P4-Based Implementation of BIER and BIER-FRR for Scalable and Resilient Multicast

Daniel Merling*, Steffen Lindner, Michael Menth

University of Tuebingen, Department of Computer Science, Chair of Communication Networks, Tuebingen, Germany

Abstract

Bit Indexed Explicit Replication (BIER) is a novel IP multicast (IPMC) forwarding paradigm proposed by the IETF. It offers a transport layer for other IPMC traffic, keeps core routers unaware of IPMC groups, and utilizes a routing underlay, e.g., an IP network, for its forwarding decisions. With BIER, core networks do not require dynamic signaling and support a large number of IPMC groups with large churn rates. The contribution of this work is threefold. First, we propose a novel fast reroute (FRR) mechanism for BIER (BIER-FRR) so that IPMC traffic can be rerouted as soon as the routing underlay is able to carry traffic again after a failure. In particular, BIER-FRR enables BIER to profit from FRR mechanisms in the routing underlay. Second, we describe a prototype for BIER and BIER-FRR within an IP network with IP fast reroute (IP-FRR). It is based on the programmable data plane technology P4. Third, we propose a controller hierarchy with local controllers for local tasks, in particular to enable IP-FRR and BIER-FRR. The prototype demonstrates that BIER-FRR reduces the protection time for BIER traffic to the protection time for unicast traffic in the routing underlay.

Keywords: Software-Defined Networking, P4, Bit Index Explicit Replication,

*Corresponding author

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

Email addresses: `daniel.merling@uni-tuebingen.de` (Daniel Merling), `steffen.lindner@uni-tuebingen.de` (Steffen Lindner), `menth@uni-tuebingen.de` (Michael Menth)

Multicast, Resilience, Scalability

1. Introduction

IP multicast (IPMC) is leveraged for many services like IPTV, multicast VPN, or the distribution of financial or broadcast data. It efficiently forwards one-to-many traffic on tree-like structures to all desired destinations by sending at most one packet copy per link in the distribution tree. IPMC is organized into sets of receivers, so-called IPMC groups. Hosts subscribe to IPMC groups to receive the traffic which is addressed to that group. Traditional IPMC methods require per-IPMC-group state within core routers to forward the packets to the right next-hops (NHs). This raises three scalability issues. First, the number of IPMC groups may be large which require lots of space in forwarding tables. Second, core routers are involved in the establishment, removal, and in the change of an IPMC group. This requires significant signaling in the core network every time subscribers change because many nodes possibly need to update their forwarding information base, which imposes heavy load on core when churn rates are large. Third, when the topology changes or in case of a failure, the forwarding of any IPMC group possibly requires fast update, which is demanding in a critical network situation. IPMC features are available in most off-the-shelf hardware, but the features are turned off by administrators due to complexity and limited scalability.

The Internet Engineering Task Force (IETF) developed Bit Index Explicit Replication (BIER) [1, 2] as a solution to those problems. BIER features a domain concept. Only ingress and egress routers of a BIER domain participate in signalling. They encapsulate IPMC packets with a BIER header containing a bit string that indicates the receivers of the IPMC group within the BIER domain. Based on that bit string the packets are forwarded through the BIER domain without requiring per-IPMC-group state in core routers.

BIER leverages the so-called bit indexed forwarding table (BIFT) for forwarding decisions. Its entries are derived from paths induced by the interior gateway protocol (IGP) which is used for unicast routing. In the following we refer to that routing protocol with the term 'routing underlay'. Therefore, BIER traffic follows the same paths as the unicast traffic carried by the routing underlay. So far, BIER lacks any protection capabilities. In case of a link or node failure, the BIFT entries need to be changed so that BIER traffic is carried around failed elements towards receivers. However, the BIFTs can be updated only after the routing underlay has updated its forwarding information base and based on the new paths, BIER forwarding entries are recomputed. This takes a significant amount of time. In the meantime, packets are dropped. When a multicast packet is dropped, all downstream subscribers cannot receive the packet. Regular IP forwarding is affected as well by failures, but for unicast traffic, fast reroute (FRR) [3] mechanisms have been proposed to reroute affected packets on backup paths until the primary forwarding entries are updated. IP-FRR leverages pre-computed backup actions for fast recovery in case of a failure without the need for signaling. However, IP-FRR is not a suitable protection method for multicast traffic because it does not consider the tree-like forwarding structures along which IPMC packets are distributed.

In this work, we introduce BIER-FRR. It has two different operation modes to protect either only against link failures or also against node failures. We recently proposed this mechanism in the BIER working group of the IETF [4]. BIER has been suggested as a novel transport mechanism for IPMC. However, it cannot be configured yet on standard hardware. New, programmable data plane technologies allow the definition of new packet headers and forwarding behavior, and offer themselves for the implementation of prototypes. In [5], we presented an early version of a P4-based prototype for BIER. It was based on the P_{14} specification of P4 [6] and required a few workarounds because at that time some P4 features were not available on our target, the software switch BMv2. Moreover, there was no protection method available for BIER. We now provide the description of a completely reworked prototype on the base of the P_{16} specification of P4 [7]. The new prototype implements IP forwarding, a simple form of IP-FRR, BIER forwarding, and BIER-FRR. It comprises a controller hierarchy with local controllers that enables FRR techniques with P4. We argue that local controllers are needed for protection and helpful for local tasks in general. The evaluation of the prototype shows that BIER traffic is not longer affected by network failures than unicast traffic when BIER-FRR is enabled. Thus, the contribution of this paper is threefold: (1) a concept for BIER-FRR, (2) an implementation of BIER and BIER-FRR with P4, and (3) a controller hierarchy with local controllers to support FRR techniques in P4. Finally, the P4-based prototype demonstrates the usefulness of BIER-FRR. The source code of our prototype with the fully working data and control plane is publicly available on GitHub.

The remainder of this paper is structured as follows. Section 2 reviews basics of multicast. Section 3 contains fundamentals about IP-FRR, explains why it is insufficient to protect multicast traffic, and examines related work. Section 4 discusses related work for both legacy- and SDN-based multicast. Section 5 gives a primer on BIER. Section 6 explains the resilience problem of BIER and introduces BIER-FRR. In Section 7 we summarize necessary basics of P4 needed for the understanding of the BIER prototype. Section 8 describes the P4-based prototype implementation of IP, IP-FRR, BIER, and BIER-FRR. The prototype is used to demonstrate the usefulness of BIER-FRR in Section 9. Finally, Section 10 summarizes this paper and discusses further research issues.

2. Technological Background for IP Multicast

This section gives a primer on IP multicast (IPMC) for readers that are not familiar with IPMC. IPMC supports one or more sources to efficiently communicate with a set of receivers. The set of receivers is called an IPMC group and is identified by an IP address from the Class D address space (224.0.0.0 – 239.255.255.255). Devices join or leave an IPMC group leveraging the Internet Group Management Protocol (IGMP) [8].

IPMC packets are delivered over group-specific distribution trees which are computed and maintained by IPMC-capable routers. In the simplest form, source-specific multicast trees based on the shortest path principle are computed and installed in the routers. The notation (S, G) identifies such a shortest path tree for the source S and the group G.

IPMC also supports the use of shared trees that can be used by multiple senders to send traffic to a multicast group. The shared tree has a single root node called rendezvous point (RP). The sources send the multicast traffic to the RP which then distributes the traffic over a shared tree. In the literature, shared trees are denoted by $(*, G)$.

Protocol-independent multicast (PIM) leverages unicast routing information to perform multicast forwarding. PIM cooperates with various unicast routing protocols such as OSPF or BGP and supports both source-specific and shared multicast distribution trees.

Pragmatic General Multicast (PGM) [9] reduces packet loss for multicast traffic. It enables receivers to detect lost or out-of-order packets and supports retransmission requests similar to TCP.

3. IP Fast Reroute

In this section we give a primer on IP fast reroute (IP-FRR). First, we explain fundamentals of IP-FRR and describe Loop-Free Alternates. Then, we discuss related work.

3.1. Fundamentals of IP-FRR

When a link or a node fails, devices may not be able to forward packets to their NHs. As soon as a failure is detected in an IP network, the changed topology is signaled through the network, new working paths are calculated, and the forwarding tables of all devices are consistently updated. This process is called reconvergence and may take up to several seconds. In the meantime, packets are dropped in case of wrong or missing forwarding entries. IP-FRR [3] protects IP unicast traffic against the failure of single links and nodes while reconvergence is ongoing. It is based on pre-computed backup actions to quickly reroute affected packets. Figure 1 shows an example for Loop-Free Alternates (LFAs) [10] which is the most popular IP-FRR mechanism. When a node's

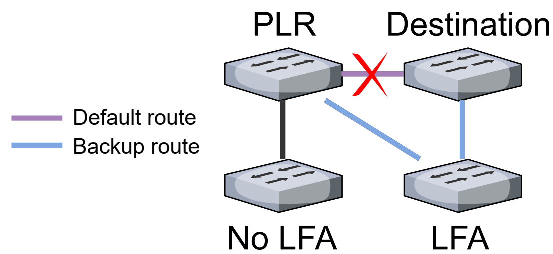


Figure 1: A PLR reroutes a packet to a backup path when the NH on the primary path is unreachable.

(primary) NH becomes unreachable, the node detects that failure after some time and reroutes the traffic locally over a backup path. Therefore, the node is also called point of local repair (PLR). To reroute packets in a timely manner, nodes store a backup NH in addition to the primary NH for each destination. When the PLR detects that the primary NH is unrechable, e.g., by loss-of-light detection, loss-of-carrier detection, a bidirectional forwarding detection[1] (BFD) [11], or any other suitable mechanism, it forwards the packet to its backup NH instead. That backup NH is called Loop-Free Alternate (LFA) and it has to be chosen such that rerouted traffic does not loop. However, some destinations remain unprotected because there is not always an alternative hop that has a shortest path towards the destination which avoids the failed element. The set of protected destinations is also called coverage. The limited coverage of LFAs has been evaluated in various studies [12, 13].

3.2. Related Work

The two surveys [14, 15] give an overview of several IP-FRR mechanisms. We discuss only some of the papers. Equal-cost multi-path (ECMP) can be used

[1]When a BFD is established between two nodes, they periodically exchanges keep-alive notifications.

as a very basic FRR mechanism. When a PLR has at least two paths with equal cost towards a destination, it quickly deviates traffic to the other path when the primary NH is unreachable. However, this works only if two equal-cost paths are available under normal conditions, which is mostly not the case. Not-via addresses [16, 13] tunnel packets to the downstream next-next-hop (NNH) if the NH is unreachable. To that end, the NNH is assigned a unique address and an explicit backup path is constructed which does not include the failed component. Loop-Free Alternates (LFAs) [10] forward packets to alternative NHs if the primary NH is unreachable. Those alternative NHs have to be chosen in a way that they have a working shortest path to the destination that avoids rerouting loops. As such alternative neighbors do not exist for all PLRs and destinations, the LFA mechanism cannot protect against all single link and node failure. Remote LFAs [17] (rLFAs) extend the protection capabilities of LFAs by sending affected packets through shortest path tunnels to nodes that still reach the destination on a working shortest path. rLFAs protect against any single link failure in unit link cost networks. However, they achieve only partial coverage in case of node failures or non-unit link costs. An analysis can be found in [12].

4. Related Work

We review work in the context of SDN-based multicast. Most traditional multicast approaches were implemented with OpenFlow. Some works considered protection mechanisms. A few studies improve the efficiency of multicast forwarding with SDN. Only a single work implements BIER without protection using OpenFlow, but the implementation itself requires dynamic forwarding state, which runs contrary to the intention of BIER.

4.1. Multicast Implementations with OpenFlow

The surveys [18, 19] provide an extensive overview of multicast implementations for SDN. They discuss the history of traditional multicast and present multiple aspects for SDN-based multicast, e.g., multicast tree planning and management, multicast routing and reliability, etc. In the following we briefly discuss some exemplary works that implement multicast for SDN. More details can be found in the surveys or the original papers.

Most related works with regard to SDN-based multicast implement explicit flow-specific multicast trees. Most authors propose to compute traffic-engineered multicast trees in the controller using advanced algorithms and leverage SDN as tool to implement the multicast path layout. The following works provide implementations in OpenFlow to show the feasibility of their approaches. Dynamic Software-Defined Multicast (DynSDM) [20, 21] leverages multiple trees to load-balance multicast traffic and efficiently handle group subscription changes. Modified Steiner tree problems are considered in [22, 23] to minimize the total cost of edges and the number of branch nodes, or to additionally minimize the source-destination delay [24]. In [25], the authors compute and install traffic-engineered shared multicast trees using OpenFlow. In [26] and [27], traffic-engineered Steiner trees are computed which minimize the number of edges of the tree and provide optimized locations for multicast sources in the network. The Avalanche Routing Algorithm (AvRA) [28] considers topological properties of data center networks to optimize utilization of network links. Dual-Structure Multicast (DuSM) [29] improves scalability and link utilization for SDN-based data center networks by deploying different forwarding approaches for high-bandwidth and low-bandwidth flows. In [30], Steiner trees are leveraged to compute a multicast path layout including certain recovery nodes which are used for reliable multicast transmission such as PGM. In [31], the authors evaluate different node-redundant multicast tree algorithms in an SDN context. They evaluate the number of forwarding rules required for each mechanism and study the effects of node failures. The authors of [32] reduce the number of forwarding entries in OpenFlow switches for multicast. They propose to use address translation from the multicast address to the receiver's unicast address on the last branching node of the multicast tree. This allows to omit multicast-specific forwarding entries in leaf switches.

