Hardware-Based Evaluation of Scalable and Resilient Multicast with BIER in P4

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

***Abstract*—Traditional IP multicast (IPMC) maintains state per IPMC group in core devices to distribute one-to-many traffic along tree-like structures through the network. This limits its scalability because whenever subscribers of IPMC groups change, forwarding state in the core network needs to be updated. Bit Index Explicit Replication (BIER) has been proposed by the IETF for efficient transport of IPMC traffic without the need of IPMC-group-dependent state in core devices. However, legacy devices do not offer the required features to implement BIER. P4 is a programming language which follows the software-defined networking (SDN) paradigm. It provides a programmable data plane by programming the packet processing pipeline of P4 devices. In this paper, we present the first hardware-based implementation of BIER and make the source code publicly available. Our hardware target is the high-performance P4 switching ASIC Tofino. The performance evaluation shows that BIER can be implemented with line rate forwarding on a high-performance, hardware-based P4 target. However, BIER processing requires packet recirculation which may decrease the throughput in case of insufficient recirculation capacity. To avoid this effect, physical ports can be turned into loopback mode and utilized for recirculation as well. We demonstrate that by experimental evaluation with our prototype and propose and validate a theoretical model for BIER throughput. We present a general model to provision recirculation ports on a P4 switch. We apply it to BIER to show that a very few physical recirculation ports suffice on a switch to support realistic traffic mixes.**

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

I. Introduction

IP multicast (IPMC) has been proposed to efficiently distribute one-to-many traffic, e.g. for IPTV, multicast VPN, commercial stock exchange, video services, public surveillance data distribution, emergency services, telemetry, or content-delivery networks, by forwarding only one packet per link. IPMC traffic is organized in IPMC groups which are subscribed by hosts. Figure 1 shows the concept of IPMC. IPMC traffic is forwarded on IPMC-group-specific distribution trees from the source to all subscribed hosts. To that end, core routers maintain forwarding state for each IPMC group to determine the next-hops (NHs) of an IPMC packet. Scalability issues are threefold. First, a significant amount of storage is required to keep extensive forwarding state. Second, when subscribers of an IPMC group change, the distribution tree needs to be updated by signaling the changes to core devices.

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

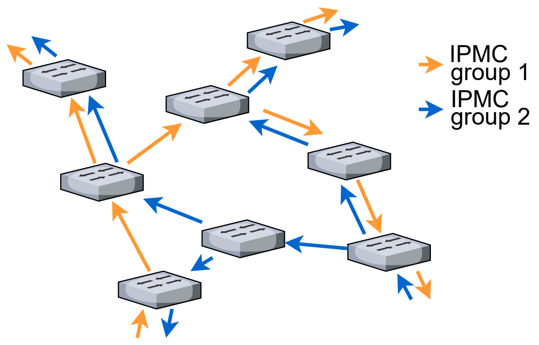

Figure 1: Two multicast distribution trees.

Third, the distribution trees have to be updated when the topology changes or a failure is detected. Therefore, traditional IPMC comes with significant management and state overhead.

The IETF proposed Bit Index Explicit Replication (BIER) [1] for efficient transport of IPMC traffic. BIER introduces a BIER domain where core routers do not need to maintain IPMC-group-dependent state. Upon entering the BIER domain, IPMC packets are equipped with a BIER header which specifies all destinations of the packet within the BIER domain. The BIER packets are forwarded through the BIER domain towards their destinations on paths from the Interior Gateway Protocol (IGP), which we call 'routing underlay' in the following. Thereby, only one packet is forwarded per link. When the BIER packets leave the BIER domain, the BIER header is removed.

Unicast and BIER traffic may be affected by failures. IP-Unicast traffic is often protected by IP fast reroute (IP-FRR) mechanisms. IP-FRR leverages precomputed backup entries to quickly reroute a packet on a backup path when the primary NH is unreachable. Tunnel-based BIER-FRR [2] is used to protect BIER traffic by tunneling BIER packets through the routing underlay. The tunnel may be also affected by a failure, but FRR or timely updates of the forwarding information base (FIB) in the routing underlay quickly restores connectivity. However, BIER is not supported by legacy devices and there is no dedicated BIER hardware available. P4 [3] is a programming language that follows the software-defined networking (SDN) paradigm for programming protocol-independent packet processors. P4 allows developers to write high-level programs to define the packet processing pipeline of network devices. A target-specific compiler translates the P4 program for execution on a particular device. With the P4-programmable data plane new protocols can be implemented and deployed in short time. In [2], [4] we presented a P4-based implementation of BIER and tunnel-based BIER-FRR for the P4 software switch BMv2 [5].

The contribution of this paper is threefold. First, we present the first implementation of BIER, and tunnel-based BIER-FRR for a P4 hardware platform, i.e., the P4 switching ASIC Tofino [6], and make it publicly available. The implementation of BIER in P4 requires that packets are recirculated so that they can be processed by the pipeline again. However, there is only limited capacity for traffic recirculation in P4 and throughput decreases if this limit is exceeded. We explain how we prevent this issue by leveraging so-called recirculation ports which increase the amount of traffic that can be recirculated. Second, we conduct a comprehensive hardware-based evaluation of BIER, and BIER-FRR which shows that BIER with line rate forwarding on a 100 Gb/s data center switch can be implemented in P4, and that BIER-FRR restores connectivity in case of failures with very little delay. Third, we present models to predict the throughput of recirculation traffic and derive the required number of physical ports that should be turned into loopback mode to support recirculation traffic. We model this problem for BIER traffic and a comparison with experimental results shows good accordance with the measured values. Finally, we utilize this model to show that only a few ports in loopback mode suffice to support realistic mixes of unicast and multicast traffic.

The paper is structured as follows. In Section II we describe related work. Section III contains a primer on BIER and tunnel-based BIER-FRR. Afterwards, we give an overview on P4 in Section IV. In Section V, we describe the P4 implementation of BIER and tunnel-based BIER-FRR for the Tofino. Section VI contains our evaluation and model derivation. We conclude the paper in Section VIII.

II. Related Work

First, we describe related work for SDN-based multicast in general. Afterwards, we review work for BIER-based multicast.

A. SDN-based Multicast

Two surveys [7], [8] provide a comprehensive overview of SDN-based multicast. They review the development of traditional multicast and different aspects of SDN-based multicast, e.g., building of distribution trees, group management, and approaches to improve the efficiency of multicast. Most of the papers in the surveys discuss multicast mechanisms that are based on explicit IPMC-group-dependent state in core devices. The papers often focus on intelligent tree building mechanisms that reduce the state, or efficient signaling techniques when IPMC-groups or the topology changes. Here, we mention only some papers that implement multicast for SDN. Other works can be found in the surveys.

1) Optimization of Multicast Trees: Rückert et al. propose Software-Defined Multicast (SDM) [9]. SDM is an OpenFlow-based platform that provides well-managed multicast for over-the-top and overlay-based live streaming services tailored for P2P-based video stream delivery. The authors extend SDM in [10] with traffic engineering capabilities. In [11] the authors propose address translation from the multicast address to the unicast address of receivers at the last multicast hop in OpenFlow switches. This reduces the number of IPMC-group-dependent forwarding entries in some nodes.

