interface
HSR High Availability Seamless Redundancy 40
HM Header modification 158
IDAG Independent Directed Acyclic Graph 39
IETF Internet Engineering Task Force 3
IGP Interior Gateway Protocol 6
IS-IS Intermediate System to Intermediate System 6
ISP Internet service provider 6
ITree Independent tree 39
LDFS Load-Dependent Flow Splitting 59
LFA Loop-free alternate 10
LD-LFA Loop-detecting LFA 60
LFB Logical Functional Block 13
LFC Loop-free condition 63
LGPL GNU Lesser General Public License 25
LISP Locator/ID Separation Protocol 13

LPM Longest Prefix Match 44
MFO Multicast Flow Overlay 143
MoFRR Multicast only Fast Reroute 124
MPLS Multi-protocol label switching 4
MPR Multipath Routing 113
MRC Multiple routing configuration 10
MRT Maximally Redundant Tree 3
NaaS Network-as-a-Service 28
NH Next-hop 8
NNH Next-next-hop 9
NPC Node-protecting condition 64
OAM Operations and management 22
ONF Open Networking Foundation 1
OPEX Operational Expenditure 32
ORTC Optimal Routing Table Constructor 36
OSC Optical Supervisory Channel 32
OSPF Open Shortest Path First 6
OTN Optical Transport Networks 24
OXM OpenFlow Extensible Match 22
OXS OpenFlow eXtensible Statisitcs 24
PBB Provider Backbone Bridge 23
PCE Path Computation Element 13
PIM Protocol Independent Multicast 122
PLR Point of local repair 8
PRP Parallel Redundancy Protocol 40
PPT Unicast tunneling 158
QoS Quality of Service 18
RCP Routing Control Platforms 45
RET Redirect Excess Traffic 82
RIB Routing Information Base 36

SDH Synchronous Digital Hierarchy 32
SDN Software-Defined Networking 1
SDM Software-Defined Multicast 121
SDX Software-defined Internet exchange point 46
SLF Single link failures 83
SNF Single node failures 83
SONET Synchronous Optical Networking 32
SPF Shortest path first 72
SPR Single-shortest path routing 114
TCP Transmission Control Protocol 40
TCAM Ternary-Content Addressable Memory 1
TE Traffic Engineering 80
TLV Type-Length-Value 6
TTL Time-To-Live 21
VLSI Very-large-scale integration 49

Bibliography and 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] "SASER: Safe and Secure European Routing," Feb. 2017. [Online]. Available: http://projects.celticplus.eu/saser/

[3] "DFG - GEPRIS - Untersuchung von neuen Routing-Architekturen für das Internet," Feb. 2017. [Online]. Available: http://gepris.dfg.de/gepris/projekt/214843191

[4] W. Braun and M. Menth, "Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices," *Future Internet*, vol. 6, no. 2, pp. 302–336, 2014. [Online]. Available: http://www.mdpi.com/1999-5903/6/2/302

[5] ——, "Wildcard Compression of Inter-Domain Routing Tables for OpenFlow-Based Software-Defined Networking," in *Proceedings of the European Workshop on Software Defined Networks (EWSDN)*, Sep. 2014, pp. 25–30.

[6] ——, "Scalable Resilience for Software-Defined Networking Using Loop-Free Alternates with Loop Detection," in *IEEE Workshop on Management Issues in SDN, SDI and NFV (MISSION) in conjunction with IEEE Conference on Network Softwarization (NetSoft)*, Apr. 2015.

[7] ——, "Load-Dependent Flow Splitting for Traffic Engineering in Resilient OpenFlow Networks," in *Workshop on Software-Defined Networking and Network Function Virtualization for Flexible Network Management (SDNFlex), collocated with ITG/GI Conference on Networked Systems (NetSys)*, Cottbus, Germany, Mar. 2015.

[8] ——, "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, no. 3, pp. 470 – 490, 2016.

[9] W. Braun, D. Merling, and M. Menth, "Destination-Specific Maximally Redundant Trees: Design, Performance Comparison, and Applications," in *International Workshop on the Design of Reliable Communication Networks (DRCN)*, Feb. 2018.