4.2. Multicast Protection with OpenFlow

Kotani et al. [33] suggest to utilize primary and backup multicast trees for SDN networks. Multicast packets carry an ID to identify the distribution tree over which they are forwarded. In case of a failure, the controller chooses an appropriate backup multicast tree and reconfigures senders accordingly. This mechanism differs in two ways from BIER-FRR. First, the controller has to be notified, which is not suitable for fast recovery. Second, it requires significant signalling effort in response to a failure.

The authors of [34] propose a FRR method for multicast in OpenFlow networks. Multicast traffic is forwarded along a default distribution tree. If a downstream neighbor is unreachable, traffic is switched to a backup distribution tree that contains all downstream nodes of the unreachable default subtree. The backup distribution tree must not contain the unreachable neighbor as forwarding node. VLAN tags are used to indicate the trees over which multicast packets should be sent. This mechanism differs from BIER-FRR in a way that it requires a significant amount of additional dynamic forwarding state to configure the backup trees.

4.3. Improved Multicast Forwarding for SDN Switches

Some contributions improve the efficiency of devices to enable hardware-based multicast forwarding. The work in [35] organizes forwarding entries of a switch based on prime numbers and the Chinese remainder theorem. It reduces the internal forwarding state and allows for more efficient hardware implementations. Reed et al. provide stateless multicast switching in SDN-based systems using Bloom filters in [36] and implement their approach for TCAM-based switches. The authors compare their approach with existing layer-2 forwarding and show that their method leads to significantly lower TCAM utilization.

4.4. SDN Support for BIER

The authors of [37, 38] implement two SDN-based multicast approaches using (1) explicit multicast tree forwarding and (2) BIER forwarding in OpenFlow.

They realize explicit multicast trees with OpenFlow group tables. To support BIER, they leverage MPLS headers to encode the BIER bit string, which limits the implementation to bit strings with a length of 20 bits, and therefore a maximum of 20 receivers. Rules with exact matches for bit strings are installed in the OpenFlow forwarding tables. When a packet with a BIER header arrives at a switch and a rule for its bit string is available, the packet can be immediately transmitted over the indicated interfaces. Otherwise, a local BIER agent running on the switch and maintaining the BIFT is queried. The local BIER agent installs a new flow entry for the specific bit string in the OpenFlow forwarding table. Thus, this approach requires bit string-specific state instead of IPMC group specific state. Furthermore, it is not likely to work well with quickly changing multicast groups as most subscription changes require configuration changes in the forwarding table of the switch. BIER with support for traffic engineering (TE) has been proposed in [39]. It uses the same header format but features different forwarding semantics and is not compatible with normal BIER. In [40] we have proposed and evaluated several FRR algorithms for BIER-TE and implemented them in a P4-based prototype [5].

5. Bit Index Explicit Replication (BIER)

First, we give an overview of BIER [2]. Afterwards, we present the Bit Index Forwarding Table (BIFT), which is the forwarding table for BIER. Then, we describe the BIER forwarding procedure.

5.1. Overview

We introduce essential nomenclature for BIER, the layered BIER architecture, the BIER header, and the BIER forwarding principle.

5.1.1. BIER Domain

BIER leverages a domain concept to transport IPMC traffic in a scalable manner, which is illustrated in Figure 2. Bit-Forwarding Routers (BFRs) forward BIER multicast traffic within the BIER domain. Inbound multicast traffic enters the domain through Bit-Forwarding Ingress Routers (BFIRs) and leaves it through Bit-Forwarding Egress Routers (BFERs). Border routers usually implement both BFIR and BFER functionality. When a BFIR receives an inbound IPMC packet, it pushes a BIER header onto the IPMC packet which indicates all BFERs that should receive a packet copy. BFRs utilize the information in the BIER header to forward BIER packets to all desired destinations along the paths induced by the interior gateway protocol (IGP). Thereby, packets are replicated if needed. Finally, the BFERs remove the BIER header before forwarding IPMC packets outside the domain.

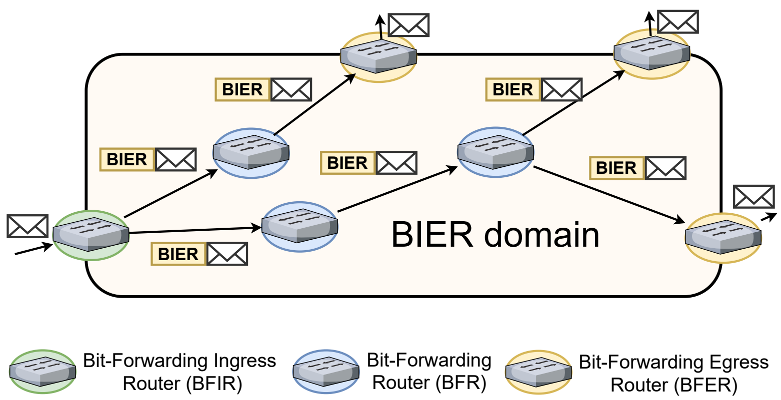


Figure 2: IPMC traffic enters a BIER domain through BFIRs which equip it with a BIER header. BFRs forwarded the traffic based on the BIER header within the domain on paths induced by the IGP. BFERs remove the BIER header when the traffic leaves the domain.

5.1.2. *A Layered BIER Architecture*

The BIER architecture can be subdivided into three layers: the IPMC layer, the BIER layer, and the routing underlay which is the forwarding mechanism for unicast traffic. In IP networks, the latter corresponds to the interior gateway protocol (IGP). Figure 3 shows the relation among the layers.

The IPMC layer requests multicast delivery for IPMC packets to a set of receivers in a BIER domain that depend on IPMC subscriptions. That is, when an inbound IMPC packet arrives at a BFIR, the BFIR equips the IPMC packet with an appropriate BIER header indicating all desired destinations. The BIER

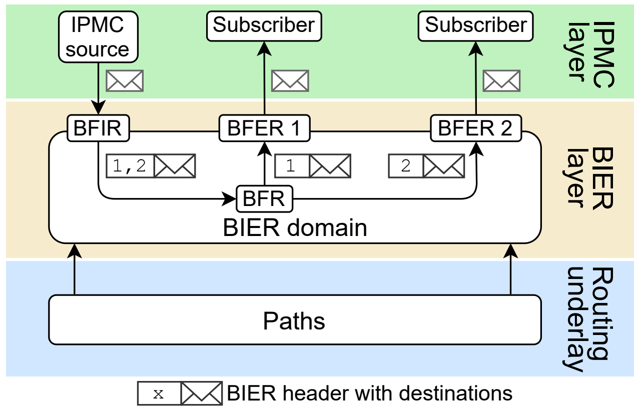


Figure 3: Layered BIER architecture with IPMC layer, BIER layer, and the routing underlay.

layer forwards these packets through the BIER domain to all receivers indicated in the BIER header, thereby implementing a stateless point-to-multipoint tunnel für IPMC. The BIER layer leverages the forwarding information of the routing underlay to populate the forwarding tables of the BFRs. As a result, BIER traffic to a specific BFER follows the same path as unicast traffic towards that BFER. If two BFR are connected on Layer 2, the BIER traffic is directly forwarded; otherwise, the BFR neighbor is reachable only over the routing underlay so that the BIER traffic is encapsulated and forwarded over the routing underlay. When a BIER packet reaches a BFER that should receive a packet copy, the BFER removes the BIER header and passed the IPMP packet to the IPMC layer for further forwarding.

5.1.3. *BIER Header and Forwarding Principle*

The BIER header contains a bit string to identify BFERs. For brevity, we talk in the following about the BitString of a packet to refer to the bit string in the BIER header of that packet. The BitString is of a specific lenght. Each bit in the BitString corresponds to one specific BFER. The bits are assigned to BFERs starting with the least significant bit. BIER devices must support a BIER header of 256 bits. As this may not suffice to assign bits to all BFERs in large networks, the standard [2] defines subdomains to cope with that problem.

This is a technical detail that we do not consider any further and our proposed solution can be easily adapted to subdomains.

When a BFIR receives an IPMC packet, it pushes a BIER header to the IPMC packet, determines the set of BFERs that must receive the traffic of the respective IMPC group, and activates in the BitString the bits corresponding to these BFERs. Packets are forwarded based on the information in their BIER header. A BFR sends a packet to any of its interfaces if at least one BFER indicated in the BIER header is reached in the routing underlay over this specific interface. To avoid duplicates, only those bits are kept in the BitString whose BFERs can be reached over the specific interface.

Figure 4 illustrates the BIER forwarding principle. It shows a small BIER domain with four nodes, each of them being BFR, BFIR, and BFER. Hosts are attached to all BIER nodes and participate in a single multicast group. Host 1 sends an IPMC packet to all other hosts. The figure visualizes how the BitString changes along the forwarding tree whose paths are inherited from the routing underlay.

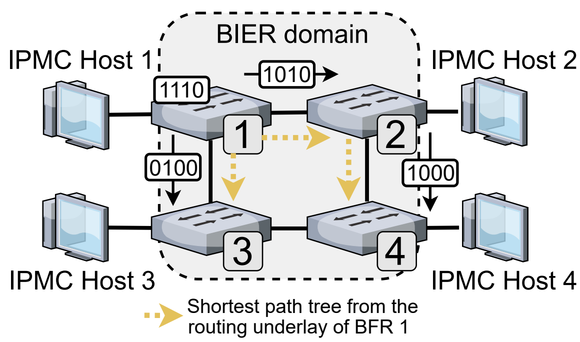


Figure 4: An IPMC packet is forwarded from Host 1 to all other hosts via a BIER domain. Within the domain, BIER packets are forwarded based on the BitString.

The information of the BIER forwarding tables depends only on the routing underlay. In Section 5.2 we explain the structure of the table and how its entries are calculated. In contrast to traditional IPMC forwarding, BIER forwarding does not require knowledge about IPMC groups. This has several advantages.

BFRs do not neet to keep state per IPMC group. This frees core nodes of a BIER domain from signalling and state processing per IPMC group when subscriptions or routes change, e.g., in case of failures. This makes BIER forwarding in core nodes more robust and scalable than traditional IPMC forwarding. BFIRs still participate in IPMC signaling to keep track of group changes in order to adapt the BIER header for each IPMC group. BFERs forward outbound IPMC traffic in a traditional way.

5.2. *Bit Index Forwarding Table*

In this section we describe the Bit Index Forwarding Table (BIFT) which is the forwarding table of BFRs. We explain its structure and the computation of its entries.

First, we define BFR neighbors (BFR-NBRs) before we introduce the structure of the BIFT. The BFR-NBRs of a BFR A are those BFRs, that are adjacent to A according to the paths of the routing underlay.

Each BFR maintains a Bit Index Forwarding Table (BIFT) to determine the NH, i.e., BFR-NBR, when forwarding a packet. Table 1 shows the structure of the BIFT. For each BFER, the BIFT contains one entry which consists of a forwarding bitmask (F-BM) and the BFR-NBR to which the packet should be sent. The F-BM is used in the forwarding process to clear bits in a packet's BitString before transmission. The BFR-NBR for a BFER is derived as the BFR-NBR on the path from the considered BFR to the BFER in the routing underlay. The F-BM for a BFER is composed as a bit string where all bits are activated that belong to BFERs with the same BFR-NBR. As a result, all BIFT entries with the same BFR-NBRs also have the same F-BM.

BFER	F-BM	BFR-NBR

Table 1: Header of the BIFT.

Table 2 illustrates the BIFT of BFR 1 in the example given in Figure 4.

BFER	F-BM	BFR-NBR
1	-	-
2	1010	2
3	0100	3
4	1010	2

Table 2: BIFT of BFR 1.

5.3. BIER Forwarding

In this paragraph we describe BIER forwarding. First, we explain the procedure how BIER processes packets. Then, we show a forwarding example. Finally, we illustrate the BIER header stack.

5.3.1. BIER Forwarding Procedure

BFRs process BIER packets in a loop. When a BFR receives a BIER packet, it determines the position of the least-significant activated bit in the BitString. The position of that bit corresponds to a BFER which is processed in this specific iteration of the loop. The BFR looks up that BFER in the BIFT, which results in a BFR-NBR and a F-BM. Then, a copy of the BIER packet is created for transmission to that BFR-NBR. Before transmission, all bits are cleared in the BitString of the packet copy that are not reachable through the same BFR-NBR. This is achieved by an AND-operation of the BitString and the F-BM. We denote this action as "applying the F-BM to the BitString". Then, the packet copy is forwarded to the indicated BFR-NBR. All BFERs in the IPMC subtree of that BFR-NBR eventually receive a copy of this sent packet if their bit is activated in the BitString of the packet copy. Thus, all BFERs of this IMPC subtree can be considered as processed. Therefore, their bits are removed from the BitString of the remaining BIER packet. To that end, the BFR applies the complement of the F-BM to the BitString of the remaining BIER packet. This ensures that packets are delivered only once to intended receivers. If the BitString in the remaining BIER packet still contains activated bits, the loop restarts; otherwise the processing loop stops.

When the BFER that is currently processed corresponds to the BFR itself, the F-BM and BFR-NBR of its BIFT entry are empty. Then, a copy of the BIER packet is created, the BIER header is removed, and the packet is passed to the IPMC layer within the BFR. Afterwards, the processed bit is cleared in the BitString of the remaining BIER packet, and the loop restarts if the BitString contains activated bits; otherwise the loop stops.