Steiner trees are often used to build multicast distribution trees [12]. Several papers modify the original Steiner-tree problem to build distribution trees with minimal cost [13], number of edges [14], number of branch nodes [15], delay [16], or for optimal position of the multicast source [17].

The authors of [18] implement a multicast platform in OpenFlow with a reduced number of forwarding entries. It is based on multiple shared trees between different IPMC groups. The Avalanche Routing Algorithm (AvRA) [19] considers properties of the topology of data center networks to build trees with optimal utilization of network links. Dual-Structure Multicast (DuSM) [20] leverages different forwarding structures for high-bandwidth and low-bandwidth flows. This improves scalability and link utilization of SDN-based data centers. Jia et. al. [21] present a way to efficiently organize forwarding entries based on prime numbers and the Chinese remainder theorem. This reduces the required state in forwarding devices and allows more efficient implementation. In [22] the authors propose a SDN-based multicast switching system that leverages bloom filters to reduce the number of TCAM-entries.

2) Resilience for Traditional Multicast: Shen et al. [23] modify Steiner trees to include recovery nodes in the multicast distribution tree. The recovery nodes cache IPMC traffic temporarily and resend it after reconvergence when the destination notified the recovery point because it did not get all packets due to a failure. The authors of [24] evaluate several algorithms that generate node-redundant multicast distribution trees. They analyse the number of forwarding entries and the effect of node failures. In [25] the authors propose to deploy primary and backup multicast trees in SDN networks. The header of multicast packets contains an ID that identifies the distribution tree on which the packet is forwarded. When a failure is detected, the controller reconfigures affected sources to send packets along a working backup tree. Pfeiffenberger et al. [26] propose a similar method. Each node that is part of a distribution tree is the root of a backup tree that does not contain the unreachable NH but all downstream destinations of the primary distribution tree. When a node cannot forward a packet, it reroutes the packet on a backup tree by switching an VLAN tag in the packet header.

B. BIER-based Multicast

In [27], [28] the authors implement explicit-state-based IPMC and BIER forwarding in OpenFlow. However, the BIER implementation suffers from two major shortcomings. First, the BIER bit string is encoded in an MPLS header which is the only way to encode arbitrary bit strings in OpenFlow. This limits the bit string length, and thus the number of receivers, to 20 which is the length of an MPLS label. Second, the implementation performs an exact match on the bitstring. If a subscriber changes, the match does not work anymore and a local BIER agent that is not part of the OpenFlow protocol needs to process the packet. The authors of [29] perform a simulation-based evaluation of BIER. They find that on metrics like delivery ratios and retransmissions BIER performs as well as traditional IPMC but has better link usage and no per-flow or per-group state in core devices.

Eckert et. al. [30] propose an extension for BIER that allows for traffic engineering (BIER-TE). In addition to the egress nodes, the BIER header encodes the distribution tree of a packet. In [31] the authors propose 1+1 protection for BIER-TE. The traffic is transported on two disjoint distribution trees, which delivers the traffic even if one tree is interrupted by a failure.

III. Bit Index Explicit Replication (BIER)

In this Section we explain BIER. First, we give an overview. Then we describe the BIER forwarding table and how BIER packets are processed. Afterwards, we show a forwarding example. Finally, we review tunnel-based BIER-FRR.

A. BIER Overview

First, we introduce the BIER domain. Then, we present the layered BIER architecture followed by the BIER header. Finally, we describe BIER forwarding.

1) BIER Domain: Figure 2 shows the concept of the BIER domain. When bit-forwarding ingress routers (BFIRs)

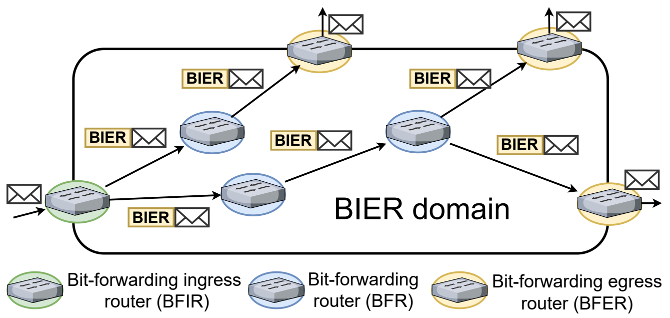
BIER
BIER
BIER
BIER
BIER
BIER
BIER domain
Bit-forwarding ingress router (BFIR)
Bit-forwarding router (BFR)
Bit-forwarding egress router (BFER)

Figure 2: The concept of the BIER domain.

receive an IPMC packet they push a BIER header onto it and forward the packet into the BIER domain. The BIER header identifies all destinations of the BIER packet within the BIER domain, i.e., bit-forwarding egress routers (BFERs). Bit-forwarding routers (BFRs) forward the BIER packets to all BFERs indicated in its BIER header. Thereby, packets are replicated and forwarded to multiple next-hops (NHs) but only one packet is sent over any involved link. The paths towards the destinations are provided by the Interior Gateway Protocol (IGP), i.e., the routing underlay. Therefore, from a specific BFIR to a specific BFER, the BIER packet follows the same path as unicast traffic. Finally, BFERs remove the BIER header.

2) The Layered BIER Architecture: The BIER architecture consists of three components. The IPMC layer, the BIER layer and the routing underlay. Figure 3 shows the three layers, their composition, and interaction. The IPMC layer contains

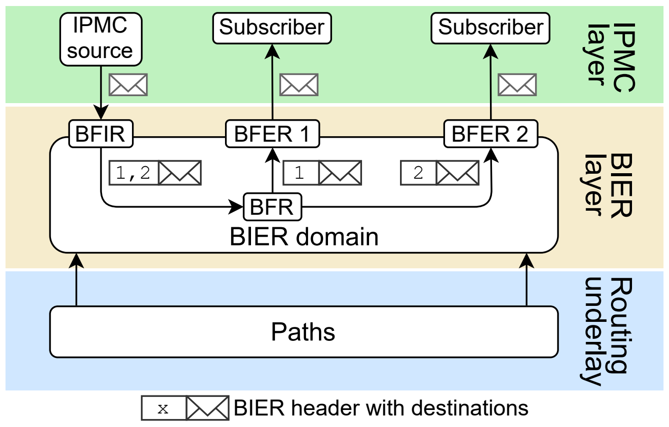

Figure 3: IPMC packets are transmitted over a layered BIER architecture; the paths are defined by the information from the routing underlay.

the sources and subscribers of IPMC traffic. The BIER layer acts as a transport layer for IPMC traffic. It consists of the BIER domain which is connected to the IPMC layer at the BFIRs, and BFERs. Therefore, the BIER layer acts as a point-to-multipoint tunnel from an IPMC source to multiple subscribers. The routing underlay refers to the IGP which provides the paths to all destinations within the network.

3) BIER Header: The BIER header contains a bit string to indicate the destinations of a BIER packet. To that end, each BFER is assigned an unique number that corresponds to a bit position in that bit string, starting by 1 for the least-significant bit. If a BFER should receive a copy of the IPMC packet, its bit is activated in the bit string in the BIER header of the packet. To facilitate readability we refer to the bit string in the BIER header of a BIER packet with the term 'BitString'.

4) BIER Forwarding: A BFR forwards a packet copy to any neighbor over which at least one destination of the packet indicated by its BitString is reached according to the paths from the routing underlay. Before a packet is forwarded to a specific NH, the BFR clears all bits that correspond to BFERs that are reached via other NHs from the BitString of that packet. This prevents duplicates at the BFERs.

