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

Daniel Merling, Steffen Lindner, Michael Menth
University of Tübingen, Department of Computer Science, 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 can support a large number of IPMC groups with large churn rates. The contribution of this work is threefold. First, we propose a simple 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 P4 technology, which is a programmable data plane. 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 observed for unicast traffic in the routing underlay.

Index Terms—Software-Defined Networking, P4, Bit Index Explicit Replication, Multicast, Resilience, Scalability

I. INTRODUCTION

IP multicast (IPMC) is leveraged for many services like IPTV, multicast VPN, or the distribution of financial or broadcast data. IPMC efficiently forwards one-to-many traffic to all desired destinations by sending at most one packet copy per involved link. Traditional IPMC methods require per-IPMC-group state within routers, leading to various scalability issues. First, core routers are involved in the establishment, removal, and also possibly in the change of an IPMC group. This may be a heavy burden for routers supporting IPMC groups with high churn rates. The number of IPMC groups may be large and require lots of space in forwarding tables. And in case of a failure, the forwarding of any IPMC group possibly requires fast update, which is demanding in a critical network situation.

Therefore, 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 bitstring that indicates the receivers of the IPMC group within the BIER domain. Based on that bitstring the packets are forwarded through the BIER domain without requiring per-IPMC-group state in core routers.

BIER leverages a bit indexed forwarding table (BIFT) for forwarding decisions. Its entries are derived from forwarding tables of 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 BIFT can be updated only after the routing underlay has reconverged, which takes time. To reduce packet loss during that process, FRR techniques have been proposed for IP networks. As the loss of an IPMC packet implies that all receivers in the same IPMC subtree lack this packet, this is a severe problem for BIER whose service would profit from an appropriate FRR solution. The objective is a simple FRR scheme which reuses the service of FRR in the routing underlay instead of introducing a new concept.

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 [3]. 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 [4], we presented an early version of a P4-based prototype for BIER, BIER-TE, and BIER-TE-FRR. It was based on the P14 specification of P4 [5] 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 P16 specification of P4 [6]. The new prototype implements IP forwarding, a simple form of IP-FRR, BIER forwarding, and BIER-FRR. The evaluation of the prototype shows that BIER-FRR enables BIER to benefit from both fast reconvergence of the routing underlay and even faster FRR mechanisms thereof so that IPMC traffic is hit no longer by network failures than unicast traffic in the routing underlay. The prototype 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. Thus, the contribution of this paper is threefold: (1) a concept for BIER-FRR, (2) an approach to implement BIER and BIER-FRR with P4, and (3) a controller hierarchy with local controllers to support FRR techniques. Finally, the P4-based prototype demonstrates the usefulness of BIER-FRR.

The remainder of this paper is structured as follows. Section II reviews basics of multicast and Section III discusses related work. Section IV gives a primer on BIER and Section V introduces BIER-FRR. In Section VI we summarize foundations of P4 needed for the understanding of the
BIER prototype. Section [VII] 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 [VIII]. Finally, Section [IX] summarizes this paper and discusses further research issues.

II. TECHNOLOGICAL BACKGROUND FOR IP MULTICAST

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) [7].

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 computed for each source-specific multicast tree. 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) [8] reduces packet loss for multicast traffic. It enables receivers to detect lost or out-of-order packets and supports retransmission requests similar to TCP.

III. RELATED WORK

We review work in the context of SDN-based multicast. Mostly 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 forwarding state which runs contrary to the intention of BIER.

A. Multicast Implementations with OpenFlow

The surveys [9], [10] 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) [11], [12] leverages multiple trees to load-balance multicast traffic and efficiently handle group subscription changes. Modified Steiner tree problems are considered in [13], [14] to minimize the total cost of edges and the number of branch nodes, or to additionally minimize the source-destination delay [15]. In [16], the authors compute and install traffic-engineered shared multicast trees using OpenFlow. In [17] and [18], traffic-engineered Steiner trees are computed that minimize the number of edges of the tree and provide optimized locations for multicast sources in the network. The Avalanche Routing Algorithm (AvRA) [19] considers topological properties of data center networks to optimize utilization of network links. Dual-Structure Multicast (DuSM) [20] improves scalability and link utilization for SDN-based data center networks by utilizing different forwarding approaches for high-bandwidth and low-bandwidth flows. In [21], 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 [22], 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 [23] 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 some switches.

B. Multicast Protection with OpenFlow

Kotani et al. [24] 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 is notified and chooses an appropriate backup multicast tree and reconfigures senders accordingly. This mechanism differs from BIER-FRR in that it requires significant signalling effort in response to a failure.

The authors of [25] 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 requires differs from BIER-FRR in that it requires many different backup trees configured in the network.