5.3.2. BIER Forwarding Example

We assume that BFIR 1 in Figure 4 receives an IPMC packet from IPMC Host 1 that should be sent to the IPMC Hosts 2, 3, and 4. Therefore, it adds a BIER header with the BitString `1110` to the IPMC packet and processes it. The least-significant activated bit corresponds to BFR 2. BFR 1 looks up the activated bit in its BIFT which is shown in Table 2. Then, it creates a packet copy and applies the F-BM to the BitString of that copy. This sets the BitString to `1010`. Then, the packet copy is forwarded to BFR 2. Aftwards, BFR 1 clears the activated bits of the F-BM from the BitString of the remaining original BIER packet. This leaves a packet with the BitString `0100`. BFR 1 processes the next activated bit, i.e. the bit for BFER 3. A packet copy is created, and the F-BM is applied which leaves the BitString `0100` in the packet copy. Then it is forwarded to BFR 3.

BFR 2 process the packet in the same way. As a result, it forwards one packet copy with the BitString `1000` to BFR 4. Additionally, it sends an IPMC packet without BIER header to its connected host. BFR 3 and 4 do the same when they receive their respective BIER packet.

5.3.3. BIER Header Stack

Without loss of generality, we assume in the following that the routing underlay is IP. Furthermore, we neglect the role of Layer 2 to facilitate readability.

Each BIER device is also an IP device. However, not every IP device is a BIER device. In Figure 5, the “Pure IP-node” is an IP node without BIER functionality. It belongs to the IP topology but not to the BIER topology. This influences the header stack of forwarded BIER packets. BFR 1, 2 and 3 are

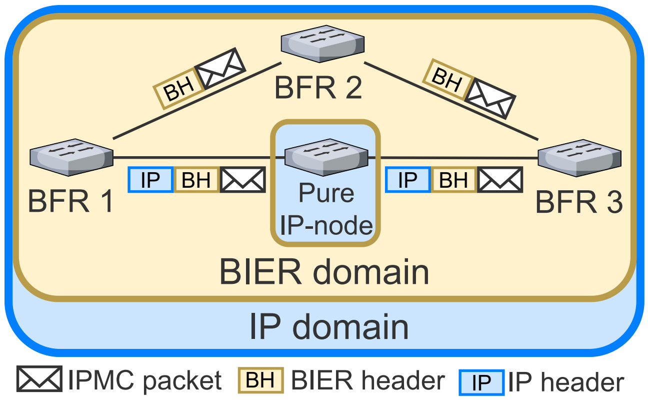


Figure 5: BIER traffic forwarded over pure IP nodes requires additional IP encapsulation.

both IP and BIER devices. The three BFRs are BFR-NBRs to each other. BFR 1 and 2 are neighbors to each other in both the IP and BIER topology because they are directly connected on Layer 2. Therefore, they exchange BIER packets on Layer 2 without an additional header. Since the pure IP node is not part of the BIER topology, BFR 1 and BFR 3 are BFR-NBRs although they are not neighbors in the IP topology. To exchange packets, BFR 1 and BFR 3 encapsulate BIER packets with an IP header and forward them via the pure IP node. When BFR 1 or 3 receive the packet, they remove the IP header and process the BIER header.

6. BIER Fast Reroute

The necessity for resilience mechanisms in BIER networks has been discussed in [41] without proposing any mechanism. In this section we introduce BIER fast reroute (BIER-FRR) to protect BIER traffic against link and node failures by taking advantage of reconvergence and FRR mechanisms of the routing underlay. We explain why regular BIER cannot protect BIER traffic sufficiently against failures, and present BIER-FRR for link and node protection, respectively. Finally, we discuss the protection properties of BIER-FRR.

6.1. Link Protection

In this paragraph we introduce BIER-FRR with link protection. First, we explain why relying on the available features of BIER and a resilient routing underlay is not sufficient for protection against link failures. Afterwards, we describe BIER-FRR with link protection and show a forwarding example.

6.1.1. Resilience Problems of BIER for Link Failures

BFR-NBRs may be directly connected over Layer 2 or they may be reachable only over Layer 3 so that IP encapsulation is needed for them to exchange BIER traffic (see Section 5.3.3). This has impact on the effect of link failures.

If neighboring BFRs are reachable only over Layer 3, they exchange BIER traffic IP-encapsulated towards each other. If a link on the path towards the BFR-NBR fails, the BFR-NBR is not reachable until the routing underlay has restored reachability. This may be due to IP-FRR, which is fast, or IP routing reconvergence, which is slow. In any case, the reachability on the BIER layer is also restored and no further action needs to be taken. Possibly, the path in the routing underlay changed, which may affect the neighboring relationships among BFRs, so that BIFTs need to be recomputed. This, however, is not time-critical.

If BFR-NBRs are directly connected over Layer 2, they exchange packets without an additional IP header. If the link between them is broken, protection mechanisms on Layer 3, in particular IP-FRR, cannot help because the BIER packet is not equipped with an IP header. Therefore, affected BIER traffic cannot be forwarded until a new BFR-NBR is provided in the BIFT for affected BFERs. Thus, the BIFT needs to be updated. This process takes time to recompute the entries based on the new paths from the routing underlay and starts only after reconvergence of the routing underlay has completed. This is significantly later than FRR mechanisms on the routing underlay restore connectivity for unicast traffic.

BIER-FRR with link protection effects that a BFR affected by a link failure can forward BIER traffic again as soon as its connectivity problem is detected on the BIER layer and the routing underlay is able to forward unicast traffic again.

6.1.2. *BIER-FRR with Link Protection*

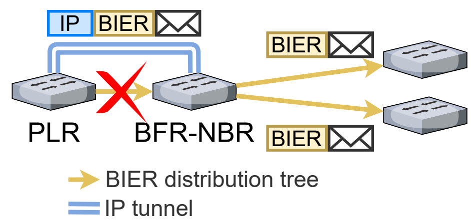


Figure 6: BIER-FRR with link protection is needed for BFR-NBRs which are directly connected on Layer 2: they IP-encapsulate BIER traffic towards a BFR-NBR after it is detected unreachable.

The concept of BIER-FRR with link protection is illustrated in Figure 6. BFRs must be able to detect link failures. This may happen, e.g., through loss of light detection or through bidirectional forwarding detection (BFD) with BFR-NBRs [42]. If an unreachable BFR-NBR is detected, a BFR IP-encapsulates BIER traffic towards that BFR-NBR. As a result, the BIER traffic will reach the affected BFR-NBR again as soon as reachability on the routing underlay is restored. This can be very fast if the routing underlay leverages FRR. When the traffic arrives at the BFR-NBR, the additional IP header is removed and packets are processed as normal BIER traffic.

6.1.3. *Example for BIER-FRR with Link Protection*

Figure 7 shows the BIER topology from the earlier forwarding example in Figure 4 with a link failure. For convenience, the BIFT of BFR 1 is displayed again in Table 3.

When BFR 1 sends a BIER packet to all other BFERs, the BitString is `1110`. First a packet copy is successfully deliverd to BFER 2 and BFER 4 so that the BitString of the remaining packet is `0100`, i.e., next a packet must be forwarded to BFER 3. However, BFR-NBR 3 is unreachable for BFR 1 due to the link failure. Therefore, BFR 1 IP-encapsulates the BIER packets towards BFR 3. As

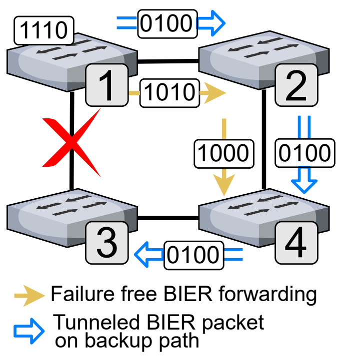


Figure 7: Packet paths and example topology for BIER-FRR with link protection.

BFER	F-BM	BFR-NBR
1	-	-
2	1010	2
3	0100	3
4	1010	2

Table 3: BIFT of BFR 1.

soon as the routing underlay restores connectivity, the IP-encapsulated BIER packets is detoured via BFR 2 and BFR 4 towards BFR 3. Thus, BIER-FRR with link protection may send a second packet copy over a link.

6.2. *Node Protection*

We introduce BIER-FRR with node protection. First, we discuss why regular BIER cannot protect BIER traffic sufficiently fast against node failures. Afterwards, we present the concept of BIER-FRR with node protection, extend the BIFT with backup entries, show a forwarding example, and explain how backup entries are computed.

6.2.1. *Resilience Problems of BIER for Node Failures*

If a BFR fails, all downstream BFRs are unreachable. This problem cannot be quickly repaired by the routing underlay because traffic directed to the failed node cannot be delivered. Thus, alternate BFR-NBRs are needed. These are provided when the routing underlay has reconverged and the BIFT entries are recomputed. This, however, may take long time.

BIER-FRR with node protection shortens this time so that affected BIER traffic can be delivered in the presence of node failures as soon as connectivity for unicast traffic is restored in the routing underlay.

6.2.2. *BIER-FRR with Node Protection*

We propose BIER-FRR with node protection to deliver BIER traffic even if the BFR-NBR fails. The concept is shown in Figure 8. When a PLR cannot

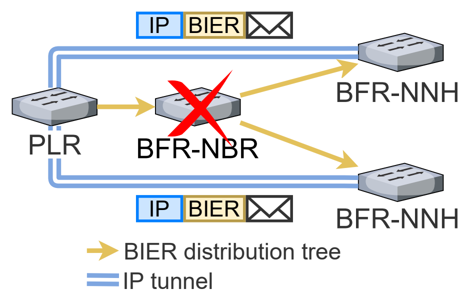


Figure 8: Concept of BIER-FRR with node protection.

reach a BFR-NBR, it tunnels copies of the BIER packet to all BFR next-next-hops (BFR-NNH) in the distribution tree that should receive or forward a copy. Thus, for each such BFR-NNH, an individual packet copy is created. The packet is then tunneled to the BFR-NNH with an additional header of the routing underlay; these packets are delivered as soon as the routing underlay restores connectivity. When the BFR-NNH receives such a packet, it removes the tunnel header and processes the resulting BIER packet.

If a BFR-NBR is unreachable, the link towards the BFR-NBR or the BFR-NBR itself may have failed. Therefore, the BFR-NBR should also receive a packet copy encapsulated by the routing underlay.

When a packet copy is sent to multiple BFR-NNHs instead of the BFR-NBR, the the BitString of the forwarded packet copies must be modified appropriately to avoid duplicate packets at BFERs. These modifications can be obtained with backup F-BMs, which is explained in more detail in Section 6.2.5.

6.2.3. *BIFT with Backup Entries*

To support BIER-FRR with node protection, the BIFT must be extended with backup entries. The structure of a BIFT with backup entries is shown in Table 4.

BFER	F-BM	BFR-NBR
1	primary F-BM backup F-BM	primary NH backup NH
...	...	...

Table 4: Structure of a BIFT with backup entries.

The normal BIFT entries are called primary entries. The backup entries have the same structure as the primary entries. When a BFR-NBR is reachable, the primary entries are used for forwarding. If a BFR-NBR becomes unreachable, the corresponding backup entry is used for forwarding in the same way as a primary entry with only one difference. The packet is not forwarded natively but instead it is always tunneled to the backup NH through the routing underlay.

6.2.4. Example for BIER-FRR with Node Protection

Figure 9 shows an example topology and Figure 10 illustrates the distribution tree for BFR 1 and BFR 2 based on the paths from the routing underlay. Table 5 shows the BIFT of BFR 1 with primary and backup entries.

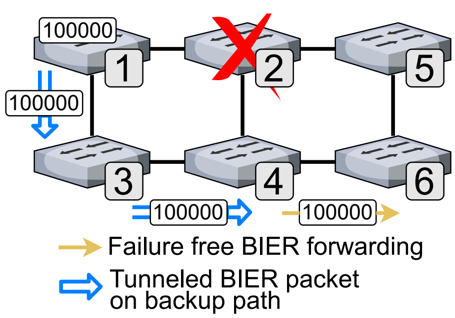


Figure 9: A packet is sent from BFR 1 to BFR 6 over a backup path using node protection.

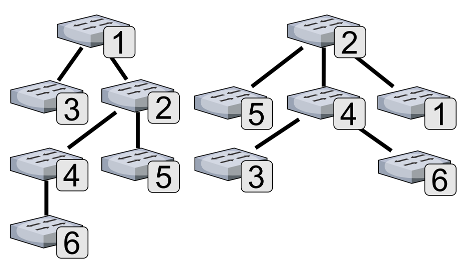


Figure 10: Shortest-path tree of BFR 1 and BFR 2.

We illustrate the forwarding with BIER-FRR when BFR 2 fails. We assume that BFR 1 needs to send a BIER packet to BFR 6, i.e. the packet contains the BitString `100000`. As BFR 2 is unreachable, the primary NH of BFR 1

BFER	F-BM	BFR-NBR
1	000001	-
	-	-
2	111010	2
	000010	2
3	000100	3
	000100	3
4	111010	2
	101000	4
5	111010	2
	010000	5
6	111010	2
	101000	4

Table 5: BIFT of BFR 1 with primary and backup entries.

to BFR 6, which is BFR 2, cannot be used. Therefore, the backup entry is utilized. That means, the backup F-BM `101000` (see Table 5) is applied to the copy of the BIER packet and then it is tunneled through the routing underlay to the backup NH BFR 4. BFR 1 applies the complement of the backup F-BM `010111` to the BitString of the original BIER packet which is then `000000`. As the BitString of the remaining BIER packet has no activated bits anymore, the forwarding process terminates at BFR 1.

The routing underlay delivers the packet copy from BFR 1 to BFR 4 as soon as connectivity is restored. BFR 4 removes the tunnel header and forwards the BIER packet to BFR 6.