B. BIFT Structure

BFRs use the Bit Index Forwarding Table (BIFT) to determine the NHs of a BIER packet. Table 1 shows the BIFT of BFR 1 from Figure 4. For each BFER there is one entry in the BIFT. Entries of the BIFT consist of a NH, and a so-called F-BM. The F-BM is a bit string similar to the BitString. It records which BFERs have the same NH. In the F-BM of an BIFT entry the bits of BFERs are activated which are reached over the NH of that entry. Therefore, BFERs with the same NH have the same F-BM. BFRs use the F-BM to clear bits from the BitString of a packet before it is forwarded to a NH.

BFER	NH	F-BM
1	-	-
2	2	1010
3	3	0100
4	2	1010

Table 1: BIFT of BFR 1 in the example of Figure 4.

C. BIER Packet Processing

When a BFR receives an BIER packet, it first stores the BitString of the packet in a separate bit string to account to which BFERs a packet has to be sent. In the following, we refer to that bit string with the term 'remaining bits'. The following procedure is repeated, until the remaining bits contain no activated bits anymore.

The BFR determines the least-significant activated bit in the remaining bits. The BFER that corresponds to that bit is used for a lookup in the BIFT. If a matching entry is found, it results in a NH nh and the F-BM fbm and the BFR creates a copy of the BIER packet. The BFR uses fbm to clear bits from the BitString of the packet copy. To that end, the BFR performs a bitwise AND operation of fbm and the BitString of the packet copy and writes the result into the BitString of the packet copy. This procedure is called applying the F-BM. It leaves only bits of BFERs in the BitString active that are reached over nh. The packet copy is then forwarded to nh. Afterwards, the bits of BFERs to which a packets has just been sent are cleared from the remaining bits. To that end, the BFR performs a bitwise AND operation of the bitwise complement of fbm with the remaining bits. The result is then stored in the remaining bits.

D. BIER Forwarding Example

Figure 4 shows a topology with four BIER devices where each is BFIR, BFR, and BFER. Table 1 shows the BIFT of BFR 1.

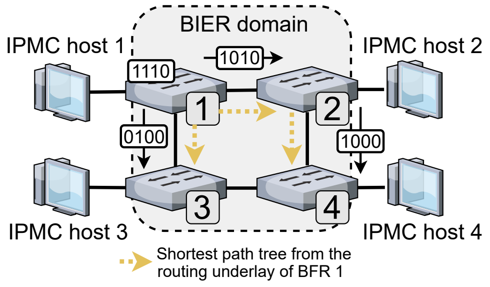

Figure 4: Example of a BIER topology and BitStrings of forwarded BIER packets.

BFR 1 receives an IPMC packet from IPMC host 1 which should be distributed to all other IPMC hosts. Therefore, BFIR 1 pushes a BIER header with the BitString `1110` to the IPMC packet.

Then, BFR 1 determines the least-significant activated bit in the BIER header which corresponds to BFER 2. This BFER is used for lookup in the BIFT, which results in the F-BM `1010` and the NH BFR 2. BFR 1 creates a packet copy and applies the F-BM to its BitString. Then, the packet copy with the BitString `1010` is forwarded to BFR 2. Finally, the activated bits of the F-BM are cleared from the remaining bits which leaves the bit string `0100`.

This leaves only one bit active which identifies BFER 3. After the F-BM `0100` is applied to the BitString of a packet copy, it is forwarded to BFR 3 with the BitString `0100`. After clearing the bits of the F-BM from the remaining bits, processing stops because no active bits remain.

E. Tunnel-Based BIER-FRR

Tunnel-based BIER-FRR is used to deliver BIER traffic even when NHs are unreachable due to link or node failures. When a BFR detects that a NH is unreachable, e.g., by loss-of-carrier, loss-of-light, or a bidirectional forwarding detection (BFD[1]) [32] for BIER [33], it becomes the point of local repair (PLR) by tunneling the BIER packet through the routing underlay to nodes downstream in the BIER distribution tree. The tunnel may be affected by the failure, too. However, FRR mechanisms or timely updates of the FIB in the routing underlay restore connectivity for unicast traffic faster than for BIER traffic because recomputation of BIER entries can start only after the FIB of the routing underlay has been updated. Tunnel-based BIER-FRR can be configured either for link protection or node protection. BIER-FRR with link protection tunnels the BIER packet to the NH where the tunnel header is removed and the BIER header is processed again. BIER-FRR with node protection tunnels copies of the BIER packets to all next-next-hops (NNHs) in the distribution tree.

IV. Introduction to P4

In this section we review fundamentals of P4. First, we give an overview and explain P4 architectures and P4 targets. Afterwards, we describe the P4 pipeline with its important components and operations.

A. Overview

P4 is a high-level programming language for protocol-independent packet processors [3]. A P4 program defines a data plane that is mapped onto the programmable pipeline of a P4 device by a target-specific compiler. A programmable (de)parser, and match+action-tables with arbitrary match fields and actions allow to use custom packets headers and program complex packet processing behavior. During runtime, the control plane configures the behavior of the P4 devices by modifying rules of the match+action tables. The first P4 specification $P4_{14}$ [34] has been published in 2014. $P4_{16}$ [35] from 2016 extends that specification by additional features.

B. P4 Architectures

A P4 architecture is a programming model that isolates the programmer from the low-level functionality of a platform. It provides a logical view of the hardware and its features. All P4 architectures implement the core functionality of P4 but

[1]When a BFR is established between two nodes, they periodically exchange notifications about their status.

often they describe interfaces to architecture-specific features. Therefore, P4 programs that are written for a specific architecture usually require adaptations to be compatible with other P4 architectures. The P4 language consortium develops the Portable Switch Architecture (PSA) [36] which is the reference architecture of P_{16}: 'the PSA is to the $P4_{16}$ language as the C standard library is to the C programming language' [36]. Tofino implements the Tofino Native Architecture (TNA). Due to NDA restrictions we cannot provide details about the TNA. However, the TNA is similar to the PSA to which we refer in the following explanations concerning P4.

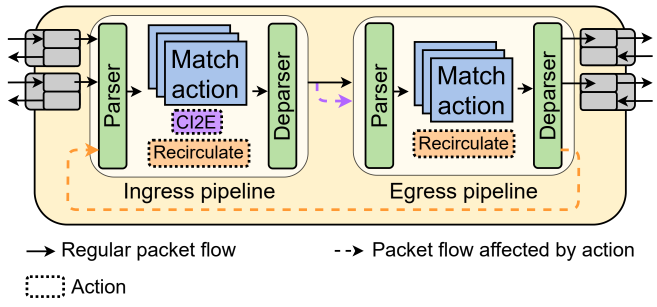

Figure 5: P4 processing pipeline.

C. Targets

A P4 target is a hardware or software platform that implements a particular P4 architecture. Software targets like the BMv2 [5] are implemented in high-level programming languages, e.g., C++, to emulate the behavior of a hardware platform. They can be easily adapted to implement even the most complex processing pipeline in P4. However, their throughput is limited because they run on non-specialized hardware and packet processing is done entirely in software. Therefore, they are not used in production environments but often for proof-of-concept implementations, and for validating and debugging P4 programs.