C. Improved Multicast Forwarding for SDN Switches

Some contributions improve the efficiency of devices to forward multicast traffic in hardware. The work in [26] organizes forwarding entries of a switch based on prime numbers and the Chinese remainder theorem. It enables the switch
to reduce internal state and allows for more efficient hardware implementations. Reed et al. provide stateless multicast switching in SDN-based systems using Bloom filters in [27] and implement their approach for TCAM-based switches. The authors compare their approach with existing layer-2 forwarding methods and show significantly lower TCAM utilization for their method.

D. SDN Support for BIER

The authors of [28], [29] implement SDN-based multicast 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 bitstring, which limits the implementation to bitstrings with a length of 20 bits. Rules with exact matches for bitstrings are installed in the OpenFlow forwarding tables. When a packet with a BIER header arrives at a switch and a rule for its bitstring 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. Then, the local BIER agent installs a new flow entry for the specific bitstring in the OpenFlow forwarding table. Thus, this approach requires bitstring-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 [30]. It uses the same header format but features different forwarding semantics and is not compatible with normal BIER. In [31] we have proposed and evaluated several FRR algorithms for BIER-TE and implemented them in a P4-based prototype [4].

IV. BIT INDEX EXPLICIT REPLICATION (BIER)

In this section we introduce the concept of BIER and explain its forwarding procedure.

A. Layered Architecture

The layered architecture in Figure 1 explains the relation between BIER, IPMC, and the underlying network infrastructure.

![Figure 1: Layered architecture explaining the relation between IPMC, BIER, and the routing underlay.](image)

BIER is used as a transport mechanism for IPMC traffic, but in contrast to traditional IPMC, it does not require per-IPMC-group state in forwarding devices of core routers. IPMC traffic entering a BIER domain is passed to the BIER layer which delivers it in a multicast fashion to multiple egress nodes of the BIER domain. Thus, BIER serves as an automated point-to-multipoint tunnel whose receiver set depends on a bitstring of the BIER packet. BIER is a forwarding overlay on top of a routing underlay, e.g., an IP network. The routing underlay is used to determine the forwarding entries of BIER-capable devices. As a result, if IPMC traffic with a single destination is carried over BIER, the packets are carried over the same paths as unicast packets in the routing underlay.

B. BIER Concept

![Figure 2: BIER concept.](image)

Figure 2 illustrates a BIER domain. The forwarding devices of the BIER domain are called bit-forwarding routers (BFRs). Ingress and egress nodes of a BIER domain are denoted as bit-forwarding ingress and egress routers (BFIRs, BFERs). BFRs forward and replicate BIER packets according to a bitstring in the BIER header. Each bit position in the bitstring is associated with one BFER, and the bits activated in the bitstring indicate the BFRs that require a copy of the packet. The bit positions are numbered starting with 1 at the least-significant bit. Likewise, BFERs are numbered starting with 1. The BIER RFC [2] defines a minimum bitstring length of 64 bits for the encoding of BFRs. BFIRs push a BIER header onto IPMC packets entering the BIER domain and encode the set of receivers of a packet in the bitstring of its BIER header. The activated bits in the bitstring are incrementally reset during forwarding through the network. Before transmission over a specific interface, a BFR clears all bits in the header of a BIER packet that correspond to BFERs which are not reachable over that interface via the multicast subtree derived from the underlying routing. This helps to ensure that every packet is delivered exactly once to each BFER.

C. BIFT Structure & F-BM

A BFR has a Bit Index Forwarding Table (BIFT) to determine the next-hops (NHS) of a packet. It is a BFR neighbor (BFR-NBR). In Section IV-E we explain how BFR-NBRS are computed. An example of a BIFT is shown in Table 1. The BIFT contains one entry for each BFER. An entry consists of a NH and a forwarding bitmask (F-BM). The F-BM is a bitstring with the same structure as the bitstring in the BIER header and its activated bits indicate the set of BFERs reachable through the NH in the forwarding entry. The F-BM is used in the forwarding process to adapt the bitstring before transmission.


D. BIER Forwarding

When a BFR receives a BIER packet, it is processed in a loop. Within every iteration step, a copy of the BIER packet is sent and the bitstring of the original header is modified. The loop stops if the bitstring of the packet contains no more activated bits. At the beginning of a loop, the BFR determines the position of the least-significant activated bit of the bitstring. We call it the processed bit which corresponds to a BFER-ID. If there is no such bit, the loop stops.

The BFR looks up this BFER-ID in the BIFT, which results in a NH and a F-BM. Then, a copy of the BIER packet is created for transmission to the NH. Before transmission, all bits are cleared in the bitstring of the packet copy that are not reachable through the same NH. 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 NH.