If the BitString of the packet was `100100`, i.e., BFER 3 should have received a copy of the packet, too, a regular BIER packet would have been forwarded directly to BFR 3 before BIER-FRR tunnels another copy of the BIER packet to BFR 4. Thus, BIER-FRR with node protection may increase the traffic on a link to ensure that all relevant NNHs receive a packet copy.

6.2.5. Computation of Backup Entries

We compute backup NHs and backup F-BMs for BFERs at a specific BFR which we call PLR in this context. To that end, we distinguish two cases: the BFER is not a BFR-NBR (1) or it is a BFR-NBR (2).

In the first case, the BFER is reached from the PLR through the routing underlay via a considered NH and next-next-hop (NNH). The considered NNH becomes the backup NH for the BFER. The corresponding backup F-BM requires activated bits for a set of BFERs. This set comprises all BFERs whose paths in the routing underlay from the PLR also traverses the considered NH and NNH. This F-BM can be computed by bitwise AND'ing the PLR's F-BM for the considered BFER and the considered NH's F-BM.

In the second case, the considered BFER is a BFR-NBR. Then, the NH is also taken as backup NH. This ensures that the NH receives a copy of the BIER packet if the NH cannot be reached due to a link failure. To avoid that the NH distributes further packet copies, the backup F-BM contains only the activated bit for the considered BFER.

We illustrate both computation rules by an example. We consider the BIFT of BFR 1 in Table 5. The backup entry of BFER 6 is an example for the first computation rule. The backup NH for BFR 6 is BFR 4 as it is the NNH of BFR 1 on the path towards BFR 6 in Figure 10. The BFERs reachable from the PLR through BFR 4 are BFER 4 and BFER 6. Therefore, the backup F-BM is `101000`. It can be obtained by bitwise AND'ing the F-BM of BFR 1 for BFER 6 (`111010`) and the F-BM of BFR 2 for BFER 6 (`101100`). The latter can be derived from the multicast subtree of BFR 2 in Figure 10.

The backup entry of BFER 2 is an example for the second computation rule. The backup NH for BFER 2 is BFR 2 and the F-BM contains only one activated bit for BFER 2 (`000010`).

6.3. Properties of BIER-FRR

We have argued that restoration of BIER connectivity may take long time in case of a link failure since this process can start only after the reconvergence of the routing underlay has completed. To shorten the outage time, we introduced BIER-FRR which restores connectivity on the BIER layer as soon as unreachable BFR-NBRs are detected and the connectivity in the routing underlay is restored.

The general concept of BIER-FRR is simple: it requires some sort of detection that a BFR-NBR is no longer reachable, but it does not require any additional signalling as it is a local mechanism. Furthermore, it leverages the restoration of routing underlay so that BIER traffic can profit from FRR mechanisms in the routing underlay. It does not define alternate paths on the BIER layer, which is in contrast to another solution reported in [43].

BIER-FRR comes in two variants: link protection and node protection. Link protection is simple, it just encapsulates BIER traffic into a header of the routing underlay, but it cannot protect against node failures. The encapsulated packet may be sent over an interface over which also a regular copy of the same BIER-packet is transmitted. That means, up to two packet copies can be transmitted over at most one link in case of a failure, which runs in contrast to the actual idea of multicast.

Node protection is more complex. It requires a PLR to send backup copies of a BIER packet to all relevant NNHs encapsulated with a header of the routing underlay. This requires extensions to the BIFT for backup entries. However, it protects against link and node failures. The encapsulated packets may be sent over interfaces over which also a regular copy of the same BIER packet is transmitted. That means that even multiple packet copies can be transmitted over several links in case of a failure.

BIER-FRR is designed for single link and node failures. In case of multiple failures, BIER-FRR suffers from potential shortcomings of the routing underlay to cope with multiple failures, too, so that some traffic may be lost until the BIFT is updated. Furthermore, if both a NH and a NNH fail, the subtree of the NNH is no longer reachable until the BIFTs are updated. Some FRR techniques may cause routing loops in case of multiple failures [12]. In contrast, BIER-FRR cannot cause routing loops because it just leverages the routing underlay and does not propose new paths in failure cases.

6.4. Application of IP-FRR Mechanism on BIER Layer

In Section 3.1 we introduced IP-FRR and described LFAs. In [43] we discussed the application of LFAs on the BIER layer, i.e., in addition to the primary BFR-NBR, the BIFT contains a backup BFR-NBRs respectively, to which a BIER packet is forwarded when the primary NH is unreachable. We identified two major disadvantages. First, their application leaves a significant amount of BFERs unprotected against link or node failures because LFAs cannot guarantee full protection coverage [12]. This holds in particular when node protection is desired for which protection coverage is even lower than for link protection. Second, LFAs on the BIER layer introduce new paths in the BIER topology, which can cause rerouting loops for BIER traffic. Third, this approach assumes IP with IP-FRR as routing underlay while our approach works with any routing underlay and FRR mechanism. Therefore, we argue that the application of IP-FRR mechanisms on BIER layer is not sufficient for appropriate protection.

7. Introduction to P4

This section serves as a primer for readers who are not familiar with P4. First, we explain the general P4 processing pipeline. Then, we describe the concept of match+action tables, control blocks, and metadata. Finally, we explain the recirculate and clone operations.

7.1. P4 Pipeline

P4 is a high-level language for programming protocol-independent packet processors [44]. Its objective is a flexible description of data planes. It introduces the forwarding pipeline shown in Figure 11. A programmable parser reads packets and stores their header information in header fields which are carried together with the packet through the pipeline. The overall processing model is composed of two stages: the ingress and the egress pipeline with a packet buffer in between. The egress port of a packet has to be specified in the ingress pipeline.

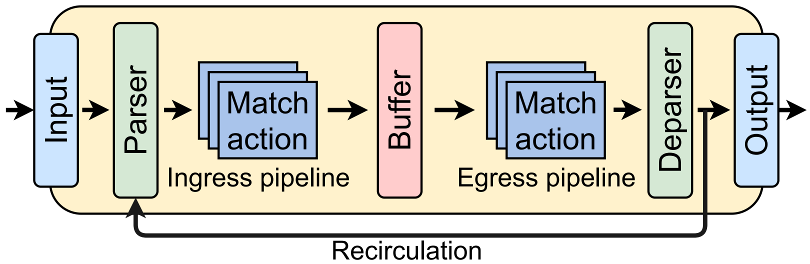


Figure 11: P4 abstract forwarding pipeline according to [44].

If no egress port has been specified for a packet at the end of the egress pipeline, the packet is dropped. At the end of the egress pipeline, a deparser constructs the packet with new headers according to the possibly modified header fields. P4 supports the definition and processing of arbitrary headers. Therefore, it is not bound to existing protocols.

7.2. Metadata

Metadata constitute packet-related information. There are standard and user-defined metadata. Examples for standard metadata are ingress port or reception time which are set by the device. User-defined metadata store arbitrary data, e.g., processing flags or calculated values. Each packet carries its own instances of standard and user-define metadata through the P4 processing pipeline.

7.3. Match+Action Tables

Match+action tables are used within the ingress and egress pipeline to apply actions to specified packets. The P4 program describes the structure of each match+action table. The rules are the contents of the table and are added to the table during runtime. ~~When a match+action table is applied to a packet, the rules are checked for a match and the action associated with the matching rule is carried out. A miss occurs if no rule matches.~~

As match+action tables are essential for the description of our prototype, we introduce a compact notation for them by an example. The example is given in Figure 12. The table has the name "MAT_Simple_IP" and describes an implementation of simplified IP forwarding with match+action tables. In the following we use the prefix "MAT_" for naming MATs.~~It is simplified as we renounce on next hop indication.~~

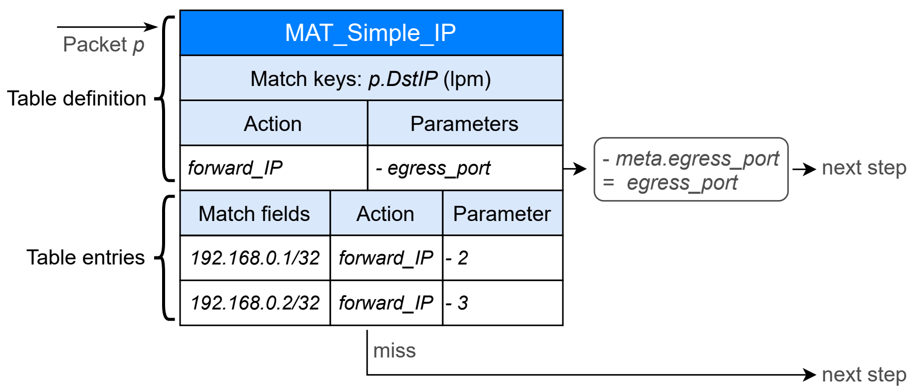


Figure 12: Match+action table for simplified IP forwarding.

7.3.1. Match Part

A table defines a list of match keys that describe which header fields or meta-data are used for matching a packet against the table. The match type indicates the matching method. P4 supports several match types: exact, longest-prefix (lpm), and ternary. The latter features a wildcard match. ~~Ternary rules are associated with a priority to specify the matching entry when multiple wildcard patterns match. The match type lpm selects the rule with the longest matching prefix. When the match type is exact, it is not possible to install two match fields that match on the same pattern.~~ In our example in Figure 12, the match key is the destination IP address and lpm matching is applied.

7.3.2. Actions

The table further defines a list of actions including their signature which can be used by rules in case of a match. Actions are similar to functions in common programming languages and consist of several primitive operations. Inside an action further actions can be executed. ~~In addition, basic protocol-independent~~

~~operations (assignment, arithmetic, etc.) can be performed.~~ Actions can modify header fields and metadata of a packet. In our example, this is the *forward_IP* action that requires the appropriate egress port as a parameter. Each action is illustrated by a flow chart on the right side of the table.

7.3.3. Rules

During runtime, the match+action tables can be filled with rules through an application programming interface (API). ~~, i.e., table entries, whose structure depends on the described match keys, defined actions, and their signatures.~~ The rules contain match fields which are patterns that are to be matched against a packet's context selected by the match keys. In our example, the match fields are IP addresses. The rules further specify an action ~~offered~~ in the table definition and suitable parameters which are applied to the packet in case of a match. ~~In our example, this is only the forward action but multiple entries with different match fields and parameters can be installed by the controller.~~

In our example in Figure 12 we install two rules. In the first one, the match field is the IP address *192.168.0.1* and it applies the action *forward_IP* with the parameter *2*. This will send packets with the destination IP *192.168.0.1* over port *2*. The match field for the second rule is *192.168.0.2* and it sends the packet over port *3*. For all other destination IPs a miss occurs and no egress port is specified.

When describing match+action tables of our implementation in Section 8, we omit the actual rules as they are configuration data and not part of the P4 implementation.

7.4. Control Blocks

A control block consists of a sequence of match+action tables, operations and if-statements. ~~Control blocks organize the order in which match+action tables are applied to a packet.~~ They encapsulate functionality. Within control blocks other control blocks can be called. ~~The behavior of the ingress and egress pipeline are defined by the ingress and egress control flow.~~ Both the ingress and egress pipeline are control blocks that apply other control blocks. We use the prefix "CB_" for naming of our other control blocks. Examples of control blocks in our implementation are *CB_IPv4*, *CB_BIER*, or *CB_Ethernet*.

7.5. Recirculation

P4 does not support native loops. However, as indicated in Figure 11, the recirculation operation returns a packet to the beginning of the ingress pipeline. ~~It can be called anywhere in the ingress or egress control flow.~~ It activates a standard metadata field, i.e., a flag, which marks the packet for recirculation. The packet still traverses the entire pipeline and only at the end of the egress pipeline the packet is returned to the start of the ingress pipeline. When setting the *recirculate* flag, it is possible to specify which metadata fields should be kept during recirculation. All others are reset to their default values. In contrast, header fields modified during the processing remain modfied after recirculation. Another standard metadata field stores whether a packet has been recirculated.

7.6. Packet Cloning

P4 supports the packet cloning operation clone-ingress-to-egress (*CI2E*). *CI2E* can be called anywhere in the ingress pipeline. This activates the *CI2E* metadata flag which indicates that the packet should be cloned. However, the copy is created only at the end of the ingress pipeline. In the packet clone all header changes are discarded that have been made within the ingress pipeline. If *CI2E* has been called within the ingress pipeline, two packets enter the egress pipeline. One is the original packet that has been processed by the ingress control flow. The second packet is the copy without modifications from the ingress pipeline. Figure 13 illustrates this by an example.

When the *CI2E* flag is set, it is possible to specify for the clone whether metadata fields should persist or be reset. When a packet clone enters the egress pipeline, an additional standard metadata flag identifies the packet as a clone. This allows different processing for original and cloned packets.

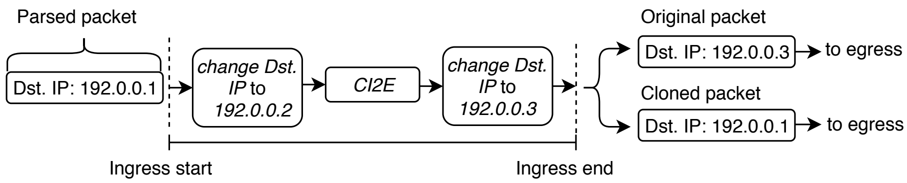


Figure 13: Illustration of the clone-ingress-to-egress (*CI2E*) operation: the destination IP of the clone is the one of the received packet although IP was modified before *CI2E* was called.

8. P4-Based Implementation of BIER and BIER-FRR

In this section, we describe the P4-based implementation of IP, IP-FRR, BIER, and BIER-FRR. We first describe the data plane followed by the control plane. In the end, we briefly explain our codebase.