Hardware targets are P4 platforms that perform packet processing in hardware. Therefore, hardware targets efficiently achieve high throughput. However, programming for a hardware target is in general more challenging than for a software target. In contrast to software targets, the feature set of a hardware platform is limited, and therefore, implementation of complex packet processing may be challenging. Furthermore, they often come with additional constraints, e.g., the number of actions which can be performed on a packet, to guarantee certain performance aspects. There are multiple types of P4 hardware targets with different focus like FPGAs, network interface cards (NICs), and ASICs. Tofino [6] is the only hardware-based P4 programmable switching ASIC. It is used in the Edgecore Wedge 100BF-32X [37] switch for flexible high-speed packet processing.

D. P4 Pipeline

In this paragraph we review the P4 processing pipeline. We explain its composition, transient and persistent memory, match+action tables, control blocks, packet cloning and packet recirculation. Figure 5 shows the concept of the P4 processing pipeline for the PSA.

1) Composition: The P4 pipeline consists of an ingress pipeline and an egress pipeline. They process packets in a similar fashion, i.e., both contain a parser, a match+action pipeline, and a deparser. When a packet arrives at the switch, it is first processed by the ingress pipeline. The header fields of the packet are parsed and carried along with the packet through the ingress pipeline. The parser is followed by a match+action pipeline which consists of a sequence of conditional statements, table matches, and primitive operations. Afterwards, the packet is deparsed and sent to the egress pipeline for further processing. Finally, the packet is sent through the specified egress port which has to be set in the ingress pipeline and cannot be changed in the egress pipeline.

The P4 program defines the parser and the deparser, which allows the use of custom packet headers. In addition, the P4 program describes the control flow of the match+action pipeline in the ingress pipeline and egress pipeline, respectively.

2) Control Blocks: Both the ingress and egress pipeline can be divided into so-called control blocks for structuring. Control blocks are used to clearly separate functionality for different protocols like IP, BIER, and Ethernet, i.e., the IP control block contains MATs and operations that are applied only to IP packets, etc.

3) Transient and Persistent Memory: Transient memory is implemented by so-called metadata. Metadata can be compared to variables of other high-level programming languages. When a packet is processed in the P4 pipeline, it carries its own instances of metadata through the pipeline until the packet is sent. Then the metadata of that packet are discarded. Registers implement persistent memory to store information independently of packets.

4) Match+Action Tables (MATs): MATs execute packet-dependent actions by matching packet header fields and/or metadata against MAT entries. To that end, an entry contains one or more match fields, and an action set. When a packet is matched against a MAT, the match fields of the entries are compared with specified header or metadata fields of the packet. Each match field has one of four types: exact, longest-prefix match (lpm), ternary, and range. A match field of the type exact has to be exactly equal to header field to match. Lpm is well-known from regular IP matching. Ternary allows wildcard matching. The range type is used to check whether the matching value is in a certain interval. Then, the action set of the matching entry is executed. An action set consists of one or more actions, e.g., reading or writing a header or metadata field, mathematical operations, setting the egress port of the packet, etc. The P4 program defines only the structure of the MATs, i.e, the match fields and action sets. After startup the MATs are empty and need to be populated with entries by the control plane. It is not possible to match a packet on the same MAT multiple times.

5) Packet Cloning: The operation clone-ingress-to-egress (CI2E) allows packet replication in P4. It can be called only in the ingress pipeline. At the end of the ingress pipeline, a copy of the packet is created. However, the packet copy resembles the packet that has been parsed in the beginning of the ingress pipeline, i.e., the header changes performed during processing in the ingress pipeline are reverted. This is illustrated in Figure 6.

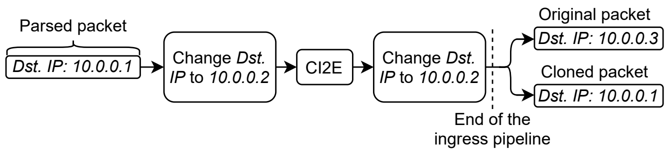

Figure 6: An example of the clone-ingress-to-egress (CI2E) operation.

If an egress port has been provided as a parameter, the egress port of the clone is set to that port. Both the original and cloned packet are processed independently in the egress pipeline. The cloned packet carries a metadata flag to identify it as a clone.

6) Packet Recirculation: P4 allows to recirculate a packet for processing it by the pipeline a second time. In [38] the authors implement a congestion control mechanism in P4 and leverage packet recirculation to create notification packets, update their header fields, and send them to appropriate monitoring nodes. The authors of [39] present a content-based publish/subscribe mechanism in P4 where they introduce a new header stack that requires packet recirculation for processing. Uddin et al. [40] implement multi-protocol edge switching for IoT based on P4. Packet recirculation is used to process packets a second time after they have been decrypted.

P4 leverages a switch-intern recirculation port for packet recirculation. When a packet should be recirculated, its egress port has to be set to the recirculation port during processing in the ingress pipeline. The flow of a packet through the pipeline when it is recirculated is shown in Figure 7. The packet is still

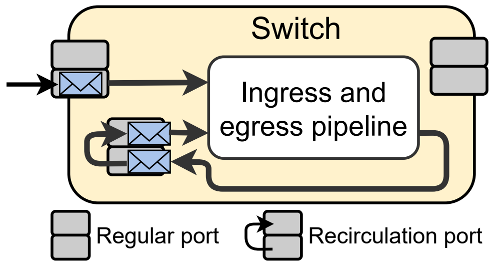

Figure 7: A packet is recirculated to a recirculation port and traverses the ingress and egress pipeline for a second time.

processed by the entire processing pipeline, i.e., the ingress pipeline and egress pipeline. However, after the packet has been deparsed, it is not sent through a regular physical egress port but pushed back into the switch-intern recirculation port. The packet is then processed as if it has been received on a physical port. The recirculation port has the same capacity as the physical ports. For example, when two physical ports receive traffic at line rate and each packet is recirculated once, the recirculation port receives recirculated packets at double line rate, which causes packet loss.

To discuss this effect we introduce the term ’recirculation capacity’. It is the available capacity to process recirculation traffic. Additional recirculation capacity is provided by using physical ports in loopback mode. When the forwarding device switches a packet to an egress port that is configured as a loopback port, the packet is immediately placed in the ingress of that port, instead. The packet is then processed as if it has been received on that port as usual, i.e., by the parser, the ingress and egress pipeline, and the deparser. Only traffic that has to be recirculated is switched to recirculation ports. In the following the term ’recirculation port’ refers to a physical port in loopback mode, or the switch-intern recirculation port. When recirculation ports are required, the switch-intern recircution port should be used first, before any physical ports are configured as loopback ports.

When multiple recirculation ports are deployed, we use a round-robin-based distribution approach for recirculation traffic to distribute the load equally over all recirculation ports. We store in a register which recirculation port receives the next packet which should be recirculated. When a packet has to be sent to a recirculation port, that register is accessed and updated in one atomic operation. This prevents any race conditions when traffic is distributed. Thus, the capacity of n recirculation ports for recirculation traffic is $n \cdot linerate$.

V. P4 Implementation of BIER and BIER-FRR for Tofino

In this section we describe the P4 implementation of BIER and tunnel-based BIER-FRR. First, we discuss the implementation basis. Afterwards, we give an overview of the implementation followed by details for both IP and BIER processing. Finally, we explain how port status monitoring is implemented.