All BFERs in the IPMC subtree of the NH eventually receive a copy from this sent packet if their BFER-IDs are activated in the bitstring. Thus, all BFERs of this IPMC subtree can be considered as processed. Therefore, their BFER-IDs are removed from the bitstring in the BIER header. To that end, the BFR applies the complement of the F-BM to the bitstring of the BIER header. This ensures that packets are delivered only once to intended receivers. Then, the loop restarts.

In the special case that the BFR is also a BFER, the processed bit may correspond to the BFER itself. Then, the BIFT entry is empty. As a result, 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 BIER header, and the loop restarts.

When BIER packets are forwarded over an outgoing interface, they consist of the IPMC packet and the BIER header. That means, they are carried directly to the NH without an additional header of the routing underlay. Typical lower-layer technologies for BIER are MPLS or Ethernet. However, BIER packets can also be tunneled over the routing underlay, e.g., IP. Then, they are equipped with an additional IP header before forwarding.

E. BIFT Computation

In a BIER network where all nodes are BIER-capable, IP neighbors are also BIER-NBRs. This is different in a hybrid BIER network where only some nodes are BIER-capable and an IP neighbor may not be a BIER-NBR. BIER-NBRs are derived as follows. Given a BFR, we consider the source tree to all BFERs in the BIER domain according to the routing underlay. All BFRs that can be reached on a path of this tree without running over another BFR are BFR-NBRs of the considered BFR. By this definition, a BFR may have BFR-NBRs that are not IP neighbors and are reachable only through a tunnel through the routing underlay.

The BIFT entries are computed as follows. For any BFER of the BIER domain, the NH is determined. It is the BFR-NBR through which the BFER can be reached along the shortest path in the routing underlay. Furthermore, the F-BM is composed of a set of activated bits. These bits correspond to the set of all BFERs that are reachable in the routing underlay through the same BFR-NBR. As a result all BIFT entries with the same NH also have the same F-BM.

F. BIER Example

Figure 3 shows an example topology with three BFRs. Each of them is both a BFIR and a BFER. Table 1 compiles the BIFT of BFR 1. BFR-NBR indicates the NH.

![BIER topology.](image)

Table 1: BIFT of BFR 1.

<table>
<thead>
<tr>
<th>BFER</th>
<th>F-BM</th>
<th>BFR-NBR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>010</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
<td>3</td>
</tr>
</tbody>
</table>

BFR 1 processes a BIER packet with the bitstring 110. The least-significant activated bit corresponds to BFR 2. BFR 1 copies the BIER packet and applies the F-BM to the copy. The result is a copy with the bitstring 010 which is then forwarded to the neighbor BFR 2. Afterwards, BFR 1 applies the complement of the F-BM to the remaining bits.

The remaining bitstring 100 still contains activated bits. The least-significant activated bit corresponds to BFR 3. A copy is created, the F-BM is applied, and the packet copy with the bitstring 100 is forwarded to neighbor BFR 3.

Since the remaining bitstring does not contain any activated bits anymore, the BIER forwarding procedure stops.

V. BIER Fast Reroute

The necessity for resilience mechanisms in BIER networks has been discussed in [32] without proposing any mechanism. In this section we introduce a method for BIER fast reroute (BIER-FRR) to protect multicast traffic against link and node failures by taking advantage of reconvergence and FRR mechanisms of the routing underlay. We first elaborate the resilience problem of BIER. Then, we explain the simple link protection mode of BIER-FRR, and afterwards the more complex node protection mode which requires changes to the BIFT. Finally, we discuss the protection properties of BIER-FRR.

A. The Resilience Problem of BIER

BFRs leverage the routing underlay to determine the NHs in their BIFTs. In the event of a failure, a NH may no longer be reachable and traffic transmitted to an unreachable NH is dropped. We distinguish between link and node failures.

1) Link Failures: In case of a link failure, the NH is no longer reachable. The lower-layer technology may protect the failure of that link and connectivity is quickly restored. MPLS with MPLS-FRR is an example for such a technology. If the network uses a lower-layer technology without protection features, e.g., Ethernet, the link failure persists and the routing underlay repairs the failure by updating its forwarding tables through a reconvergence process, which may take a few seconds. If the routing underlay utilizes FRR, connectivity is much faster restored in the order of 50 ms. However, BIER still
Forwards traffic over the failed link until affected entries in the BIFT are updated based on the reconverged forwarding entries of the routing underlay. This process takes necessarily longer than the reconvergence process of the routing underlay. This is a problem as all nodes of the IPMC subtree reachable through the unreachable NH cannot receive IPMC traffic during that time. The objective of BIER-FRR with link protection is to shorten this time such that BIER traffic can be forwarded again as soon as the routing underlay is able to forward unicast traffic after a link failure.