8.1. Data Plane

First, we specify the handling of packet headers, then, we give a high-level overview of the processing pipeline, followed by a detailed explanation of applied control blocks.

8.1.1. Packet Header Processing

P4 requires that potential headers of a packet are defined a priori. Our implementation supports the header suite Ethernet/outer-IP/BIER/inner-IP. We use the inner IP header for regular forwarding and the outer IP header for FRR. During packet processing, headers may be activated or deactivated. Deactivated headers are not added by the deparser. *Encaps* actions in our implementation activate a specific header and set header fields. *Decaps* actions deactivate specific headers. ~~P4 also supports header stacks to apply the same header type several times. However, we do not leverage this feature~~.

8.1.2. Overview of Ingress and Egress Control Flow

Figure 14 shows an overview of the entire data plane implementation which is able to perform IP and BIER forwarding as well as IP-FRR and BIER-FRR. It

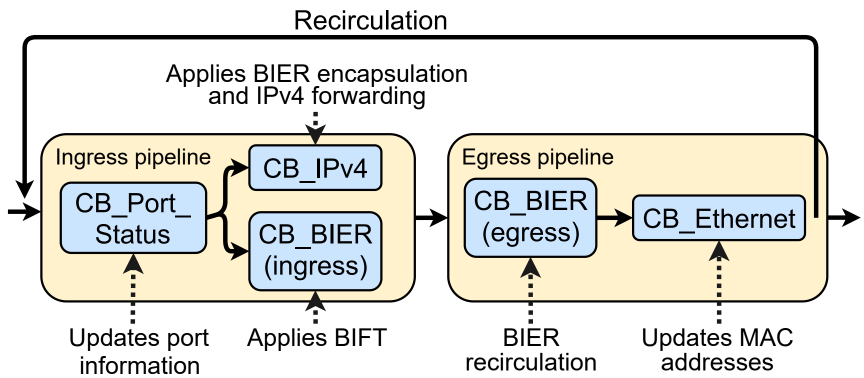


Figure 14: Overview of ingress and egress control flow.

is divided into ingress and egress control flow which are given as control blocks. In the ingress and egress control block the CB_IPv4 and CB_BIER control block are only applied to their respective packets, i.e., the CB_IPv4 control block is applied to IP packets and the CB_BIER control block is applied to BIER packets. We first summarize their operations and describe their implementation in detail in the following Sections.

When a packet enters the ingress pipeline, it is processed by the *CB_Port_Status* control block. It updates the port status (up/down) and records it in the user-defined metadata *meta.live_ports* of the packet. This possibly triggers FRR actions later in the pipeline. Then, the *CB_IPv4* control block or the *CB_BIER* control block is executed depending on the packet type.

The *CB_IPv4* control block is applied to both unicast and multicast IP packets. Unicast packets are processed by setting an appropriate egress port, possibly using IP-FRR in case of a failure. IPMC packets entering the BIER domain are equipped with a BIER header and recirculated for BIER forwarding. IPMC packets leaving the BIER domain are forwarded using native multicast.

The *CB_BIER* control block is applied to BIER packets. There is a *CB_BIER* control block for the ingress control flow and another for the egress control flow. A processing loop for BIER packets is implemented which extends over both *CB_BIER* control blocks. At the beginning of the processing loop in the ingress flow the BitString is copied to metadata *meta.remaining_bits*. This metadata is used to track for which BFERs a copy of the BIER packet still needs to be sent. Then, rules from the *MAT_BIFT* are applied to the packet. This also comprises BIER-FRR actions which encapsulate BIER packets with an IP header if necessary. Within these procedures, the BIER packet is cloned so that the original packet and a clone enter the egress control flow. The processing loop stops if the *meta.remaining_bits* are all zero.

In the *CB_BIER* control block of the egress control flow, the *recirculate* flag is set for cloned packets. At the end of the egress control flow, the clone is recirculated to the ingress control flow with modified *meta.remaining_bits* to continue the processing loop. The non-cloned BIER packet is just passed to the *CB_Ethernet* control block.

The *CB_Ethernet* control block updates the Ethernet header of each packet. Then, the packet is sent if an egress port is set and the *recirculate* flag has not been activated. If the *recirculate* flag is activated, the packet is recirculated instead. This applies to cloned BIER packets in the processing loop or to packets that require a second pass through the pipeline: BIER-encapsulated IPMC packets, BIER-decapsulated IPMC packets, IP-encapsulated BIER packets, or IP-decapsulated BIER packets. If neither *recirculate* flag is activated and nor the egress port is set, the packet is dropped.

8.1.3. CB_Port_Status Control Block

The control block *CB_Port_Status* records whether a port is up or down in the user-defined metadata *meta.live_ports* of a packet. Figure 15 shows that it consists of only the match+action table *MAT_Port_Status*.

The table does not define any match keys. As a result, the first entry matches every packet. We install only a single rule which calls the action *set_port_status*. It copies the parameter *live_ports* to the user-defined metadata *meta.live_ports*. *Meta.live_ports* is a bit string where each bit corresponds to a port of the switch. If the port is currently up, the bit is activated, otherwise, the bit is deactivated. The metadata field *meta.live_ports* is later used by both the *CB_IPv4*

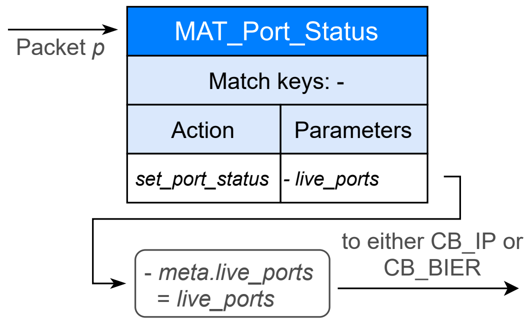


Figure 15: In the control block *CB_Port_Status* the table *MAT_Port_Status* copies the information about live ports to the user-defined metadata field *meta.live_ports* of the packet.

and *CB_BIER* control block to decide whether IP-FRR and BIER-FRR should be applied. The parameter *live_ports* in the table is updated by the local controller when the port status changes, which will be explained in Section 8.2.1.

8.1.4. CB_IPv4 Control Block

The *CB_IPv4* control block handles IPv4 packets. Its operation is shown in Figure 16.

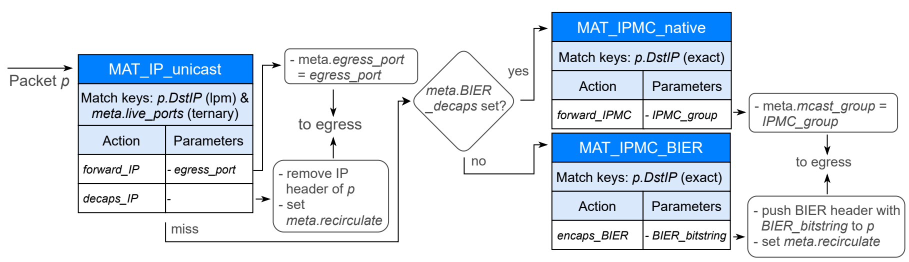


Figure 16: The *CB_IPv4* control block handles IPv4 packets.

It leverages three match+action tables: *MAT_IP_unicast*, *MAT_IPMC_native*, and *MAT_IPMC_BIER*. Packets are processed by these tables depending on their type. *MAT_IP_unicast* performs IP unicast forwarding including IP-FRR. IPMC packets encounter a miss and are relayed by the control flow to *MAT_IPMC_native* or *MAT_IPMC_BIER*. *MAT_IPMC_native* performs native multicast forwarding for IPMC packets leaving the BIER domain while *MAT_IPMC_BIER* just adds a BIER header for IPMC packets entering the BIER domain.

MAT_IP_unicast. This match+action table uses the IP destination address and the metadata *meta.live_ports* as match keys. The IP destination address is associated with a longest prefix match and the *meta.live_ports* with a ternary match. ~~The table works similarly to our introductory example in Figure 12, but has more actions and implements IP-FRR~~. We first explain our implementation of IP-FRR. The rules contain an IP prefix and a *required_port* pattern as match fields (not shown in the table). *Required_port* corresponds to a bit string of all egress ports and is a wildcard expression with only a single zero or one for the primary egress port of the traffic, i.e., `*...*0*...*` or `*...*1*...*`. If FRR is desired for an IP prefix, two rules are provided: a primary rule with `*...*1*...*` as *required_port* pattern, and a backup rule with `*...*0*...*`.

~~A different FRR approach for P4 has been proposed in [45]. When a packet is sent to a failed port, the packet is recirculated and steered to another port. However, for our prototype implementation, we were unable to apply actions to a packet after sending it to a port. Therefore, we developed the above mentioned FRR approach.~~

The table offers two actions: *forward_IP* and *decaps_IP*. We explain both in the following in detail.

The *decaps_IP* action is applied to packets that are addressed to the node itself. For such rules the *required_port* pattern is set to `*...*`. Those IP packets are typically BIER packets that have been encapsulated in IP by other nodes for BIER-FRR. Therefore, the IP header is removed and the *recirculate* flag is set so that the packet can be forwarded as BIER packet in a second pass of the pipeline. In theory, other IP packets with the destination IP addresses of the node itself may have reached their final destination. They need to be handed over to a higher layer within the node. However, this feature is not required in our prototype so that we omit it in our implementation.

The *forward_IP* action is applied for other unicast address prefixes and requires an *egress_port* as parameter. It sets the *meta.egress_port* to the indicated egress port so that the packet is switch-internally relayed to the right egress port. The IP-FRR mechanism as explained above may be used in conjunction with *forward_IP* to provide an alternate egress port when the primary egress port is down. This mechanism allows implementation of LFAs. ~~In some scenarios there is no LFA available [13]. This problem is not specific to our implementation and there are ways to solve this problem with appropriate tunnels [12].~~

IPMC addresses encounter a miss in this table so that their packets are further treated by the control flow in the *CB_IPv4* control block. It checks whether the *meta.BIER_decaps* bit has been set. If so, the IPMC packet came from the BIER domain and has been decapsulated. Therefore, it is relayed to the *MAT_IPMC_native* table for outbound IPMC traffic. Otherwise, the IMPC packet has been received from a host and requires forwarding through the BIER domain. Therefore, it is relayed to the *MAT_IPMC_BIER* table.

MAT_IPMC_native. This match+action table implements native IPMC forwarding. It is used by a BFER to send IPMC packets to hosts outside the BIER domain that have subscribed to a specific IPMC group. The table *MAT_IPMC_native* uses the IP destination address as match key with an exact match. It defines only the *forward_IPMC* action and requires a switch-internal multicast group as parameter, which is specific to the IPMC group (IP destination address) of the packet. The action sets this parameter in the *meta.mcast_group* of the packet. As a consequence, the packet is processed by the native multicast feature of the switch. This results in packet copies for every egress port contained in the switch-internal multicast group *meta.mcast_group* with the corresponding egress port set in the metadata of the packets. The set of egress ports belonging to that group can be defined through a target-specific interface, which is done by the controller in response to received IGMP packets. Packets encountering a miss in this table are dropped at the end of the pipeline.

MAT_IPMC_BIER. This match+action table uses the IP destination address as match key with an exact match. It defines only the *encaps_BIER* action and requires the bit string as parameter, which is specific to the IPMC group (IP destination address) of the packet. The action pushes a BIER header onto the packet and sets the specified BitString. ~~This bit string indicates the set of all BFERs that should receive traffic from this IPMC group.~~ Then the *recirculate* flag is set so that the packet can be forwarded as a BIER packet in a second pass of the pipeline. Packets encountering a miss in this table are dropped at the end of the pipeline.

8.1.5. CB_BIER Control Block

The *CB_BIER* control block processes BIER packets. It is illustrated in Figure 17.

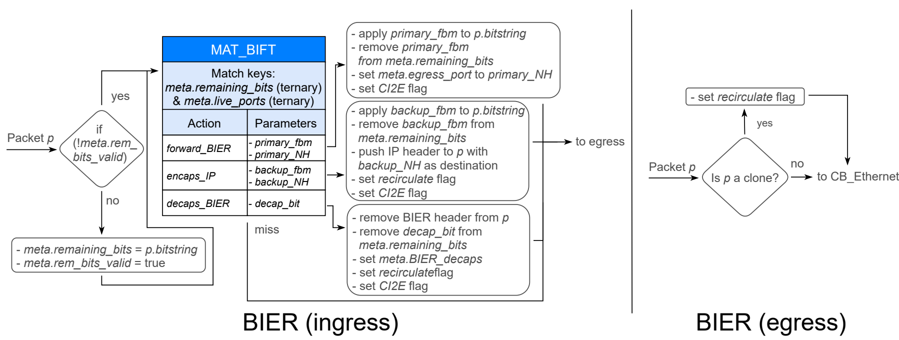


Figure 17: The *CB_BIER* control blocks in the ingress and egress pipeline implement BIER fowarding as a processing loop.

The user-defined metadata *meta.remaining_bits* is used during BIER processing to account for the BFERs that still need a copy of the packet. It serves as a control variable for the processing loop. When a BIER packet is processed by the *CB_BIER* control block for the first time, *meta.remaining_bits* is initialized with the BitString of the packet. The user-defined metadata *meta.remaining_bits_valid* is initially zero. It is activated after *meta.remaining_bits* is initialized and prevents overwriting
meta.remaining_bits when the packet is recirculated.