[10] W. Braun, M. Albert, T. Eckert, and M. Menth, "Performance Comparison of Resilience Mechanisms for Stateless Multicast Using BIER," in *IFIP/IEEE Symposium on Integrated Management (IM)*, May 2017.

[11] W. Braun, J. Hartmann, and M. Menth, "Scalable and Reliable Software-Defined Multicast with BIER and P4," in *IFIP/IEEE Symposium on Integrated Management (IM)*, May 2017.

[12] W. Braun and J. Hartmann, "P4 Bit Forwarding Router (p4-bfr)," Dec. 2017. [Online]. Available: https://bitbucket.org/wb-ut/p4-bfr

[13] T. Eckert, G. Cauchie, W. Braun, and M. Menth, "Traffic Engineering for Bit Index Explicit Replication BIER-TE," Internet Engineering Task Force, Internet-Draft draft-ietf-bier-te-arch-00, Jan. 2018.

[14] ——, "Protection Methods for BIER-TE," Internet Engineering Task Force, Internet-Draft draft-eckert-bier-te-frr-02, Jun. 2017.

[15] P. Psenak, N. Kumar, I. Wijnands, A. Dolganow, T. Przygienda, Z. J. Zhang, and S. Aldrin, "OSPF Extensions for BIER," Internet Engineering Task Force, Internet-Draft draft-ietf-bier-ospf-bier-extensions-12, Feb. 2018.

[16] L. Ginsberg, T. Przygienda, S. Aldrin, and Z. J. Zhang, "BIER support via ISIS," Internet Engineering Task Force, Internet-Draft draft-ietf-bier-isis-extensions-07, Feb. 2018.

[17] A. Atlas, G. Swallow, and P. Pan, "Fast Reroute Extensions to RSVP-TE for LSP Tunnels," RFC 4090, May 2005.

[18] R. Martin and M. Menth, "Backup Capacity Requirements for MPLS Fast Reroute," in 7^{th} *ITG-Fachtagung Photonische Netze*, Leipzig, Germany, Apr. 2006, pp. 95–102.

[19] A. D. Zinin, "Basic Specification for IP Fast Reroute: Loop-Free Alternates," RFC 5286, Sep. 2008.

[20] L. Csikor, J. Tapolcai, and G. Retvari, "Optimizing IGP link costs for improving IP-level resilience with Loop-Free Alternates," *Computer Communications*, vol. 36, no. 6, pp. 645 – 655, Mar. 2013.

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

[22] A. Bashandy, C. Filsfils, B. Decraene, S. Litkowski, and P. Francois, "Topology Independent Fast Reroute using Segment Routing," Internet Engineering Task Force, Internet-Draft draft-bashandy-rtgwg-segment-routing-ti-lfa-01, Jul. 2017.

[23] S. Bryant, S. Previdi, and M. Shand, "A Framework for IP and MPLS Fast Reroute Using Not-Via Addresses," RFC 6981, Aug. 2013.

[24] S. Rai, B. Mukherjee, and O. Deshpande, "IP Resilience within an Autonomous System: Current Approaches, Challenges, and Future Directions," *IEEE Communications Magazine*, vol. 43, no. 10, pp. 142–149, Oct. 2005.

[25] A. Raj and O. Ibe, "A Survey of IP and Multiprotocol Label Switching Fast Reroute Schemes," *Computer Networks*, vol. 51, no. 8, pp. 1882–1907, 2007.

[26] 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.

[27] M. Menth and W. Braun, "Performance Comparison of Not-Via Addresses and Maximally Redundant Trees (MRTs)," in *IFIP/IEEE Symposium on Integrated Management (IM)*, Ghent, Belgium, Apr. 2013.

[28] 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 (ICC)*, June 2014, pp. 1131–1136.

[29] 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.

[30] "Open Networking Foundation," https://www.opennetworking.org/, Feb. 2017. [Online]. Available: https://www.opennetworking.org/

[31] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming Protocol-Independent Packet Processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014.