2) Node Failures: If a NH BFR fails, protection features of lower-layer technologies cannot help and BFERs in the affected IPMC subtree are only reachable when the BIFT is updated. We consider what happens in case of a node failure. Neighboring nodes may quickly detect the failure and locally reroute affected traffic using FRR mechanisms. From then on, unicast traffic can be again delivered in the routing underlay. Little later, the reconvergence process computes new forwarding entries so that unicast traffic can reach again all nodes (except for the failed one) via the standard entries of the forwarding tables. Only then, BFRs can update BIFT entries so that also BIER traffic can be delivered again. Thus, the outage of IPMC subtrees rooted at the failed node takes rather long in spite of FRR in the routing underlay or protection mechanisms of the lower-layer technology. The objective of BIER-FRR with node protection is to shorten this time so that affected BIER traffic can be delivered in the presence of node failures as soon as unicast traffic can be forwarded in the routing underlay.

B. BIER-FRR with Link Protection

To protect BIER against single link failures, we introduce BIER-FRR with link protection. BFRs may detect link failures through loss of light or through a Bidirectional Forwarding Detection (BFD) with the NH, which has recently been proposed for BIER [33]. When a BFR detects a link failure, it becomes the point of local repair (PLR). Then, it tunnels affected traffic to the unreachable NH through the routing underlay instead of forwarding BIER packets natively over the failed link. Thereby, the traffic reaches the NH again as soon as the routing underlay is able to deliver unicast traffic. The NH decapsulates the packet and processes the obtained BIER packet as usual. Thus, the tunnel effects that the packet is no longer carried over the failed interface but over a backup path which is utilized for unicast traffic. BIER traffic is tunneled as long as its NH is not reachable or until a new NH is computed based on the reconverged routing underlay.

C. BIER-FRR with Node Protection

With BFDs, a BFR can detect unreachable NHs, but it cannot differentiate between link and node failures. It becomes PLR and tunnels affected packets to relevant backup NHs. The set of backup NHs for a potentially failed NH consists of the NH itself and the set of all next-next hops (NNHs) BFRs on the downstream MC subtree routed at the potentially failed NH. The relevant backup NHs are those that should receive a packet copy either for further forwarding or for delivery to the IPMC layer. Before tunneling a packet copy to a relevant backup NH, the bitstring in the BIER header is modified with a backup F-BM to prevent that BFERs receive duplicate packets. To support this mechanism, the BIFT needs to provide backup NHs and backup F-BMs. In the following, we explain the necessary modifications to the BIFT. Afterwards, we illustrate BIER-FRR with node protection with an example and explain the computation of the backup NHs and backup F-BMs.

1) BIFT with Backup Entries: We explained the BIFT in Section IV-C. To facilitate BIER-FRR with node protection, the BIFT requires extensions for backup entries. The structure of a BIFT with backup entries is shown in Table 2.

Table 2: Structure of a BIFT with backup entries.

<table>
<thead>
<tr>
<th>BFER</th>
<th>F-BM</th>
<th>BFR-NBR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>primary F-BM</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>backup F-BM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>primary NH</td>
<td>backup NH</td>
</tr>
</tbody>
</table>

The normal BIFT entries are called primary entries. The backup entries have the same structure as the primary entries. When a NH is reachable, the primary entries are used for forwarding. If a NH becomes unreachable, the corresponding backup entry is used for forwarding in the same way as a primary entry. The difference is that a backup entry requires tunneling the BIER packet to the backup NH through the routing underlay instead of forwarding it natively over an interface.

2) Example: Figure 4 shows an example topology and Figure 5 illustrates the IPMC distribution tree for BFR 1 and BFR 2 based on the shortest paths of the routing underlay. Table 3 shows the BIFT of BFR 1 with primary and backup entries.

Table 3: BIFT of BFR 1 with primary and backup entries.

<table>
<thead>
<tr>
<th>BFER</th>
<th>F-BM</th>
<th>BFR-NBR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>000001</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>111010</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>000010</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>111010</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>111010</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>111010</td>
<td>6</td>
</tr>
</tbody>
</table>

We illustrate the forwarding with BIER-FRR when BFR 2 fails. We assume that BFR 1 needs to forward a BIER packet with bitstring 100000, i.e., the packet should be sent to BFR
6. As BFR 2 is unreachable, the primary NH of BFR 1 to BFR 6 cannot be used. Therefore, the backup entry is utilized. That means, the backup F-BM 101000 is applied to the copy of the BIER packet and the complement of the backup F-BM 010111 is applied to the bitstring of the BIER packet which is then 000000. Furthermore, the packet copy is tunneled through the routing underlay to the backup NH BFR 4. As the bitstring of the BIER packet no activated bits anymore, the forwarding process terminates.