A. Codebase

In [2] we presented a software-based prototype of a $P4_{16}$ implementation of BIER and tunnel-based BIER-FRR for the P4 software switch BMv2. We provided a very detailed description of the P4 programs including MATs with match fields and action parameters, metadata, control blocks, registers, and applied operations. The prototype and the controller are publicly available on GitHub[2]. The P4 implementation of BIER and tunnel-based BIER-FRR for the Tofino and the controller can be accessed on GitHub[3] by anyone.

B. Summary of the Processing Pipeline

In this paper we give only an abstract characterization of the control blocks for BIER and IP, and describe their application in the processing pipeline. P4-related implementation details about the control blocks can be found in [2].

This paragraph gives a high-level description of the entire processing pipeline including IP-unicast, IP-multicast, BIER, and Ethernet forwarding. Figure 8 shows an overview of the implemented pipeline.

The ethertype in the packet header is used to differentiate between BIER and IP packets. IP-unicast and IP-multicast

[2]https://github.com/uni-tue-kn/p4-bier
[3]https://github.com/uni-tue-kn/p4-bier-tofino

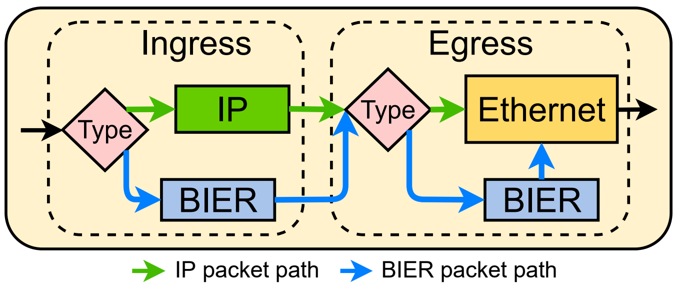

Figure 8: Overview of the implemented pipeline.

packets are handled by the single IP control block while BIER packets are handled by the BIER control blocks in the ingress and egress pipeline. The Ethernet control block changes the MAC addresses of the packets. The ingress pipeline contains an IP control block to determine the NH of IP packets. IP processing includes IP-unicast, where the NH of the packet is determined according to its destination IP-address, and IP-multicast whereby traditional multicast groups are used to forward the packets to the right NHs. The BIER control block in the ingress pipeline selects the proper NH and F-BM, computes the BitString and the remaining bits, and invokes cloning and recirculation of the BIER packet. In the egress pipeline, IP packets are handled by the Ethernet control block and sent afterwards. BIER packets are processed by a second BIER control block which performs header changes on cloned packets. Afterwards, the MAC addresses of BIER packets are changed by the Ethernet control block as well, before the packets are sent.

C. IP Processing

The IP control block processes both IP-unicast and IP-multicast packets. They are differentiated by their destination IP address.

1) IP-Unicast: The egress port of an IP packet is determined by its destination IP address. In addition, we implement an IP-FRR mechanism based on loop-free alternates (LFAs) [41] to bypass failures. When a device cannot forward a packet to the primary NH, it is forwarded to a backup NH that still has a working shortest path to the destination, instead. To that end, the switch monitors the status of all its ports. In Section V-E we explain how port monitoring is implemented. When an IP unicast packet is received by a node that is the egress node of an IP-tunnel, the IP header is removed and the packet is recirculated so that the packet is processed again.

2) IP-Multicast: P4 supports traditional IP multicast forwarding. For each multicast group, i.e., IP multicast address, a rule is configured in a forwarding table that stores to which neighbors the packet should be forwarded. When an IP packet matches the rule, the switch creates the right number of copies of the packet and sets the egress ports appropriately. This feature does not require recirculation or CI2E, but it comes with the previously discussed disadvantages of traditional IPMC. IP-multicast processing adds BIER headers to IPMC packets that should enter the BIER domain. Those packets are then recirculated for further processing.

D. BIER Processing

First, we describe regular BIER forwarding. Afterwards, we explain operation of tunnel-based BIER-FRR. We omit IP and Ethernet processing in the following descriptions to facilitate readability.

1) BIER Forwarding: Figure 9 shows the processing of a BIER packet during one pipeline iteration.

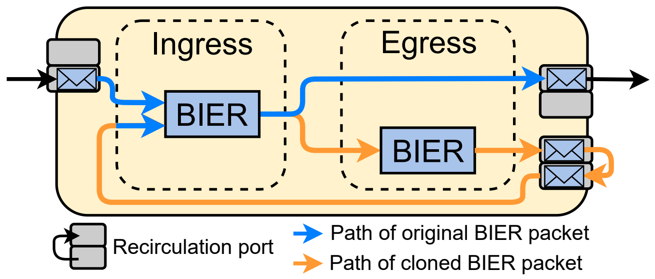

Figure 9: Paket flow of a BIER packet in the processing pipeline.

When the switch receives a BIER packet, it is processed by the BIER control block in the ingress pipeline. There, the BitString of the packet is matched against the BIFT which determines the egress port and the F-BM. The F-BM is applied to the BitString of the packet and cleared from the remaining bits. If the remaining bits still contain activated bits, CI2E is called and the egress port is set to a recirculation port so that the packet will be processed again. After the ingress pipeline, the copy is created and both packet instances enter the egress pipeline independently of each other. The original packet is sent through an egress port towards its NH. The packet clone is processed by a second BIER control block in the egress pipeline which sets the BitString of the packet copy to the remaining bits. Since the egress port of the packet clone is a recirculation port, the packet is recirculated, i.e., it is processed by the ingress pipeline again.

BIER forwarding removes BIER headers from packets that leave the BIER domain, and adds IP headers for tunneling through the routing underlay by tunnel-based BIER-FRR. Whenever a header is added or removed, the packet is recirculated for further processing.

When a BIER packet has more than one NH, two challenges appear. First, the BitString of a BIER packet has to be matched several times against the BIFT to determine all NHs. However, matching a packet multiple times against the same MAT is not possible in P4. Second, multiple packet copies have to be created for forwarding. However, P4 does not allow to dynamically generate more than one copy of a packet. Therefore, we implemented a packet processing behavior where in each pipeline iteration one packet is forwarded to a NH and a copy of the packet is recirculated for further processing. This is repeated until all NHs receive a packet over which at least one destination of the BIER packet is reached. Figure 10 shows the processing of a BIER packet which has to be forwarded to three neighbors. In the first and second pipeline iteration the original BIER packet is sent through a phyiscal egress port towards a NH and the copied BIER packet is

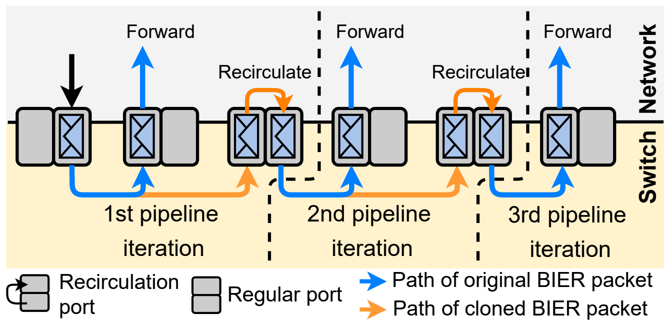

Figure 10: BIER processing over multiple pipeline iterations.

recirculated by sending the packet copy to a recirculation port. In the last iteration when the remaining bits contain no activated bits anymore, no further packet copy is required and only the original BIER packet is sent through the egress port. Therefore, to forward the packet to all of its three NHs, the packet is recirculated two times. In general, a BIER packet with n NHs, has to be recirculated $n-1$ times and the first NH can be served without packet recirculation.