[32] A. Doria, J. H. Salim, R. Haas, H. Khosravi, W. Wang, L. Dong, R. Gopal, and J. Halpern, "Forwarding and Control Element Separation (ForCES) Protocol Specification," RFC 5810 (Proposed Standard), Internet Engineering Task Force, Mar. 2010.

[33] L. Yang, R. Dantu, T. Anderson, and R. Gopal, "Forwarding and Control Element Separation (ForCES) Framework," RFC 3746 (Informational), Internet Engineering Task Force, Apr. 2004.

[34] S. Hares, "Analysis of Comparisons between OpenFlow and ForCES," Internet Engineering Task Force, Internet-Draft draft-hares-forces-vs-openflow-00, Jul. 2012.

[35] E. Haleplidis, S. Denazis, O. Koufopavlou, J. Halpern, and J. H. Salim, "Software-Defined Networking: Experimenting with the Control to Forwarding Plane Interface," in *Proceedings of the European Workshop on Software Defined Networks (EWSDN)*, Oct. 2012, pp. 91–96.

[36] T. V. Lakshman, T. Nandagopal, R. Ramjee, K. Sabnani, and T. Woo, "The SoftRouter Architecture," in *Proceedings of the ACM Workshop on Hot Topics in Networks (HotNets)*, Nov. 2004.

[37] H. M. Zheng and X. Zhang, "Path Computation Element to Support Software-Defined Transport Networks Control," Internet Engineering Task Force, Internet-Draft draft-zheng-pce-for-sdn-transport-00, Feb. 2014.

[38] A. Rodriguez-Natal, A. Cabellos-Aparicio, S. Barkai, V. Ermagan, D. Lewis, F. Maino, and D. Farinacci, "Software Defined Networking extensions for the Locator/ID Separation Protocol," Internet Engineering Task Force, Internet-Draft draft-rodrigueznatal-lisp-sdn-00, Feb. 2014.

[39] J. Rexford, M. J. Freedman, N. Foster, R. Harrison, C. Monsanto, M. Reitblatt, A. Guha, N. P. Katta, J. Reich, and C. Schlesinger, "Languages for Software-Defined Networks," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 128–134, Feb. 2013.

[40] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: A Network Programming Language," in *Proceedings of*

the ACM SIGPLAN International Conference on Functional Programming, Sep. 2011.

[41] C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker, "Composing Software-Defined Networks," in *Proceedings of the USENIX Syposium on Networked Systems Design & Implementation (NSDI)*, 2013, pp. 1–14.

[42] A. Voellmy, H. Kim, and N. Feamster, "Procera: A Language for High-Level Reactive Network Control," in *Proceedings of the ACM Workshop on Hot Topics in Software Defined Networks (HotSDN)*, Oct. 2012, pp. 43–48.

[43] F. M. Facca, E. Salvadori, H. Karl, D. R. Lopez, P. A. A. Gutierrez, D. Kostic, and R. Riggio, "NetIDE: First Steps towards an Integrated Development Environment for Portable Network Apps," in *Proceedings of the European Workshop on Software Defined Networks (EWSDN)*, 2013, pp. 105–110.

[44] OpenFlow Switch Consortium and others, "OpenFlow Switch Specification Version 1.0.0," December 2009.

[45] ——, "OpenFlow Switch Specification Version 1.1.0," Decemeber 2011.

[46] ——, "OpenFlow Switch Specification Version 1.2.0," Decemeber 2011.

[47] ——, "OpenFlow Switch Specification Version 1.3.5," April 2015.

[48] ——, "OpenFlow Switch Specification Version 1.4.1," April 2015.

[49] ——, "OpenFlow Switch Specification Version 1.5.1," Mar. 2015.

[50] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "NOX: Towards an Operating System for Networks," *ACM SIGCOMM Computer Communications Review*, vol. 38, no. 3, pp. 105–110, Jul. 2008.