Then the match+action table *MAT_BIFT* is applied. It implements BIER forwarding including BIER-FRR according to the principle we developed for IP-FRR in Section 8.1.4. Match keys are the packet's *meta.remaining_bits* indicating BFERs, and *meta.live_ports* indicating live egress ports. The match types are ternary. Rules are provided for all individual BFERs both for failure-free cases and failure cases. The match field of these rules consists of two bit strings that we call *dest_BFER* and *required_port* (not shown in the table). The *dest_BFER* bit string has the bit position for the respective BFER activated and all other bit positions set to wildcards (`*...*1*...*`). The *required_port* bit string is used as in Section 8.1.4 to select between primary and backup rules. In case of a match, there are three possible actions.

Decaps_BIER is called by the rule whose activated bit in *dest_BFER* refers to the node itself. It has a F-BM with only the bit of the BFER activated and no primary or backup NH. If this rule matches, the node should receive a copy of the packet. The action removes the BIER header of the packet, activates the user-defined metadata flag *meta.BIER_decaps*, and the *recirculate* flag so that the resulting IPMC packet is processed in a second pass of the pipeline. In addition, the complement of F-BM is used to clear the bit for the processing node itself in *meta.remaining_bits*.

Forward_BIER is called by rules whose activated bit in *dest_BFER* refers to other nodes and where the *required_port* bit string indicates that the egress port works. Thus, *forward_BIER* is used for primary forwarding. It has the primary F-BM and the primary NH (egress port) as parameters. The primary F-BM is applied to clear bits from the BitString of the packet and the complement of the backup F-BM is applied to *meta.remaining_bits*. In addition, *meta.egress_port* is set to the primary NH.

Encaps_IP is called by rules where the *required_port* bit string indicates that the primary egress port does not work for the BFER specified in *dest_BFER*. Thus, *encaps_IP* is used for backup forwarding. It has the backup F-BM and the backup NH (IP address) as parameters. The backup F-BM is applied to clear bits from the BitString of the packet and the complement of the backup F-BM is applied to *meta.remaining_bits*. Then, an IP header is pushed with the destination address of the backup NH. The *recirculate* flag for the packet is activated as it requires IP forwarding in a second run through the pipeline.

At the end of *decaps_BIER*, *forward_BIER*, and *encaps_IP*, a flag for *CI2E* is set. This effects that a packet copy is generated at the end of the ingress pipeline. For the copy (clone), the *recirculate* flag is activated in the *CB_BIER* control block in the egress control flow. With this packet, the BIER processing loop continues. The *meta.remaining_bits* information must be kept to account for the BFERs that still need a packet copy.

When packets enter the *MAT_BIFT* table with *meta.remaining_bits* equal to zero, they encounter a miss. As a result, they are dropped at the end of the pipeline, which stops the processing loop for these BIER packets.

8.1.6. CB_Ethernet Control Block

The *CB_Ethernet* control block is visualized in Figure 18.

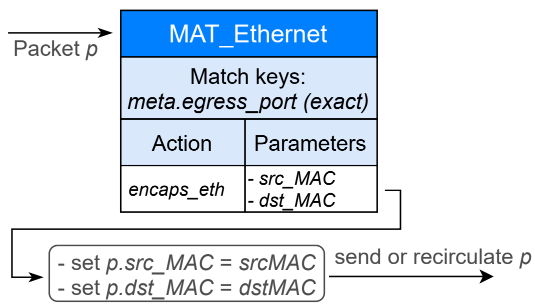


Figure 18: *CB_Ethernet* control block.

It applies the match+action table *MAT_Ethernet* to all packets. The match key is the egress port of the packet and the match type is exact. Only the action *encaps_eth* is defined which requires the parameters *src_MAC* and *dst_MAC*. It updates the Ethernet header of the packet by setting the source and destination MAC address which are provided as parameters. Rules are added for every egress port.

This behavior is sufficient as we assume that any hop is an IP node. Although MAC addresses are not utilized for packet switching, they are still necessary as packet receivers in Mininet discard packets if their destination MAC address does not match their own address.

The control plane is visualized in Figure 19. It consists of one global con-

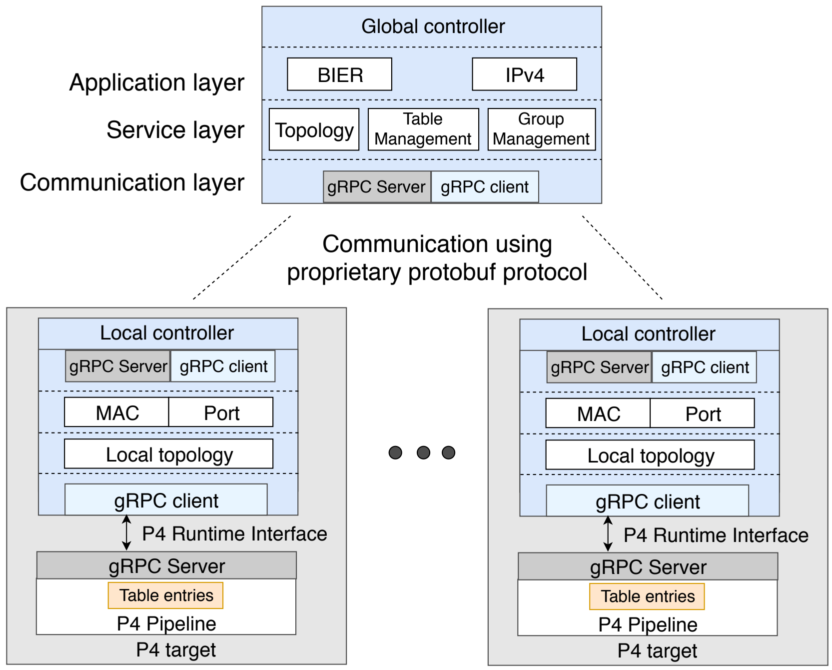


Figure 19: Controller architecture.

troller and one local controller per switch. The local controllers run directly on the switch hardware as P4 switches are mostly whiteboxes. The local controller takes care of tasks that can be performed locally while the global controller is in charge of configuration issues that require a global network view. In theory, a single controller could perform all tasks. However, there are three reasons that call for a local controller: scalability, speed, and robustness. Performing local tasks at the local controller relieves the global controller from unnecessary work. A local controller can reach the switch faster than a global controller. And, most important, a local controller does not need to communicate with the switch via a network. In case of a network failure, the local controller still reaches the switch while the global controller may be unable to do so. Local controllers have also been applied for similar reasons in LoCoSDN [46], P4-MACSec [47], and P4-IPSec [48]. In the following we explain the local and global controller in more detail.

8.2.1. Local Controller

Each switch has a local controller. Switch and local controller communicate via the so-called P4 Runtime which is essentially the Southbound interface in the SDN context. The P4 Runtime uses a gRPC channel and a protobuf-based communication protocol. It allows the controller to write table entries on the switch.

Figure 19 shows that the local controller keeps information about the local topology, learns about neighboring nodes, and port status, and configures this information in the tables of the switch. Moreover, it relays some packets to the global controller and writes table entries as a proxy for the global controller.

We leverage the local controller for three local tasks that we describe in the following: IGMP handling, neighbor discovery, and port monitoring.

IGMP Handling. Multiple hosts are connected to a switch. They leverage the Internet Group Message Protocol (IGMP) to join and leave IPMC groups. If the switch receives an IGMP packet, it forwards it to its local controller which then configures the switch for appropriate actions. For example, it adds a new host to the IPMC group and configures the native IPMC feature of the switch to deliver IPMC packets to the hosts. That feature is used only for carrying multicast traffic from the switch to the hosts. To populate the *MAT_IPMC_native* table, the local controller utilizes the Thrift channel instead of the P4 Runtime as this API is target-specific.

Neighbor Discovery. For neighbor discovery, we implemented a simple proprietary topology recognition protocol. All nodes announce themselves to their neighbors. It allows the local controller to learn the MAC address of the neighbor for each egress port. The local controller stores this information in the match+action table *MAT_Ethernet* which is utilized in the *CB_Ethernet* control block (see Section 8.1.6).

Port Monitoring. A P4 switch by itself is not able to find out whether a neighboring node is reachable. However, a fast indication of this information is crucial to support FRR. In a real network a local controller may test for neighbor reachability, e.g., using a BFD towards all neighbors, loss-of-light, loss-of-carrier, or any other suitable mechanism. Then, the local controller configures this information as a bit string in the match+action table *MAT_Port_Status* of the switch whenever the port status changes. Failure detection is target-dependent and out of scope of this document. Therefore we trigger failure processing of the local controller manually with a software signal. The local controller then activates IP-FRR and BIER-FRR if enabled and notifies the global controller for recomputation of forwarding entries.

8.2.2. Global Controller

We divide the architecture of the global controller in three layers: communication, service, and application (see Figure 19).

The communication layer is responsible for the communication with the local controllers. Each switch is connected to its local controller. Since the P4 runtime only allows one controller with write access, the global controller cannot directly control the switches. Therefore, it communicates with the local controllers to configure the switches. All changes calculated by the global controller are sent to the local controller using a separate channel. The local controller forwards the changes to the switch using the P4 runtime interface.

The service layer provides services for the application layer. This includes information about the topology, multicast groups, and entries in the tables on the switches. The application layer utilizes that information to calculate the table entries.

The global controller receives IGMP messages and keeps track of subscriptions to IPMC groups. If a host is the first to enter or the last to leave an IPMC group at a BFER, the global controller configures the *MAT_IPMC_BIER* table of all BFIRs with an appropriate bit string for the specific IPMC group by activating or deactivating the corresponding bit of the BFER. As a result, the BFIR starts or stops sending traffic from this IPMC group to the BFER.

The global controller sets all entries in the *MAT_IP_unicast* and *MAT_IPMC_BIER* tables of all switches and the entries in the *MAT_BIFT*s. If the global controller is informed by a local controller about a failure, it first reconfigures the *MAT_IP_unicast* and *MAT_IPMC_BIER* tables and then the entries of the *MAT_BIFT*s accordingly.

8.3. Codebase

The implementation of the BIER data plane and control plane including a demo can be downloaded at https://github.com/uni-tue-kn/p4-bier. The provided code contains a more detailed documentation of the BIER(-FRR) implementation. The demo contains several Mininet network topologies that were used to verify the functionality of BIER(-FRR). One of them is described in Section 9.1. Links can be disabled using Mininet, which enables the verification of the BIER-FRR mechanism. A simple host CLI allows multicast packets to be sent and incoming multicast packets to be displayed.

9. Evaluation

In this section we illustrate that BIER traffic is better protected with BIER-FRR. To that end, we conduct experiments in a testbed using our prototype. We first explain the experimental setup, the timing behavior of our emulation and our metrics. Finally, we describe the testbed setup and present experimental protection results in an BIER/IP network with and without IP-FRR and BIER-FRR, for link protection and node protection, respectively.

9.1. Methodology

First, we describe the general approach for our evaluation. Then, we discuss the timing behavior of a software-based evaluation. As the prototype switch is differently controlled than typical routers, we adapt reaction times of the controller after a failure to mimic the timely behaviour of updates for IP forwarding tables and BIFTs. Finally, we explain our metrics.

9.1.1. General Setup

We emulate different topologies in Mininet [49]. The core network is implemented with our P4-based prototype and the software-based *simple_switch* which is based on the BMv2 framework [50]. It forwards IP unicast, IP multicast, and BIER traffic. One source and several subscribers are connected to the core network. The source periodically sends IP unicast and IP multicast packets. IP unicast packets are forwarded as usual through the core network. When IP multicast packets enter the core network, they are encapsulated with a BIER header at the BFIR. BFERs remove BIER headers and forward the IP multicast packets to the subscribers.

Rules for the match+action tables are computed by the global controller in an initial setup phase. In different scenarios we simulate link and node failures and observe packet arrivals at the subscribers. We study different combinations of IP-FRR and BIER-FRR to evaluate the delay until subscribers receive traffic again after a failure has been detected. Also in those cases, the local controller notifies the global controller to perform IP reconvergence and BIFT recomputation because FRR is meant to be only a temporary measure until the global forwarding information base has been updated as a response to the link or node failure.

We report events at the PLR and at all subscribers before and after the failure. For the PLR we show the following signals: failure detection at t_0, updates of IP forwarding entries, and updates of BIFT entries. For the subscribers we record receptions of unicast and multicast packets.

9.1.2. Timing Behavior

Our switch implementation in a small, virtual environment has a different timing behavior than a typical router in a large, physical environment. In particular signaling can be executed with insignificant delay in our virtual environment, e.g., notifying the global controller about the failure or the distribution of updated forwarding entries. This is different with routers and routing protocols in the physical world. Signaling requires significant time as routing protocols need to exchange information about the changed topology. Routers compute alternative routes and push them to their forwarding tables. Only after all unicast paths have been recomputed and globally updated by the routing underlay, BFRs can compute new forwarding entries for BIER and push them to their BIFTs. Thus, the BIFT is updated only significantly later compared to the unicast forwarding information base. To respect that in our evaluation, we configure the global controller to install new IP forwarding entries on the switches only after 150 ms after being informed about a failure and new BIFT entries another 150 ms later.

~~In contrast, IP-FRR and BIER-FRR are local mechanisms which take effect as soon as the node is informed about a failure. Therefore, we do not delay their actions in our experiments.~~

9.1.3. *Metric*

We perform experiments with and without IP-FRR and BIER-FRR, and compare the time after which unicast and multicast traffic is delivered again at the subscribers after a failure has been detected by the affected BFR.

9.2. *Link Protection*

We perform experiments for the evaluation of BIER-FRR with link protection. First, we explain the experimental setup. Afterwards, we report and discuss the results for all scenarios.