2) Forwarding with Tunnel-Based BIER-FRR: The switch monitors the status of its ports as described in Section V-E. When the match on the BIFT results in a NH which is reached by a port that is currently down, the processing of the BIER packet differs in the following way from the BIER processing described above. An IP header is added to the original BIER packet to tunnel the packet through the routing underlay towards an appropriate node in the BIER distribution tree. The egress port of the original packet is set to a recirculation port to process the IP header in another pipeline iteration, i.e., forward the IP packet to the right NH.

E. Port Monitoring

FRR reroutes affected traffic on a backup path when the primary NH is not reachable. To determine which neighbor is reachable during forwarding, we leverage a register for each port. Each register stores one bit. If a port is up, the bit in the corresponding register is activated. If the port is down, the bit in its corresponding register is deactivated. After the NH of a packet has been determined, the corresponding register is examined to determine whether the port is up or down. If the port is up, the packet can be processed regularly, i.e., without IP-FRR and BIER-FRR. If the port is down, FRR actions are executed.

This approach requires reliable updates of the the port status bits when the port status changes, in particular, when a port goes down. To that end, we leverage the port monitoring of the Tofino. As soon as a port goes down, the Tofino generates a special packet which contains the port number of the port that went down, and places it in a switch-intern ingress port. When the packet is processed, the bit in the corresponding port status register is deactivated.

To activate a bit in a port status register we leverage a different feature. Our centralized control plane maintains a view of the entire topology to configure the P4 forwarding devices appropriately. To that end, it regularly broadcasts topology packets. As soon as the Tofino receives a topology packet on a port, we activate the corresponding bit in the port status register. Depending on the interval in which the topology packets are sent, this may introduce a brief delay until a port is recognized as up again, but this is not a time-critical operation. As soon as a port changes its status, the controller is notified. It updates its view on the topology, and calculates and installs new rules on all affected devices.

VI. Performance Evaluation of the P4-Based Hardware Prototype

In this section we perform experiments to evaluate the performance of the P4-based hardware prototype for BIER regarding Layer-2 throughput and failover time, i.e., the time until BIER traffic is successfully delivered after a network failure.

A. Failover Time for BIER Traffic

Here we evaluate the restoration time after a failure in three scenarios and vary the protection properties of IP and BIER. First, only the IP FIB and BIER FIB are updated by the controller, respectively, and no FRR mechanisms are activated. This process is triggered by a device that detects a failure. It notifies the controller which computes new forwarding rules and updates the IP and BIER FIB of affected devices. This scenario measures the time until the BIER FIB is updated after a failure, which is our baseline restoration time. The control plane, i.e., the controller, is directly connected to the P4 switch, which keeps the delay to a minimum in comparison to networks where the controller is several hops away.

Second, only BIER-FRR is deployed. In this scenario BIER is able to utilize tunnel-based BIER-FRR in case of a failure. However, FRR for IP traffic remains deactivated. Thus, IP traffic can be forwarded only after the IP FIB is updated. Third, both IP-FRR and BIER-FRR are deployed. This scenario evaluates how quickly the P4 switch can react to network failures and restore connectivity of BIER and IP forwarding.

In the following, we first explain the setup and the metric. Then, we present our results.

1) Experiment Setup: Figure 11 shows the testbed. The Tofino [6], a P4-programmable switching ASIC, is at the core of the hardware testbed. We utilize a Tofino based Edgecore Wedge 100BF-32X [37] switch with 32 100 Gb/s ports. An EXFO FTB-1 Pro [42] 100 Gb/s traffic generator is connected to the Tofino to generate a data stream that is as precise as possible. Furthermore, we deploy two BMv2s that act as BFRs and BFERs. The traffic generator, the controller and two BMv2s are connected to the Tofino. The traffic generator sends IPMC traffic to the Tofino. The IPMC traffic has been subscribed only by BMv2-1. As long as the link between the Tofino and BMv2-1 works, the BIER packets are forwarded on the primary path. When the Tofino detects a failure, it notifies the controller which computes new rules and updates forwarding entries of affected devices. In the meantime, the Tofino uses BIER-FRR to protect BIER traffic, and IP-FRR to protect IP traffic if enabled. This causes the Tofino to forward traffic on the backup path via BMv2-2 towards BMv2-1.

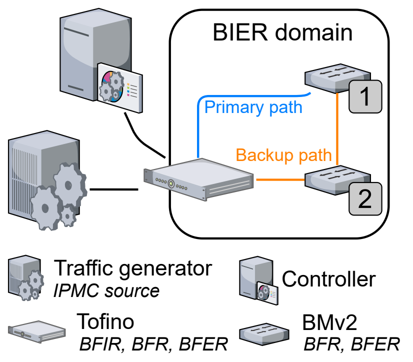

Figure 11: Experimental setup for evaluation of restoration time.

2) *Metric:* We disable the link between the Tofino and BMv2-1 and measure the time until BMv2-1 receives BIER traffic again. We evaluate different combinations with and without IP-FRR and with and without BIER-FRR. To avoid congestion on the BMv2s and the VMs, the traffic generator sends only with 100 Mb/s, which has no impact on the results.

Figure 12 shows the restoration time for the different deployed protection scenarios, which we discuss in the following.

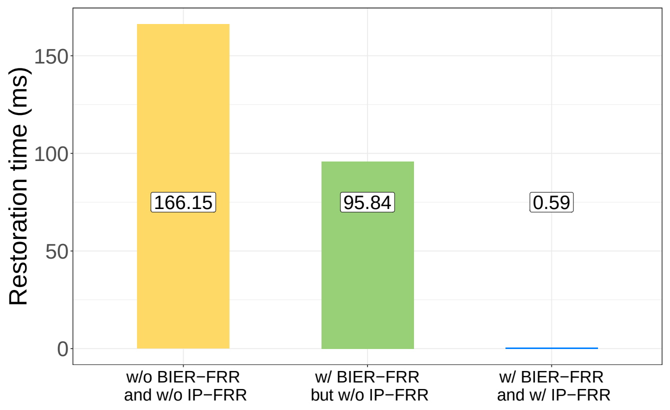

Figure 12: Restoration time of BIER w/o FRR, and BIER-FRR w/ and w/o IP-FRR.

3) *Failover Time w/o BIER-FRR and w/o IP-FRR:* When no FRR mechanism is activated, multicast traffic arrives at the host only after the IP and BIER forwarding rules have been updated, which takes about 166 ms. The controller is directly connected to the Tofino. In a real deployment the controller may be multiple hops away, which would increase the restoration time significantly.

The same failover time is achieved without BIER-FRR but with IP-FRR, for which we do not present separate results. As BIER forwarding entries are updated only after IP forwarding entries have been updated, the use of IP-FRR in the network does not shorten the failover time for BIER traffic.