[51] "NOXrepo.org," Nov. 2013. [Online]. Available: http://www.noxrepo.org

[52] D. Erickson, "The Beacon OpenFlow Controller," in *Proceedings of the ACM Workshop on Hot Topics in Software Defined Networks (HotSDN)*, 2013, pp. 13–18.

[53] "Project Floodlight: Open Source Software for Building Software-Defined Networks," Nov. 2013. [Online]. Available: http://www.projectfloodlight.org/floodlight/

[54] Z. Cai, A. L. Cox, and T. S. Eugene Ng, "Maestro: Balancing Fairness, Latency and Throughput in the OpenFlow Control Plane," Rice University, Tech. Rep., 2011.

[55] "NodeFLow OpenFlow Controller," Nov. 2013. [Online]. Available: https://github.com/gaberger/NodeFLow

[56] "Trema: Full-Stack OpenFlow Framework in Ruby and C," Nov. 2013. [Online]. Available: http://trema.github.io/trema/

[57] "Opendaylight," Nov. 2013. [Online]. Available: http://www.opendaylight.org/

[58] "Ryu SDN Framework," Nov. 2017. [Online]. Available: https://osrg.github.io/ryu/

[59] OpenFlow Switch Consortium and others, "OpenFlow Management and Configuration Protocol 1.2 (OF-Config 1.2)," April 2015.

[60] B. Heller, R. Sherwood, and N. McKeown, "The Controller Placement Problem," in *Proceedings of the ACM Workshop on Hot Topics in Software Defined Networks (HotSDN)*, 2012, pp. 7–12.

[61] D. Hock, M. Hartmann, S. Gebert, M. Jarschel, T. Zinner, and P. Tran-Gia, "Pareto-Optimal Resilient Controller Placement in SDN-based Core Networks," in *International Teletraffic Congress (ITC)*, Sep. 2013, pp. 1–9.

[62] A. Tootoonchian and Y. Ganjali, "HyperFlow: A Distributed Control Plane for OpenFlow," in *Proceedings of the USENIX Workshop on Research on Enterprise Networking (WREN)*, Apr. 2010, pp. 3–3.

[63] S. H. Yeganeh and Y. Ganjali, "Kandoo: A Framework for Efficient and Scalable Offloading of Control Applications," in *Proceedings of the ACM Workshop on Hot Topics in Software Defined Networks (HotSDN)*, 2012, pp. 19–24.

[64] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, "Logically Centralized?: State Distribution Trade-offs in Software Defined Networks," in *Proceedings of the ACM Workshop on Hot Topics in Software Defined Networks (HotSDN)*, 2012, pp. 1–6.

[65] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, and P. Tran-Gia, "Modeling and Performance Evaluation of an OpenFlow Architecture," in *Proceedings of the International Teletraffic Congress (ITC)*, 2011, pp. 1–7.

[66] M. P. Fernandez, "Comparing OpenFlow Controller Paradigms Scalability: Reactive and Proactive," in *Proceedings of the International Conference on Advanced Information Networking and Applications (AINA)*, Mar. 2013, pp. 1009–1016.

[67] "CIDR REPORT," July 2013. [Online]. Available: http://www.cidr-report.org/as2.0/

[68] N. Sarrar, S. Uhlig, A. Feldmann, R. Sherwood, and X. Huang, "Leveraging Zipf's Law for Traffic Offloading," *ACM SIGCOMM Computer Communications Review*, vol. 42, no. 1, pp. 16–22, Jan. 2012.

[69] N. Sarrar, A. Feldmann, S. Uhrig, R. Sherwood, and X. Huang, "Towards Hardware Accelerated Software Routers," in *Proceedings of the ACM CoNEXT Student Workshop*, 2010, pp. 1–2.

[70] M. Soliman, B. Nandy, I. Lambadaris, and P. Ashwood-Smith, "Source Routed Forwarding with Software Defined Control, Considerations and Implications," in *Proceedings of the ACM CoNEXT Student Workshop*, 2012, pp. 43–44.

[71] X. Zhang, P. Francis, J. Wang, and K. Yoshida, "Scaling IP Routing with the Core Router-Integrated Overlay," in *Proceedings of the IEEE International Conference on Network Protocols (ICNP)*, 2006, pp. 147–156.