9.2.1. *Setup for Link Protection*

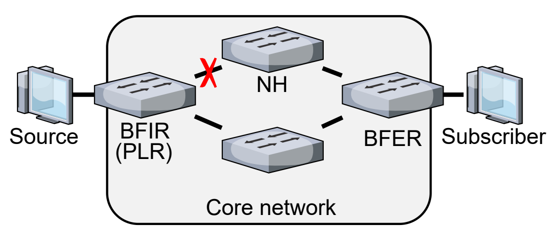


Figure 20: Two hosts the *Source* and the *Subscriber* are connected to a BIER network with IP as the routing underlay.

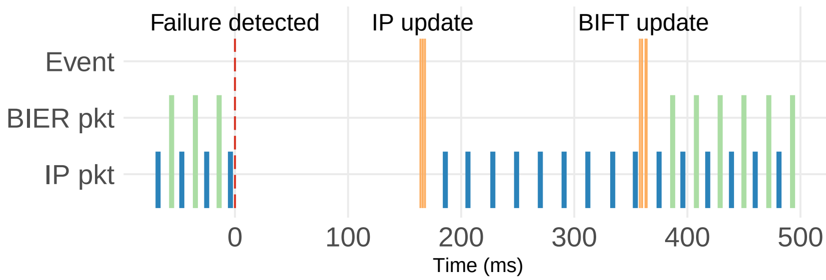
We emulate the testbed depicted in Figure 20 in Mininet. Two hosts the *Source* and the *Subscriber* are connected to a BIER/IP network. The host *Source* sends every 10 ms packets to the host *Subscriber* over the core network. Every other packet is sent by IP unicast and IPMC. The primary path carries packets from PLR via NH to $BFER$. We simulate the failure of the link between the PLR and the NH to interrupt packet delivery. We compare the time until the host *Subscriber* receives unicast and multicast traffic again, after the failure has been detected by the PLR. We perform experiments with and without IP-FRR and BIER-FRR with link protection.

9.2.2. Without IP-FRR and BIER-FRR

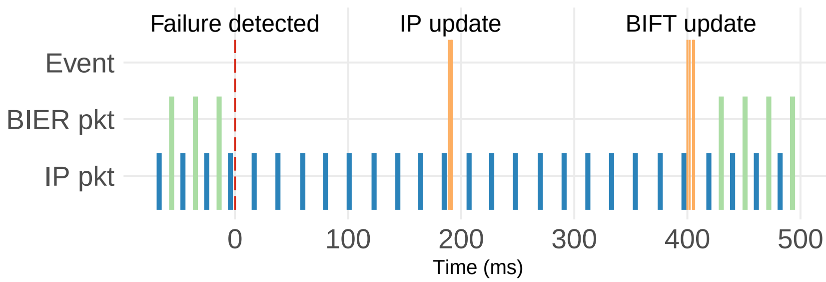
In the first experiment, failure recovery is based only on IP reconvergence and BIFT recomputation. Neither IP-FRR nor BIER-FRR are enabled. Figure 21(a) shows that the failure interrupts packet delivery at the *Subscriber*. Unicast reconvergence is completed after about 170 ms after failure detection. Updating the BIFT entries has finished only after about 370 ms in total. Unicast and multicast packets are received again by the *Subscriber* only after updated IP and BIER forwarding rules from the controller have been installed at the PLR.

9.2.3. With IP-FRR but without BIER-FRR

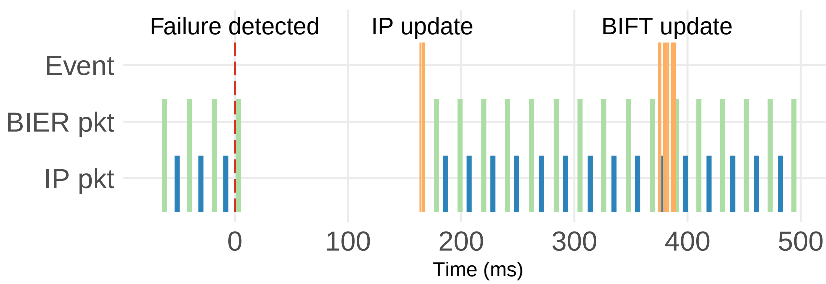
In the second experiment, IP-FRR is enabled but BIER-FRR remains disabled. Figure 21(b) shows that IP unicast traffic immediately benefits from IP-FRR when the PLR detects the failure. IP-FRR instantly reroutes packets and, therefore, IP unicast traffic is still delivered at the *Subscriber*. Both IP reconvergence and BIFT recompuration are finished slightly later compared to the previous scenario. The reason for the extended duration is that the global controller needs to compute new forwarding entries for IP-FRR during reconvergence, which is not needed if IP-FRR is disabled. After 200 ms, IP reconvergence has finished and the primary IP unicast forwarding entries have been updated. Multicast packets are delivered only after BIFT recomputation



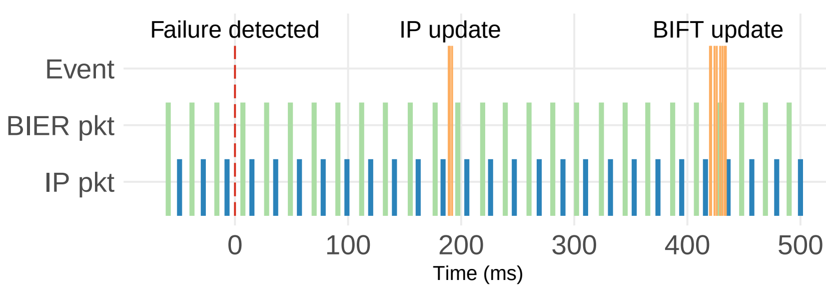
(a) Without IP-FRR and BIER-FRR.



(b) With IP-FRR but without BIER-FRR.



(c) Without IP-FRR but with BIER-FRR.



(d) With IP-FRR and BIER-FRR.

Figure 21: Reception time of packets in the link failure scenario.

after about 400 ms.

9.2.4. Without IP-FRR but with BIER-FRR

In the third experiment, IP-FRR is disabled but BIER-FRR is enabled. Figure 21(c) shows that unicast traffic is delivered at the *Subscriber* when IP reconvergence has finished after about 170 ms. Due to BIER-FRR, BIER traffic benefits from the faster IP reconvergence, too. Multicast traffic is delivered after 170 ms as well, and not only after BIFT recomputation. The BIFT is updated only after about 400 ms in total which is slightly longer than in the scenario without BIER-FRR. Although conceptually the BIFT does not require modification for BIER-FRR with link protection, the match+action tables in the P4 implementation need backup entries that tunnel BIER packets in case of a failure. Therefore, the global controller has to compute new backup entries for BIER-FRR in addition to primary BIFT entries during the recomputation process. The slightly delayed BIFT recomputation is not a disadvantage for BIER traffic because BIER-FRR reroutes BIER packets until both primary and backup BIFT entries have been updated.

9.2.5. With IP-FRR and BIER-FRR

In the last experiment, IP-FRR and BIER-FRR are enabled. Figure 21(d) illustrates that both unicast and multicast traffic are delivered at the *Subscriber* without any delay despite of the failure. This is achieved by FRR mechanisms in both the routing underlay and the BIER layer. IP-FRR immediately restores connectivity for unicast traffic. BIER-FRR leverages the resilient routing underlay to immediately reroute BIER packets. IP reconvergence has finished after about 200 ms. BIFT recomputation finishes only after about 420 ms. In both cases the longer time is explained by the additional FRR entries the global controller has to compute during IP reconvergence and BIFT recomputation, respectively.

9.3. Node Protection

In this paragraph we evaluate BIER-FRR with node protection. First, we describe the experimental setup. Then, we report and discuss the evaluation results for all four scenarios.

9.3.1. Setup for Node Protection

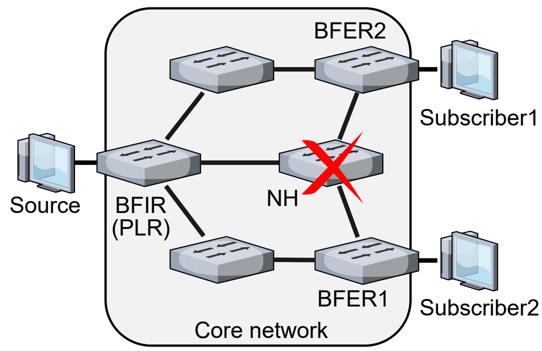
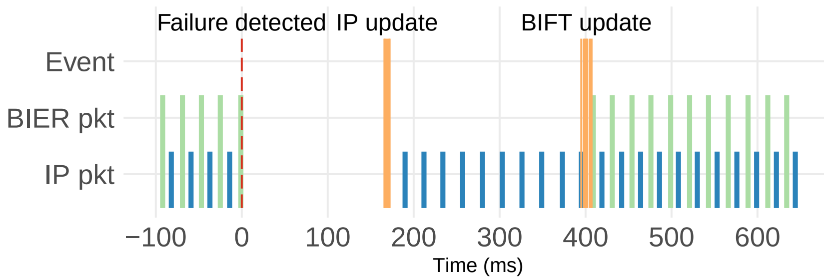
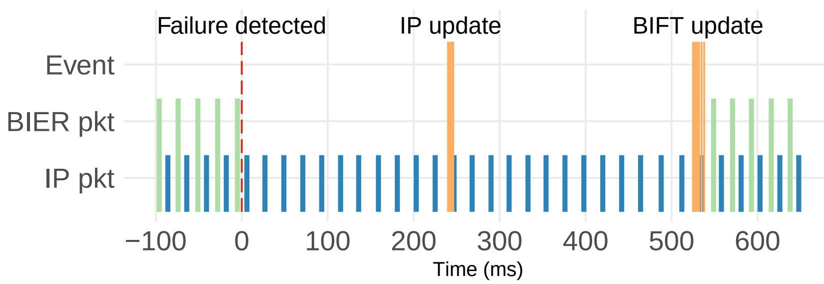


Figure 22: Three hosts, *Source*, *Subscriber*1 and *Subscriber*2 are connected to a BIER network with IP as the routing underlay.

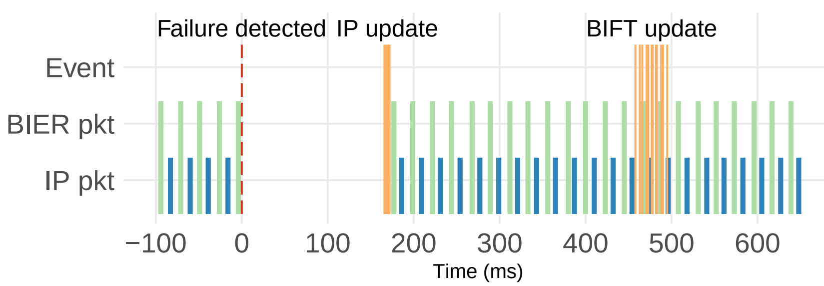
Figure 22 shows the topology we emulated in Mininet. The three hosts *Source*, *Subscriber*1, and *Subscriber*2 are connected to an BIER/IP network. The *Source* alternately sends two IP unicast packets and one IP multicast packet with 10 ms in between. The unicast packets are sent to *Subscriber*1 and *Subscriber*2. The IPMC group of the the IPMC packet is subscribed by *Subscriber*1 and *Subscriber*2. On the primary path, packets are carried from the PLR via the NH to $BFER1$ and $BFER2$, respectively. We simulate the failure of the NH to interrupt packet delivery with a node failure. We evaluate the time until both the *Subscriber*1 and the *Subscriber*2 receive traffic again after the PLR detects the failure. We perform experiments with and without IP-FRR and BIER-FRR with node protection. We discuss the outcome and show figures only for *Subscriber*1 because results for *Subscriber*2 are very similar.



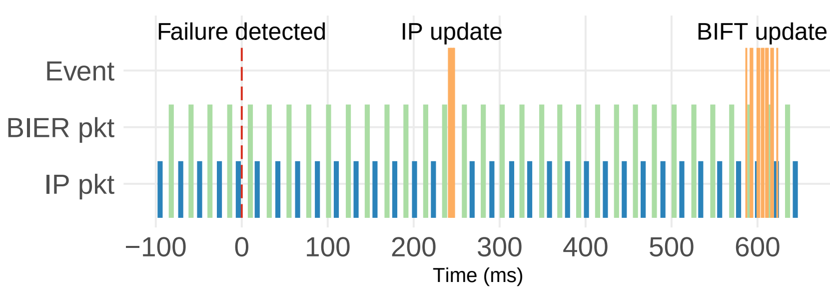
(a) Without IP-FRR and BIER-FRR.



(b) With IP-FRR but without BIER-FRR.



(c) Without IP-FRR but with BIER-FRR.



(d) With IP-FRR and BIER-FRR.

Figure 23: Reception time of packets in the node failure scenario.

9.3.2. *Without IP-FRR and BIER-FRR*

In the first scenario, the local controller at the PLR triggers only IP reconvergence and BIFT recomputation after failure detection. No FRR measures are enabled. Figure 23(a) shows that the $Subscriber1$ receives IP unicast traffic only after IP reconvergence which takes about 180 ms. $Subscriber1$ receives multicast traffic only after BIFT recomputation which takes about 400 ms. Both IP reconvergence and BIFT recomputation require slightly more time than in the link failure scenario because now the local controller reports a node failure which requires more rules to be recomputed.