3) Computation of Backup Entries: We distinguish two cases for the computation of backup entries for BFERs: (1) the considered BFER is the primary NH and (2) the considered BFER is not the primary NH.

In the first case, the primary NH is also taken as backup NH. This ensures that the NH receives a copy of the BIER packet if the reason for the unreachability of the NH is just a link failure. However, traffic to other nodes should not be forwarded via this unreachable NH. Therefore, the backup F-BM contains only the activated bit for the considered BFER.

In the second case, the BIER packet is tunneled to the NNH BFR on the IP shortest path towards the BFER, i.e., that NNH BFR is the backup NH. The backup F-BM requires the bits activated for all BFERs in the downstream IPMC subtree reachable from the PLR through the backup NH. This F-BM can be computed by bitwise AND’ing the PLR’s F-BM for the considered BFER and the failed NH’s F-BM for the considered BFER.

We illustrate both computation rules by an example. We consider the BIFT of BFR 1 in Table 3.

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

The backup entry of BFR 6 is an example for the second computation rule. The backup NH for BFR 6 is BFR 4 as it is the NNH of BFR 1 on the shortest path towards BFR 6 in Figure 5. The BFERs reachable from the PLR through BFR 4 are BFR 4 and BFR 6. Therefore, the backup F-BM is 101000. It can be obtained by bitwise AND’ing the F-BM of BFR 1 for BFR 6 (111010) and the F-BM of BFR 2 for BFR 6 (101100). The latter can be derived from the MC subtree of BFR 2 in Figure 5.

D. Protection Properties

Without BIER-FRR, the connectivity on the BIER layer may be restored in case of a link or node failure only after reconvergence of the routing underlay and the recomputation of the forwarding entries in the BIFTs. BIER-FRR restores the connectivity on the BIER layer as soon as unicast connectivity is restored on the routing underlay. This restoration also profits from FRR mechanisms in the routing underlay. BIER-FRR is a local mechanism without any need for signalling, possibly except for the detection of unreachable NHs, e.g., through BFDs. BIER-FRR comes with two protection modes. Link protection is simple but protects only against single link failures. Node protection is more complex, requires extensions to the BIFT, but protects against single link and node failures. Multiple component failures may still lead to packet loss.

In any failure scenario, the design of BIER-FRR prevents duplicates.

The proposed BIER-FRR method avoids loops on the BIER layer because the backup NHs are either identical with the primary NH or they are closer to destinations. If the backup NH is not reachable, traffic cannot be forwarded on the BIER layer, which prevents loops on the BIER layer. Nevertheless, a loop may still occur on the IP layer if the backup NH is not reachable and IP-FRR tries to deliver the packet. This, however, is a general problem of IP-FRR and not caused by BIER-FRR. Loop avoidance techniques have been proposed to solve this problem in SDN [34].

VI. INTRODUCTION TO P4

In this section we revise aspects of the programming language P4 needed for implementation of BIER(-FRR). 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.

A. P4 Pipeline

P4 is a high-level language for programming protocol-independent packet processors [35]. It introduces the forwarding pipeline shown in Figure 6.

![P4 pipeline](image)

Figure 6: P4 abstract forwarding pipeline according to [35].

A programmable parser reads packets and stores their header information in header fields which are carried together with the packet throughout the pipeline. The overall processing model is composed of two stages: the ingress and the egress pipeline with a packet buffer in between. The ingress port of a packet has to be specified in the ingress pipeline. If no egress port has been specified for a packet at the end of the egress pipeline, it 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.

B. 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 resemble packet-related variables in common programming languages. They may store arbitrary data, e.g., processing flags or calculated values. Each packet carries its own instances of standard and user-defined metadata through the P4 processing pipeline.
C. Match+Action Tables

P4 defines match+action tables. They are used within the ingress or egress pipeline to apply various actions and parameters to specified packets. The behaviour of a match+action table depends on rules whose structure is table-dependent and part of a P4 implementation. The rules themselves 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 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 7. The table has the name “Simplified IP unicast forwarding” and describes an implementation of simplified IP forwarding with match+action tables. It is simplified as we renounce on next hop indication.

![Match+action table for simplified IP forwarding](image)

**Figure 7: Match+action table for simplified IP forwarding.**

1) **Match Part:** A table defines a list of match keys that describe which header fields or metadata 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 corresponds to 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, the match key is the destination IP address and lpm matching will be applied.

2) **Actions:** The table further defines a list of actions including their signature. 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.