[72] H. Ballani, P. Francis, T. Cao, and J. Wang, "ViAggre: Making Routers Last Longer!" in *Proceedings of the ACM Workshop on Hot Topics in Networks (HotNets)*, Oct. 2008, pp. 109–114.

[73] J. Heitz, R. Raszuk, L. Zhang, X. Xu, and A. Lo, "Simple Virtual Aggregation (S-VA)," RFC 6769, Oct. 2012. [Online]. Available: https://rfc-editor.org/rfc/rfc6769.txt

[74] A. Masuda, C. Pelsser, and K. Shiomoto, "SpliTable: Toward Routing Scalability Through Distributed BGP Routing Tables," *IEICE Transaction on Communications*, vol. E94-B, no. 1, Jan. 2011.

[75] G. Rètvàri, J. Tapolcai, A. Kõrösi, A. Majdàn, and Z. Heszberger, "Compressing IP Forwarding Tables: Towards Entropy Bounds and Beyond," in *Proceedings of the ACM SIGCOMM*, Aug. 2013, pp. 111–122.

[76] R. Draves, C. King, V. Srinivasan, and B. Zill, "Constructing Optimal IP Routing Tables," in *Proceedings of the IEEE Infocom*, Mar. 1999, pp. 88–97.

[77] T. Yang, B. Yuan, S. Zhang, T. Zhang, R. Duan, Y. Wang, and B. Liu, "Approaching Optimal Compression with Fast Update for Large Scale Routing Tables," in *IEEE International Workshop on Quality of Service*, 2012, pp. 32:1–32:9.

[78] Z. A. Uzmi, M. Nebel, A. Tariq, S. Jawad, R. Chen, A. Shaikh, J. Wang, and P. Francis, "SMALTA: Practical and Near-optimal FIB Aggregation," in *ACM Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, 2011, pp. 29:1–29:12.

[79] Y. Liu, X. Zhao, K. Nam, L. Wang, and B. Zhang, "Incremental Forwarding Table Aggregation," in *IEEE Globecom*, Dec. 2010, pp. 1–6.

[80] S. Singh, F. Baboescu, G. Varghese, and J. Wang, "Packet Classification Using Multidimensional Cutting," in *ACM SIGCOMM*, 2003, pp. 213–224.

[81] A. X. Liu, C. R. Meiners, and E. Torng, "TCAM Razor: A Systematic Approach Towards Minimizing Packet Classifiers in TCAMs," *IEEE/ACM Transactions on Networking*, vol. 18, no. 2, pp. 490–500, Apr. 2010.

[82] A. X. Liu, E. Torng, and C. Meiners, "Firewall Compressor: An Algorithm for Minimizing Firewall Policies," in *IEEE Infocom*, Phoenix, Arizona, April 2008.

[83] 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.

[84] R. McGeer and P. Yalagandula, "Minimizing Rulesets for TCAM Implementation," in *Proceedings of the IEEE Infocom*. Rio de Janeiro, Brazil, Apr. 2009, pp. 1314–1322.

[85] L. Luo, G. Xie, S. Uhlig, L. Mathy, K. Salamatian, and Y. Xie, "Towards TCAM-based Scalable Virtual Routers," in *ACM Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, 2012, pp. 73–84.

[86] Y. E. Oktian, S.-G. Lee, H.-J. Lee, and J.-H. Lam, "Distributed SDN Controller System: A Survey on Design Choice," *Computer Networks*, vol. 121, pp. 100–111, 2017.

[87] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "OpenFlow: Meeting Carrier-Grade Recovery Requirements," *Computer Communications*, vol. 36, no. 6, pp. 656–665, 2013.

[88] J. Kempf, E. Bellagamba, A. Kern, D. Jocha, A. Takàcs, and P. Sköldström, "Scalable Fault Management for OpenFlow," in *Proceedings of the IEEE International Conference on Communications (ICC)*, 2012, pp. 6606–6610.