9.3.3. *With IP-FRR but without BIER-FRR*

In the second scenario, IP-FRR is enabled but not BIER-FRR. Figure 23(b) shows that IP unicast traffic immediately benefits from IP-FRR. Traffic is delivered at the $Subscriber1$ without any delay despite of the failure. IP reconvergence requires about 240 ms. Multicast traffic is received by the $Subscriber1$ only after BIFT recomputation which has finished only after about 520 ms. Again, IP reconvergence and BIFT recomputation require slightly more time than without IP-FRR because because additional IP-FRR entries have to be computed.

9.3.4. *Without IP-FRR but with BIER-FRR*

In the third scenario, BIER-FRR is enabled but not IP-FRR. Figure 23(b) shows that both IP unicast and multicast traffic are received at the $Subscriber1$ only after IP reconvergence which takes about 170 ms. Afterwards, IP traffic is rerouted because of the updated forwarding entries. BIER traffic is rerouted after that time as well, because BIER-FRR leverages the updated routing underlay instead of requiring BIFT recomputation which has finished only after about 500 ms.

9.3.5. *With IP-FRR and BIER-FRR*

In the last scenario, both IP-FRR and BIER-FRR are enabled. Figure 23(d) shows that both IP unicast and multicast traffic are received by the $Subscriber1$

without any delay despite of the failure. IP-FRR reroutes IP unicast traffic as soon as the failure is detected by the PLR. Similarly, BIER-FRR reroutes BIER traffic immediately, too. Therefore, BIER traffic benefits from the resilience of the routing underlay to forward BIER traffic although the NH failed and BIFT recomputation has not finished, yet. IP reconvergence takes about 240 ms. BIFT recomputation finished only after 600 ms.

10. Conclusion

BIER is a novel, domain-based, scalable multicast transport mechanism for IP networks that does not require state per IP multicast (IPMC) group in core nodes. Only ingress nodes of a BIER domain maintain group-specific information and push a BIER header on multicast traffic for simplified forwarding within the BIER domain. Bit-forwarding routers (BFRs) leverage a bit index forwarding table (BIFT) for forwarding decisions. Its entries are derived from the interior gateway protocol (IGP), the so-called routing underlay. In case of a failure, the BIFT entries are recomputed only after IP reconvergence. Therefore, BIER traffic encounters rather long outages after link or node failures and cannot profit from fast reroute (FRR) mechanisms in the IP routing underlay.

In this work, we proposed BIER-FRR to shorten the time until BIER traffic is delivered again after a failure. BIER-FRR deviates BIER traffic around the failure via unicast tunnels through the routing underlay. Therefore, BIER benefits from fast reconvergence or FRR mechanisms of the routing underlay to deliver BIER traffic as soon as connectivity for unicast traffic has been restored in the routing underlay. BIER-FRR has a link and a node protection mode. Link protection is simple but cannot protect against node failures. To that end, BIER-FRR offers a node protection mode which requires extensions to the BIFT structure.

As BIER defines new headers and forwarding behavior, it cannot be configured on standard networking gears. Therefore, a second contribution of this paper is a prototype implementation of BIER and BIER-FRR on a P4-

programmable switch based on $P4_{16}$. It works without extern functions or other extensions such as local agents that impede portability. The switch offers an API for interaction with controllers. A local controller takes care of local tasks such as MAC learning and failure detection. A global controller configures other match+action tables that pertain to forwarding decisions. A predecessor of this prototype without BIER-FRR and based on $P4_{14}$ has been presented as a demo in [5]. The novel BIER prototype including BIER-FRR demonstrates that P4 facilitates implementation of rather complex forwarding behavior.

~~In future work, we would like to implement BIER and BIER-FRR on hardware switches and on FPGA cards, improve the scalability of the forwarding behaviour through aggregation of BIFT entries, and conduct extensive performance tests regarding throughput considering various multicast group configurations. Furthermore, the scalability issues due to BIER's limited header size, i.e., existing or novel solutions and their impact, should be investigated.~~

We deployed our prototype on a virtualized testbed based on Mininet and the software switch BMv2. Our experiments confirm that BIER-FRR significantly reduces the time until multicast traffic is received again by subscribers after link or node failures. Without BIER-FRR, multicast packets arrive at the subscriber only after reconvergence of the routing underlay and BIFT recomputation. With BIER-FRR, multicast traffic is delivered again as soon as connectivity in the routing underlay is restored, which is particularly fast if the routing underlay applies FRR methods.

Acknowledgment

The authors thank Wolfgang Braun and Toerless Eckert for valuable input and stimulating discussions.

References

[1] D. Merling, M. Menth, et al., An Overview of Bit Index Explicit Replication (BIER), IETFJournal (Mar. 2018).

[2] I. Wijnands, E. Rosen, et al., RFC8279: Multicast Using Bit Index Explicit Replication (BIER), https://tools.ietf.org/html/rfc8279 (Nov. 2017).

[3] M. Shand, S. Bryant, IP Fast Reroute Framework, https://tools.ietf.org/html/rfc5714 (Jan. 2010).

[4] D. Merling, M. Menth, BIER Fast Reroute, https://datatracker.ietf.org/doc/draft-merling-bier-frr/ (Mar. 2019).

[5] W. Braun, J. Hartmann, et al., Demo: Scalable and Reliable Software-Defined Multicast with BIER and P4, IFIP/IEEE International Symposium on Integrated Network Management (IM) (May 2017).

[6] The P4 Language Consortium, The P4 Language Specification Version 1.0.5, https://p4.org/p4-spec/p4-14/v1.0.5/tex/p4.pdf (Nov. 2018).

[7] The P4 Language Consortium, The P4 Language Specification Version 1.1.0, https://p4.org/p4-spec/p4-14/v1.0.5/tex/p4.pdf (Nov. 2018).

[8] H. Holbrook, B. Cain, et al., Using Internet Group Management Protocol Version 3 (IGMPv3) and Multicast Listener Discovery Protocol Version 2 (MLDv2) for Source-Specific Multicast, https://tools.ietf.org/html/rfc4604 (Aug. 2006).

[9] T. Speakman, J. Crowcroft, et al., PGM Reliable Transport Protocol Specification, https://tools.ietf.org/html/rfc3208 (Dec. 2001).

[10] G. Rétvári, J. Tapolcai, et al., IP fast ReRoute: Loop Free Alternates revisited, IEEE Conference on Computer Communications (Apr. 2011).

[11] D. Katz, D. Ward, et al., Bidirectional Forwarding Detection (BFD), https://tools.ietf.org/html/rfc5880 (Jun. 2010).

[12] D. Merling, W. Braun, et al., Efficient Data Plane Protection for SDN, IEEE Conference on Network Softwarization and Workshops (Jun. 2018).

[13] M. Menth, M. Hartmann, et al., Loop-Free Alternates and Not-Via Addresses: A Proper Combination for IP Fast Reroute?, Computer Networks 54 (Jun. 2010).

[14] A. Raj, O. Ibe, et al., A survey of IP and multiprotocol label switchingfast reroute schemes, Computer Networks 51 (Jun. 2007).

[15] V. S. Pal, Y. R. Devi, et al., A Survey on IP Fast Rerouting Schemes using Backup Topology, International Journal of Advanced Research inComputer Science and Software Engineering 3 (Apr. 2003).

[16] S. Bryant, S. Previdi, et al., A Framework for IP and MPLS Fast Reroute Using Not-Via Addresses, https://tools.ietf.org/html/rfc6981 (Aug. 2013).

[17] L. Csikor, G. Rétvári, et al., IP fast reroute with remote Loop-Free Alternates: The unit link cost case, International Congress on Ultra Modern Telecommunications and Control Systems (Feb. 2012).

[18] S. Islam, N. Muslim, et al., A Survey on Multicasting in Software-Defined Networking, IEEE Communications Surveys Tutorials 20 (Nov. 2018).

[19] Z. Al-Saeed, I. Ahmada, et al., Multicasting in Software Defined Networks: A Comprehensive Survey, Journal of Network and Computer Applications 104 (Feb. 2018).

[20] J. Rückert, J. Blendin, et al., Software-Defined Multicast for Over-the-Top and Overlay-based Live Streaming in ISP Networks, Journal of Network and Systems Management 23 (Apr. 2015).

[21] J. Rückert, J. Blendin, et al., Flexible, Efficient, and Scalable Software-Defined Over-the-Top Multicast for ISP Environments With DynSdm, IEEE Transactions on Network and Service Management 13 (Sep. 2016).

[22] L. H. Huang, H.-J. Hung, et al., Scalable and Bandwidth-Efficient Multicast for Software-Defined Networks, IEEE Global Communications Conference (Dec. 2014).

[23] S. Zhou, H. Wang, et al., Cost-Efficient and Scalable Multicast Tree in Software Defined Networking, Algorithms and Architectures for Parallel Processing (Dec. 2015).

[24] J.-R. Jiang, S.-Y. Chen, Constructing Multiple Steiner Trees for Software-Defined Networking Multicast, Proceedings of the 11th International Conference on Future Internet Technologies (Jun. 2016).

[25] Y.-D. Lin, Y.-C. Lai, et al., Scalable Multicasting with Multiple Shared Trees in Software Defined Networking, Journal of Network and Computer Applications 78 (Jan. 2017).

[26] Z. Hu, D. Guo, et al., Multicast Routing with Uncertain Sources in Software-Defined Network, IEEE/ACM International Symposium on Quality of Service (Jun. 2016).

[27] B. Ren, D. Guo, et al., The Packing Problem of Uncertain Multicasts, Concurrency and Computation: Practice and Experience 29 (August 2017).

[28] A. Iyer, P. Kumar, et al., Avalanche: Data Center Multicast using Software Defined Networking, International Conference on Communication Systems and Networks (Jan 2014).

[29] W. Cui, C. Qian, et al., Scalable and Load-Balanced Data Center Multicast, IEEE Global Communications Conference (Dec 2015).

[30] S. H. Shen, L.-H. Huang, et al., Reliable Multicast Routing for Software-Defined Networks, IEEE Conference on Computer Communications (April 2015).

[31] M. Popovic, R. Khalili, et al., Performance Comparison of Node-Redundant Multicast Distribution Trees in SDN Networks, International Conference on Networked Systems (Apr. 2017).

[32] T. Humernbrum, B. Hagedorn, et al., Towards Efficient Multicast Communication in Software-Defined Networks, IEEE International Conference on Distributed Computing Systems Workshops (Jun. 2016).

[33] D. Kotani, K. Suzuki, et al., A Multicast Tree Management Method Supporting Fast Failure Recovery and Dynamic Group Membership Changes in OpenFlow Networks, Journal of Information Processing 24 (2016).

[34] T. Pfeiffenberger, J. L. Du, et al., Reliable and Flexible Communications for Power Systems: Fault-tolerant Multicast with SDN/OpenFlow, International Conference on New Technologies, Mobility and Security (Jul. 2015).

[35] W. K. Jia, L.-C. Wang, et al., A Unified Unicast and Multicast Routing and Forwarding Algorithm for Software-Defined Datacenter Networks, IEEE Journal on Selected Areas in Communications 31 (Dec. 2013).

[36] M. J. Reed, M. Al-Naday, et al., Stateless Multicast Switching in Software Defined Networks, IEEE International Conference on Communications (May 2016).

[37] A. Giorgetti, A. Sgambelluri, et al., First Demonstration of SDN-based Bit Index Explicit Replication (BIER) Multicasting, European Conference on Networks and Communications (Jun. 2017).

[38] A. Giorgetti, A. Sgambelluri, et al., Bit Index Explicit Replication (BIER) Multicasting in Transport Networks, International Conference on Optical Network Design and Modeling (May 2017).

[39] T. Eckert, G. Cauchie, et al., Traffic Engineering for Bit Index Explicit Replication BIER-TE, http://tools.ietf.org/html/draft-eckert-bier-te-arch (Nov. 2017).

[40] W. Braun, M. Albert, et al., Performance Comparison of Resilience Mechanisms for Stateless Multicast Using BIER, IFIP/IEEE International Symposium on Integrated Network Management (May 2017).

[41] Q. Xiong, G. Mirsky, et al., The Resilience for BIER, https://datatracker.ietf.org/doc/draft-xiong-bier-resilience/ (Mar. 2019).

[42] Q. Xiong, G. Mirsky, et al., BIER BFD, https://datatracker.ietf.org/doc/draft-hu-bier-bfd/ (Mar. 2019).

[43] D. Merling, S. Lindner, et al., Comparison of Fast-Reroute Mechanisms for BIER-Based IP Multicast, International Conference on Software Defined Systems (Apr. 2020).

[44] P. Bosshart, D. Daly, et al., P4: Programming Protocol-Independent Packet Processors, ACM SIGCOMM Computer Communication Review 44 (Jul. 2014).

[45] R. Sedar, M. Borokhovich, et al., Supporting emerging applications with low-latency failover in p4, ACM SIGCOMM Workshop on Networking for Emerging Applications and Technologies (Aug. 2018).

[46] M. Schmidt, F. Hauser, et al., LoCoSDN: A Local Controller for Operation of OFSwitches in non-SDN Networks, Software Defined System (Apr. 2018).

[47] F. Hauser, M. Schmidt, et al., P4-MACsec: Dynamic Topology Monitoring and Data Layer Protection with MACsec in P4-Based SDN, IEEE Access (Mar. 2020).

[48] F. Hauser, M. Häberle, et al., P4-IPsec: Implementation of IPsec Gateways in P4 with SDN Control for Host-to-Site Scenarios, ArXiv (Jul. 2019).

[49] B. Lantz, B. Heller, et al., A Network in a Laptop: Rapid Prototyping for Software-defined Networks, ACM SIGCOMM HotNets Workshop (Oct. 2010).

[50] p4lang, behavioral-model, `https://github.com/p4lang/behavioral-model` (Mar. 2019).