3) **Rules:** During runtime, the match+action tables can be filled through an application programming interface (API) with rules, 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 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 entries with different match fields and parameters can be installed by the controller.

In our example in Figure 7 we install two rules. The first matches on the IP address 192.168.0.1 and 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 second rule matches on 192.168.0.2 and 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 VII we omit the actual rules as they are configuration data and not part of the P4 implementation.

D. 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 also 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. They both are control blocks that possibly apply other control blocks. Examples of control blocks in our implementation are `IPv4`, `BIER`, or `Ethernet` handling in the ingress or egress pipeline.

E. Recirculation

P4 does not support native loops. However, as indicated in Figure 6, 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 control flow the packet is returned to the ingress control flow. 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 value. In contrast, header fields modified during the processing remain modified after recirculation. Another standard metadata field stores whether a packet has been recirculated.

F. Packet Cloning

P4 supports the packet cloning operation clone-ingress-to-egress (CI2E).

![Packet Cloning](image)

**Figure 8: Illustration of the clone-ingress-to-egress (CI2E) operation:** the IP number of the clone is the one of the original packet although its IP number was modified before CI2E was called.

CI2E can be called anywhere in the ingress control flow. This activates a metadata flag for cloning. 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 control flow. If a packet has been cloned 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 any header modification. Figure 8 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 CI2E is performed at the end of the ingress. There is a standard metadata field indicating whether a packet is a clone.

VII. P4-BASED IMPLEMENTATION OF BIER AND BIER-FRR

In this section, we describe a P4-based implementation of IP, IP-FRR, BIER, and BIER-FRR. We first describe the data plane and then the control plane. Finally, we give a pointer to the code base.

A. 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.

1) Packet Header Processing: P4 requires that potential headers of a packet are defined a priori. In our case, the supported header suite is Ethernet/outer-IP/BIER/inner-IP. Headers may be activated or deactivated. Deactivated headers will not be sent. 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.

2) Overview of Ingress and Egress Control Flow: Figure 9 shows an overview of the entire data plane implementation of our prototype which is able to perform IP and BIER forwarding as well as IP-FRR and BIER-FRR. It is divided into ingress and egress control flow which are given as control blocks. In the ingress and egress control block the IPv4 and BIER control block are only applied to their respective packets, i.e., the IPv4 control block is applied only to IP packets and the BIER control block is applied only to BIER packets. Here, we only summarize their operations and describe their implementation later.

When a packet enters the ingress pipeline, it is processed by the 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 IPv4 control block or the BIER control block is executed depending on the packet type.

The IPv4 control block pertains 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 BIER control block pertains to BIER packets. There is a BIER control block for the ingress control flow and another for the egress control flow. A processing loop is implemented which extends over both BIER control blocks. At the beginning of the processing loop in the ingress flow the bitstring of the BIER header 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 BIFT are applied to the packet. This also comprises BIER-FRR actions which encapsulate BIER packets with IP headers. 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 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 original BIER packet is just passed to the Ethernet control block.

The 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.

3) Port Status Control Block: The control block Port Status records whether a port is up or down in the user-defined metadata meta.live_ports of a packet. Figure 10 shows that it consists of only the match-action table Port_Status.

Figure 9: Overview of ingress and egress control flow.

Figure 10: In the control block Port Status the information about live ports is copied to the user-defined metadata field meta.live_ports of the packet.
The table does not define any match keys. As a result, the first entry always 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 bitstring 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 parameter live_ports in the table is updated by the controller when the port status changes, which will be explained in Section VII-B1c.

4) IPv4 Control Block: The IPv4 control block handles IPv4 packets. Its operation is shown in Figure [11] It leverages three match-action tables: IP_unicast, IPMC_native, and IPMC_BIER. Packets are processed by these tables depending on their type. IP_unicast performs IP unicast forwarding including IP-FRR. IPMC packets encounter a miss and are relayed by the control flow to IPMC_native or IPMC_BIER. IPMC_native performs native multicast forwarding for IPMC packets leaving the BIER domain while IPMC_BIER just adds a BIER header for IPMC packets entering the BIER domain.

a) 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 [7] but has more actions and implements IP-FRR. We first explain our implementation of 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 bitstring 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., $* \ldots *0* \ldots *$ or $* \ldots *1* \ldots *$. If FRR is desired for an IP prefix, two rules are provided: a primary rule with $* \ldots *1* \ldots *$ as required_port pattern, and a backup rule with $* \ldots *0* \ldots *$.