[89] N. L. M. van Adrichem, B. J. van Asten, and F. A. Kuipers, "Fast Recovery in Software-Defined Networks," in *Proceedings of the European Workshop on Software Defined Networks (EWSDN)*, 2014.

[90] 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 (NetSoft)*, June 2016, pp. 296–302.

[91] 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.

[92] 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 (NFV-SDN)*, Nov 2016, pp. 179–185.

[93] 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 (ISCC)*, 2016.

[94] R. Couto, M. Campista, and L. Costa, "A Reliability Analysis of Datacenter Topologies," in *IEEE Globecom*, Dec. 2012, pp. 1890–1895.

[95] S. Cho, T. Elhourani, and S. Ramasubramanian, "Independent Directed Acyclic Graphs for Resilient Multipath Routing," *IEEE/ACM Transactions on Networking*, vol. 20, no. 1, pp. 153 –162, Feb. 2012.

[96] G. Jayavelu, S. Ramasubramanian, and O. Younis, "Maintaining Colored Trees for Disjoint Multipath Routing under Node Failures," *IEEE/ACM Transactions on Networking*, vol. 17, no. 1, pp. 346 – 359, 2009.

[97] W. Zhang, G. Xue, J. Tang, and K. Thulasiraman, "Faster Algorithms for Construction of Recovery Trees Enhancing QoP and QoS," *IEEE/ACM Transactions on Networking*, vol. 16, no. 3, pp. 642–655, Jun. 2008.

[98] N. Taft-Plotkin, B. Bellur, and R. Ogier, "Quality-of-Service Routing using Maximally Disjoint Paths," in *International Workshop on Quality of Service*, 1999, pp. 119–128.

[99] Y. Guo, F. A. Kuipers, and P. V. Mieghem, "Link-disjoint Paths for Reliable QoS Routing," *International Journal of Communication Systems*, vol. 16, no. 9, pp. 779–798, 2003.

[100] G. S. Envedi, A. Csaszar, A. Atlas, C. Bowers, and A. Gopalan, "An Algorithm for Computing IP/LDP Fast Reroute Using Maximally Redundant Trees (MRT-FRR)," RFC 7811, Jun. 2016.

[101] H. Kirrmann, M. Hansson, and P. Muri, "IEC 62439 PRP: Bumpless Recovery for Highly Available, Hard Real-Time Industrial Networks," in *2007 IEEE Conference on Emerging Technologies and Factory Automation (EFTA 2007)*, Sept 2007, pp. 1396–1399.

[102] H. Heine and O. Kleineberg, "The High-Availability Seamless Redundancy Protocol (HSR): Robust Fault-Tolerant Networking and Loop Prevention Through Duplicate Discard," in *IEEE International Workshop Factory Communication Systems (WFCS)*, May 2012, pp. 213–222.

[103] S. Paul, K. K. Sabnani, J. C. H. Lin, and S. Bhattacharyya, "Reliable Multicast Transport Protocol (RMTP)," *IEEE Journal on Selected Areas in Communications*, vol. 15, no. 3, pp. 407–421, Apr 1997.

[104] B. N. Levine, D. B. Lavo, and J. J. Garcia-Luna-Aceves, "The Case for Reliable Concurrent Multicasting Using Shared ACK Trees," in *ACM International Conference on Multimedia*, 1996, pp. 365–376.

[105] Y. Ofek and B. Yener, "Reliable Concurrent Multicast From Bursty Sources," in *IEEE Infocom*, vol. 3, Mar 1996, pp. 1433–1441.

[106] D. Frost, S. Bryant, M. Bocci, and L. Berger, "A Framework for Point-to-Multipoint MPLS in Transport Networks," RFC 7167, Oct. 2015.

[107] S. Y. (Ed.), "RFC4655: Signaling Requirements for Point-to-Multipoint Traffic-Engineered MPLS Label Switched Paths (LSPs)," Apr. 2006.