4) *Failover Time w/ BIER-FRR but w/o IP-FRR:* When tunnel-based BIER-FRR but not IP-FRR is activated, BMv2-1 receives multicast traffic after 95 ms. In case of a failure, BIER-FRR tunnels the BIER traffic through the routing underlay. As soon as IP forwarding rules are updated, multicast traffic arrives at the host again. Since IP rules are updated faster than BIER rules, BIER-FRR decreases the restoration time for multicast traffic even if no IP-FRR mechanism is deployed.

5) *Failover Time w/ BIER-FRR and w/ IP-FRR:* In the fastest and most resilient deployment both BIER-FRR and IP-FRR are activated. Then, multicast packets arrive at the host with virtually no delay after only 0.5 ms. In contrast to the previous scenario, unicast traffic is rerouted by IP-FRR which immediately restores connectivity for IP traffic.

B. *Throughput for BIER Traffic*

The P4-based implementation of BIER described in Section V-D requires recirculation and is limited by the amount of recirculation capacity. The PSA defines a virtual port for this purpose. In this section we show the impact of insufficient recirculation capacity on throughput and the effect when additional physical recirculation ports, i.e., ports in loopback mode, are used for recirculation. We validate our experimental results in Section VI-C based on a theoretical model.

1) *Experimental Setup:* The experimental setup is illustrated in Figure 13. A source node sends IPMC traffic to

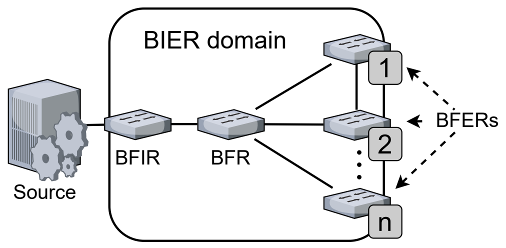

Figure 13: Theoretical setup for evaluation of BIER throughput.

a BFIR. The BFIR encapsulates that traffic and sends it to a BFR. The BFR forwards the traffic to n BFERs which decapsulate the BIER traffic and send it as normal IPMC traffic to connected subscribers.

The goal of the experiment is to evaluate the forwarding performance of the BFR depending on the number of NHs. With n NHs, BIER packets have to be recirculated $n-1$ times, and internal packet loss occurs if recirculation capacity does not suffice. The objective of the experiment is to measure the BIER throughput depending on the number of recirculation ports for which only physical loopback ports are utilized in the experiment. However, the n subscribers may see different throughput. The first BFER does not see any packet loss while the last BFER sees most packet loss. Therefore, we measure the rate of IPMC traffic received on Layer 2 at the last subscriber.

2) *Hardware Setup and Configuration:* Due to to hardware restrictions in our lab, we utilize one traffic generator, one P4-capable hardware switch, and one server running multiple

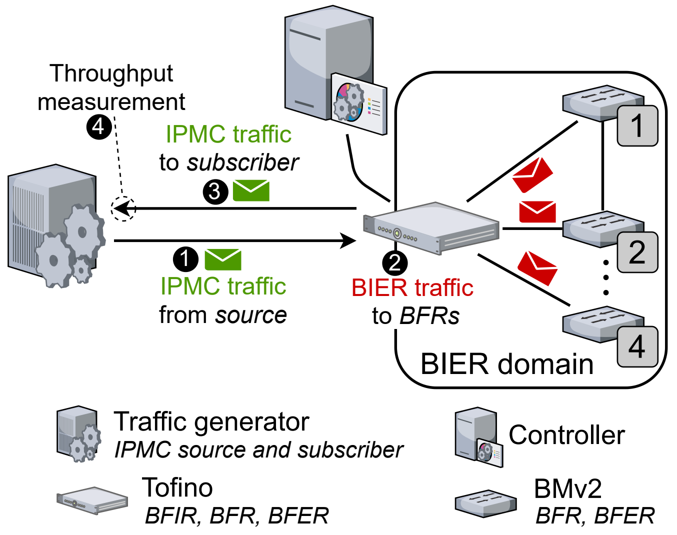

Figure 14: Hardware setup for evaluation of BIER throughput.

P4 software switches to build the logical setup sketched above. The hardware setup is shown in Figure 14. The traffic generator is the source of IPMC traffic and sends traffic to the BFIR. The traffic generator is also the subscriber of BFER n and measures the throughput of received IPMC traffic on Layer 2. The hardware switch acts as BFIR, BFR, and BFER n while BFERs 1 to $n-1$ are deployed as P4 software switches on the server. In addition, we collapse the BFIR and the BFR in the hardware switch so that packet forwarding from the BFIR to the BFR is not needed. Therefore, the traffic generator is the last NH of the BIER packet when it is processed by the BFR.

Packet recirculation is required after (1) encapsulation to enable further BIER processing, (2) decapsulation to enable further IP forwarding, and (3) BIER packet replication to enable BIER forwarding to additional NHs. We set up the hardware switch so that all recirculation operations in connection with encapsulation and decapsulation are supported by two dedicated ports in loopback mode and spend another k ports in loopback mode to support packet recirculation after packet replication. This models the competition for recirculation ports on a mere BFR as in the theoretical model.

The P4 software switches are BMv2s that run alongside our controller on VMs on a server with an Intel Xeon Scalable Gold 6134 (8x 3.2 GHz) and 4 x 32 GB RAM. The P4 hardware switch is a Tofino [6] inside an Edgecore Wedge 100BF-32X [37] which is a 100 Gb/s P4-programmable switch with 32 ports. The traffic generator is an EXFO FTB-1 Pro [42] which generates up to 100 Gb/s. All devices are connected with QSFP28 cables which transmit up to 100 Gb/s.

3) BIER Throughput Depending on Recirculation Ports: The traffic generator sends IPMC traffic at a rate of 100 Gb/s to the hardware switch, the hardware switch encapsulates the IPMC traffic, forwards BIER traffic iteratively n-1 times to BMv2s, recirculates the BIER packet to process the last activated header bit, decapsulates the traffic as BFER n, and returns it back to the traffic generator, which measures the received IPMC rate on Layer 2. These results are compiled in Fig. 15. We consider 1, 2, 3, and 4 NHs and perform these experiments with 1, 2, and 3 ports in loopback mode to support recirculation for BIER forwarding. With a single recirculation

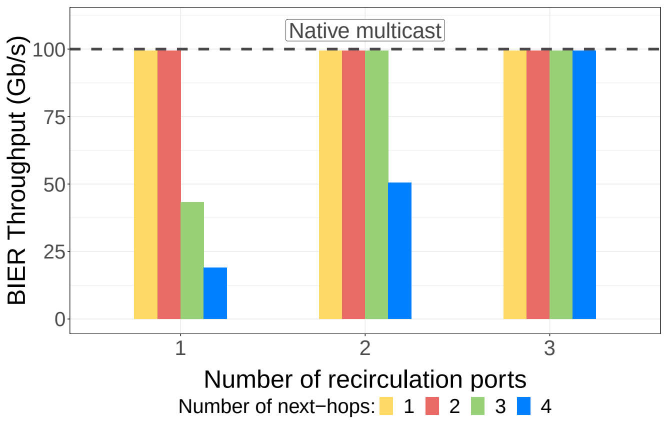

Figure 15: Experimental throughput of BIER and traditional IPMC on the 100 Gb/s Tofino-based switch for different numbers of NHs and recirculation ports.

port, the last NH receives the full IPMC rate of 100 Gb/s if at most 2 NHs are connected. For 3 or 4 NHs, the IPMC traffic rate received by the last NH is reduced to 43 and 19 Gb/s, respectively.