A different FRR approach for P4 has been proposed in [36]. 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 tables offers two actions: forward_IP and decaps_IP. The decaps_IP action is applied to the IP addresses of the node itself without FRR, i.e., the required_port pattern is set to $* \ldots *$ in such rules. Those IP packets are typically BIER packets that have been encapsulated in IP by 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 demo 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 a loop-free alternate (LFA), which is a simple IP-FRR method. In some scenarios there is no LFA available [37]. This problem is not specific to our implementation and there are ways to solve this problem with appropriate tunnels [34].

IPMC addresses encounter a miss in this table so that their packets are further treated by the control flow in the IPv4 control block. It checks whether the meta.BIER_decaps bit has been set. If so, the IPMC packet has been received from the BIER domain and decapsulated. Therefore, it is relayed to the 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 IPMC_BIER table.

b) IPMC_native: This match-action table implements native IPMC forwarding. It is used by a BIER to send IPMC packets to hosts outside the BIER domain that have subscribed to a specific IPMC group. The table 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

![Figure 11: In the IPv4 control block IPv4 packets are handled.](image-url)
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.

c) IPMC_BIER: This match+action table uses also the IP destination address as match key with an exact match. It defines only the *encaps_BIER* action and requires the bitstring 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 in the BIER header. This bitstring 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.

5) BIER Control Block: The BIER control block processes BIER packets. It is illustrated in Figure 12

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. When a BIER packet is processed by the BIER control block for the first time, *meta.remaining_bits* is initialized with the bitstring of the packet’s BIER header. 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 BIFT is applied. It implements BIER forwarding including BIER-FRR according to the principle we developed for IP-FRR in Section VII-A4a 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 bitstrings that we call *dest_BFER* and *required_port* (not shown in the table). The *dest_BFER* bitstring has the bit position for the respective BFER activated and all other bit positions set to wildcards (*...*1*...*). The *required_port* bitstring is used as in Section VII-A4a 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* bitstring 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* bitstring indicates that the 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 C12E is set. This effects that a packet copy is generated at the end of the ingress pipeline. For the this copy (clone), the *recirculate* flag is activated in the 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 BIFT table with

![Figure 12: The BIER control block in ingress and egress pipeline implements BIER forwarding within a processing loop.](image-url)
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.

![Ethernet Control Block](image1)

**Figure 13:** Ethernet control block.

6) **Ethernet Control Block:** The Ethernet control block is visualized in Figure [13]. It applies the match+action table 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.

**B. Control Plane Architecture**

The control plane is visualized in Figure [14]. It consists of one global controller 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. Keeping local tasks local through a 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.

In the following we explain the local and global controller in more detail.

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.

![Controller Architecture](image2)

**Figure 14:** Controller architecture.

Figure 14 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.

**a) 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 IPMC_native table, the local controller utilizes the Thrift channel instead of the P4 Runtime as this API is target-specific.

**b) 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 Ethernet which is utilized in the Ethernet control block (see Section VII-A6).

**c) 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 mechanisms. A local controller may test for neighbor reachability, e.g., using a BFD towards all neighbors. Then, the local controller configures this information as a bitstring in the match+action table Port_Status of the switch whenever this information changes.

In our demo, we use the software-based simple switch which is based on the BMv2 framework. The BMv2 framework propagates additional information on a so-called nanomsg channel. This additional information includes port status changes. As this is simpler than a BFD, the controller leverages that information to monitor neighbor reachability instead of a BFD.
2) Global Controller: We divide the architecture of the global controller in three layers: communication, service, and application (see Figure [14]).

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 message 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 BFIR, the global controller configures the IPMC_BIER table of all BFIRs with an appropriate bitstring for the specific IPMC group by activating or deactivating the corresponding bit of the BFER. As a result, the BIFR starts or stops sending traffic from this IPMC group to the BFER.

The global controller sets all entries in the IP_unicast and IPMC_BIER tables of all switches and the entries in the BIFTs. If the global controller is informed by a local controller about a failure, it first reconfigures the IP_unicast and IPMC_BIER tables and then the entries of the BIFTs accordingly.

C. Code Base

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 VIII-A. 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.

VIII. EVALUATION

In this section we illustrate that BIER traffic is better protected in BIER networks with BIER-FRR. To that end, we conduct experiments in a testbed using our prototype. We first explain the general testbed setup and the methodology. 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 show experimental protection results in an BIER/IP network with and without IP-FRR and BIER-FRR.

A. Experimental Setup and Methodology

We emulate the testbed depicted in Figure [15] in Mininet [38]. Two hosts $D$ and $S$ are connected to a BIER/IP network.

Figure 15: Two hosts $S$ and $D$ are connected to a BIER network with IP as the routing underlay.