[108] R. Fabregat, Y. Donoso, B. Baran, F. Solano, and J. L. Marzo, "Multi-objective Optimization Scheme for Multicast Flows: A Survey, a Model and a MOEA Solution," in *IFIP/ACM Latin American Conference on Networking*, 2005, pp. 73–86.

[109] Y. Nakagawa, K. Hyoudou, and T. Shimizu, "A Management Method of IP Multicast in Overlay Networks Using Openflow," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, ser. HotSDN '12. New York, NY, USA: ACM, 2012, pp. 91–96.

[110] X. Li and M. J. Freedman, "Scaling IP Multicast on Datacenter Topologies," in *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '13. New York, NY, USA: ACM, 2013, pp. 61–72.

[111] J. Ruckert, J. Blendin, R. Hark, and D. Hausheer, "DYNSDM: Dynamic and Flexible Software-Defined Multicast for ISP Environments," in *International Conference on Network and Services Management (CNSM)*, Nov 2015, pp. 117–125.

[112] A. Giorgetti, A. Sgambelluri, F. Paolucci, P. Castoldi, and F. Cugini, "First Demonstration of SDN-based Bit Index Explicit Replication (BIER) Multicasting," in *2017 European Conference on Networks and Communications (EuCNC)*, Jun. 2017, pp. 1–6.

[113] Z. Al-Saeed, I. Ahmad, and I. Hussain, "Multicasting in Software Defined Networks: A Comprehensive Survey," *Journal of Network and Computer Applications*, vol. 104, pp. 61 – 77, 2018.

[114] "Route views project page," Mar. 2014. [Online]. Available: http://www.routeviews.org/

[115] B. Agrawal and T. Sherwood, "Modeling TCAM POWER for Next Generation Network Devices," in *IEEE International Symposium on Performance Analysis of Systems and Software*, Mar. 2006, pp. 120–129.

[116] R. L. Rudell, "Multiple-Valued Logic Minimization for PLA Synthesis," EECS Department, University of California, Berkeley, Tech. Rep. UCB/ERL M86/65, 1986.

[117] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe, "Design and Implementation of a Routing Control Platform," in *Proceedings of the USENIX Syposium on Networked Systems Design & Implementation (NSDI)*, 2005, pp. 15–28.

[118] C. E. Rothenberg, M. R. Nascimento, M. R. Salvador, C. N. A. Corrêa, S. C. de Lucena, and R. Raszuk, "Revisiting Routing Control Platforms with the Eyes and Muscles of Software-Defined Networking," in *Proceedings of the ACM Workshop on Hot Topics in Software Defined Networks (HotSDN)*, 2012, pp. 13–18.

[119] R. Bennesby, P. Fonseca, E. Mota, and A. Passito, "An Inter-AS Routing Component for Software-Defined Networks," in *Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2012, pp. 138–145.

[120] A. Gupta, L. Vanbever, M. Shahbaz, S. P. Donovan, B. Schlinker, N. Feamster, J. Rexford, S. Shenker, R. Clark, and E. Katz-Bassett, "SDX: A Software Defined Internet Exchange," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 579–580, Aug. 2014.

[121] J. Rexford, J. Wang, Z. Xiao, and Y. Zhang, "BGP Routing Stability of Popular Destinations," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurment*, 2002, pp. 197–202.

[122] M. Shand and S. Bryant, "IP Fast Reroute Framework," RFC 5714, Jan. 2010. [Online]. Available: https://rfc-editor.org/rfc/rfc5714.txt

[123] G. S. Envedi, A. Atlas, and C. Bowers, "An Architecture for IP/LDP Fast Reroute Using Maximally Redundant Trees (MRT-FRR)," RFC 7812, Jun. 2016.

[124] G. Iannaccone, C.-N. Chuah, R. Mortier, S. Bhattacharyya, and C. Diot, "Analysis of Link Failures in an IP Backbone," in *ACM SIGCOMM Internet Measurement Workshop*, Marseille, France, Nov. 2002, pp. 237 – 242.

[125] T. Schwabe and C. G. Gruber, "Traffic Variations Caused by Inter-domain Rerouting," in *International Workshop on the Design of Reliable Communication Networks (DRCN)*, Ischia Island, Italy, Oct. 2005.

[126] G. Enyedi, G. Retvaria, and A. Csaszar, "On Finding Maximally Redundant Trees in Strictly Linear Time," in *IEEE Symposium on Computers and Communications (ISCC)*, July 2009, pp. 206–211.

[127] 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.

[128] S. Li, Y. Shao, S. Ma, N. Xue, S. Li, D. Hu, and Z. Zhu, "Flexible Traffic Engineering: When OpenFlow Meets Multi-Protocol IP-Forwarding," *IEEE Communications Letters*, vol. 18, no. 10, pp. 1699–1702, Oct. 2014.

[129] J. C. Mogul and P. Congdon, "Hey, You Darned Counters!: Get off my ASIC!" in *Proceedings of the ACM Workshop on Hot Topics in Software Defined Networks (HotSDN)*, 2012, pp. 25–30.

[130] 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.

[131] 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.

[132] 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.

[133] 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.

[134] 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.

[135] A. Medina, N. Taft, K. Salamatian, S. Bhattacharyya, and C. Diot, "Traffic Matrix Estimation: Existing Techniques and New Directions," in *ACM SIGCOMM*, Pittsburgh, USA, Aug. 2002.

[136] 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.

[137] J. W. Suurballe and R. E. Tarjan, "A Quick Method for Finding Shortest Pairs of Disjoint Paths," *Networks Magazine*, vol. 14, pp. 325–336, 1984.

[138] Population Reference Bureau, "2012 World Population Data Sheet," Oct. 2014. [Online]. Available: http://www.prb.org/pdf12/2012-population-data-sheet_eng.pdf

[139] United States Census Bureau, "Population Estimates," Oct. 2014. [Online]. Available: http://www.census.gov/popest/data/cities/totals/2013/SUB-EST2013.html

[140] G. Shepherd, A. Dolganow, and A. Gulko, "Bit Indexed Explicit Replication (BIER) Problem Statement," Apr. 2016.

[141] N. Kumar, R. Asati, M. Chen, X. Xu, A. Dolganow, T. Przygienda, A. Gulko, D. Robinson, V. Arya, and C. Bestler, "BIER Use Cases," Jan. 2018.

[142] I. Wijnands, E. C. Rosen, A. Dolganow, T. Przygienda, and S. Aldrin, "Multicast Using Bit Index Explicit Replication (BIER)," RFC 8279, Nov. 2017.

[143] A. Karan, C. Filsfils, I. Wijnands, and B. Decraene, "Multicast only Fast Re-Route," https://tools.ietf.org/html/rfc7431, Aug. 2015.

[144] A. Atlas, R. Kebler, I. Wijnands, A. Csaszar, and G. Enyedi, "An Architecture for Multicast Protection Using Maximally Redundant Trees," http://tools.ietf.org/id/draft-atlas-rtgwg-mrt-mc-arch, Mar. 2012.

[145] C. Filsfils, S. Previdi, B. Decraene, S. Litkowski, and R. Shakir, "Segment Routing Architecture," Internet Engineering Task Force, Internet-Draft draft-ietf-spring-segment-routing-10, Nov. 2016.

[146] E. C. Rosen, A. Viswanathan, and R. Callon, "RFC3031: Multiprotocol Label Switching Architecture," Jan. 2001.

[147] I. Wijnands, E. C. Rosen, A. Dolganow, J. Tantsura, S. Aldrin, and I. Meilik, "Encapsulation for Bit Index Explicit Replication (BIER) in MPLS and Non-MPLS Networks," RFC 8296, Jan. 2018.

[148] "Behavioral model repository," Jan 2017. [Online]. Available: https://github.com/p4lang/behavioral-model

[149] "Tools for multicast testing (msend and mreceive)," Jan 2017. [Online]. Available: https://github.com/troglobit/mtools

[150] "The P4 Language Specification Version 1.0.4," May 2017. [Online]. Available: https://p4.org/p4-spec/p4-14/v1.0.4/tex/p4.pdf