The BFIRs are implemented by our P4-based prototype on the software switch BMv2 and we utilize a global controller to control their behavior through appropriate entries in match+action tables. Host $S$ sends every 30 ms packets to host $D$ over the BIER network. Every other packet is sent by IP unicast and IPMC. The BFIR adds a BIER header to the IPMC packet and the BFER removes it. Disconnecting the link between $PLR$ and $NH$ interrupts the packet delivery. We compare the time until host $D$ receives unicast and multicast traffic again with various combinations of IP reconvergence, BIFT recomputation, IP-FRR, and BIER-FRR.

B. 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 the software switch, failure detection is triggered by a software signal from the switch to the local controller right after disconnection of the link. The local controller informs the global controller to perform IP reconvergence and BIFT updates which can be executed in our virtual environment without any delay. This is different with routers and distributed routing protocols in the physical world. BFDs require some time to detect a failure. Then, routing protocols exchange information about the changed topology. Routers compute alternative routes and push them to their forwarding tables. When forwarding tables have been globally updated, routers can compute new forwarding entries for BIER and push them to their BIFTs. These actions require significant time. 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, their reaction does not need to be delayed.

C. Experiments

We perform experiments with and without IP-FRR and BIER-FRR and compare the time the network takes to deliver unicast and multicast traffic again after a failure.

1) Without IP-FRR and BIER-FRR: In the first experiment, only IP reconvergence and BIFT recomputation are enabled but IP-FRR and BIER-FRR are disabled. When the link between $PLR$ and $NH$ is disconnected in Figure [15] the switch $PLR$ is informed about that event at time $t = 0$. Figure [16(a)] shows signals at the switch $PLR$ (failure detection, update of IP forwarding entries, and update of BIFT entries) as well as unicast and multicast packets received by host $D$. Small bars
In this work, we proposed BIER-FRR to shorten the time of BIER’s disconnectivity after a failure. It deviates BIER traffic around the failure location via a tunnel through the routing underlay. This effects that BIER works again as soon as the routing underlay is able to forward traffic after a failure. The new method has a link and a node protection mode. While the link protection mode is simple but protects only link failures, the node protection mode protects also node failures but requires an extension to the regular 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. An predecessor of this prototype without BIER-FRR and based on P4_14 has been presented as a demo in [4].

We deployed our prototype on a virtualized testbed based on Mininet and the software switch BMv2. Experiments confirmed that BIER-FRR significantly reduces the time of a BIER network to deliver traffic again after a failure. Without BIER-FRR, BIER delivers traffic 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.

**IX. Conclusion**

BIER is a novel, scalable multicast method for IP networks without forwarding state per IP multicast (IPMC) group on core nodes. Only ingress nodes of a BIER domain keep group-specific information and push a BIER header including a bitstring on multicast traffic for simplified forwarding within the BIER domain. A bit index forwarding router (BFR) has a bit index forwarding table (BIFT) for forwarding decisions. Its entries are derived from a routing underlay, e.g., an IP network. In case of a failure, the BIFT is updated 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 routing underlay.

In this work, we proposed BIER-FRR to shorten the time of BIER’s disconnectivity after a failure. It deviates BIER traffic around the failure location via a tunnel through the routing underlay. This effects that BIER works again as soon as the routing underlay is able to forward traffic after a failure. The new method has a link and a node protection mode. While the link protection mode is simple but protects only link failures, the node protection mode protects also node failures but requires an extension to the regular BIFT structure.

are unicast traffic, medium bars are multicast traffic, and large bars are the signals. Unicast traffic is delivered again after IP reconvergence, which is completed about 200 ms after failure detection. The extra delay is due to software delay. Multicast traffic is delivered again after BIFT recomputation, which is completed only about 600 ms after failure detection.

2) With IP-FRR but without BIER-FRR: In the second experiment, IP-FRR is enabled but BIER-FRR remains disabled. Figure 16(b) shows that unicast traffic is delivered again little later after the failure occurs. In contrast, multicast traffic is delivered only 600 ms after the failure occurs, i.e., after BIFT updates which requires IP reconvergence before. Thus, BIER without BIER-FRR cannot profit from fast protection in the routing underlay.

3) Without IP-FRR but with BIER-FRR: In the third experiment, IP-FRR is disabled but BIER-FRR is enabled. Figure 16(c) shows that unicast traffic is delivered after IP reconvergence and the same holds for multicast traffic. The latter is due to BIER FRR as it effects that BIER packets are bypassed over IP tunnels around the failure location so
The BIER prototype demonstrates that P4 facilitates implementation of rather complex forwarding behavior. However, some workarounds were needed that point out limitations of P4 and some targets. It is desirable that link status (up/down) indication is available on the switch and does not need to be set by an external controller. 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 behavior 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.

REFERENCES