With 2 recirculation ports, the last NH does not perceive a throughput degradation if at most 3 NHs are connected. For 4 NHs, the IPMC traffic rate received by the last NH is reduced to 50 Gb/s.

And with 3 recirculation ports, even 4 NHs can be supported without throughput degradation for the last NH.

Thus our experiments confirm that when multicast traffic arrives with 100 Gb/s at the Tofino, n-1 recirculation ports are needed to forward BIER traffic to n NHs without packet loss. This is different for a realistic multicast portion in the traffic mix, i.e., a minor fraction instead of 100%.

The hardware switch also supports traditional multicast in P4. With traditional multicast forwarding, all NHs receive 100 Gb/s regardless of the number of NHs. However, this comes with all the disadvantages of traditional IPMC we have discussed earlier.

C. Throughput Model for BIER Forwarding with Insufficient Recirculation Capacity

We model the throughput of BIER forwarding with insufficient recirculation capacity and validate the results with the experimentally measured values.

To forward a BIER packet to n NHs, it has to be recirculated $n-1$ times (see Section V-D). Any time a packet is sent to a recirculation, the packet is dropped with a certain probability if insufficient recirculation capacity is available. Due to the implemented round robin approach (see Section IV-D6), the drop probability p is equal for all recirculation ports. The drop probability p in a system can be determined by comparing the available recirculation capacity and the sustainable recirculation load. The latter results from recirculations after BIER packet replication and takes packet loss into account. It is shown in the following formula.

$$C \cdot \sum_{m=1}^{n-1} (1-p)^m = k \cdot C \quad (1)$$

The available recirculation capacity is $k \cdot C$ where k is the number of recirculation ports and C is line capacity. The sustainable recirculation load is the sum of the successfully recirculated traffic rates after any number of recirculations. The traffic amount that has been successfully recirculated once is $C \cdot (1-p)$. The traffic amount that has been recirculated twice is $C \cdot (1-p)^2$, and so on. Therefore, the total amount is $C \cdot \sum_{m=1}^{n-1}(1-p)^m$.

We calculate the BIER throughput at any NH, i.e., after any number of recirculations. At the first NH, the throughput of the BIER traffic is C because the BIER packet is forwarded to the first NH before the packet is recirculated the first time. At the second NH, the BIER throughput is $C \cdot (1-p)$, at the third NH its $C \cdot (1-p)^2$, and so on. Therefore, the BIER throughput $T(i)$ at NH $1 \leq i \leq n$ is:

$$T(i) = C \cdot (1-p)^{i-1} \quad (2)$$

Table 2 shows the throughput predictions $T(i)$, and measured values $M(i)$ from the performance evaluation (see Section VI-B3). We make predictions for the same scenarios as we evaluated in the performance evaluation (see Section VI-B3). The comparison shows that the model provides reasonable predictions for the BIER throughput.

VII. Provisioning Rule for Recirculation Ports

In this section we propose a provisioning rule for recirculation ports. It may be used for general P4-based applications requiring packet recirculation, not just for BIER forwarding. We first point out the importance for sufficient recirculation capacity. Then, we derive a general provisioning rule for recirculation ports and illustrate how their number depends on other factors. Finally, we apply that rule to provision the number of loopback ports for BFRs in the presence of traffic mixes.

A. Impact of Packet Loss due to Missing Recirculation Capacity

If recirculation capacity does not suffice and packets need to be recirculated several times, packet loss observed at the last stage may be quite high. We first illustrate this effect. If the packet loss probability due to missing recirculation capacity is p, then the overall packet loss probability after n recirculations is $p(n) = 1-(1-p)^n$. We illustrate this connection in Figure 16, which utilizes logarithmic scales to better view several orders of magnitude in packet loss. With only one recirculation, we obtain a diagonal for the overall packet loss. A fixed number of recirculations shifts the entire curve upwards, and with several recirculations like $n = 6$ or $n = 10$, the overall loss probability $p(6)$ or $p(10)$ is an order of magnitude larger than the packet loss probability p of a single recirculation step. Therefore, avoiding packet loss

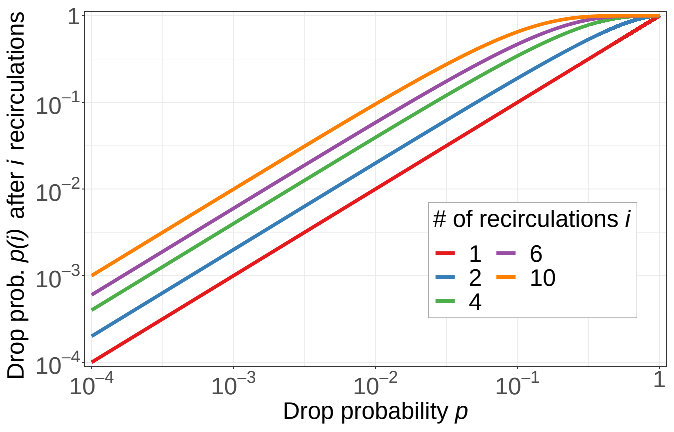

Figure 16: Loss probability after multiple recirculations.

due to recirculations is important. Thus, sufficient recirculation capacity must be provisioned but overprovisioning is also costly since this means that entire ports at high speed cannot be utilized for operational traffic. Therefore, well-informed provisioning of recirculation ports is an important issue.

B. Derivation of a Provisioning Rule for Recirculation Ports

We first introduce the recirculation factor R and the utilization ratio U. Then, we use them to derive a provisioning rule for recirculation ports.

The recirculation factor R is the average number of recirculations per packet. Not all packets may be recirculated or the number how often a packet is recirculated depends on the particular packet.

The utilization ratio U describes the multiple by which a recirculation port can be higher utilized than a normal port. For example, if the average utilization of each normal ports is 10%, the each recirculation port may be operated with a utilization of 40%, in particular if multiple of them are utilized. This corresponds to a utilization ratio of $U = 4$. We give some rationales for that idea. Normal ports at high speed are often underutilized in practice because bandwidths exist only in fixed granularities and usually link speeds are heavily overprovisioned to avoid upgrades in the near future. Furthermore, some links operate at lower utilization, others at higher utilization. Recirculation ports can be utilized to a higher degree. First, there is no need to keep the utilization of recirculation ports low for reasons like missing appropriate lower link speeds as it can be the case for normal ports. Second, recirculation ports are shared for all recirculation traffic of a switch so that resulting traffic fluctuations are lower and the utilization of the ports can be higher than the one of other ports.

If m incoming ports carry traffic with a recirculation factor R and a utilization ratio U can be used on the switch, then

$$m' = \left\lceil \frac{m \cdot R}{U} \right\rceil \quad (3)$$

describes the number of required recirculation ports.

	1 NH		2 NHs		3 NHs		4 NHs	
Recirculation ports:	$T(1)$	$M(1)$	$T(2)$	$M(2)$	$T(3)$	$M(3)$	$T(4)$	$M(4)$
1	100	99.32	100	99.32	38.2	43.3	15.74	19
2	100	99.32	100	99.32	100	99.32	53.14	50.5
3	100	99.32	100	99.32	100	99.32	100	99.32

Table 2: Model predictions for BIER throughput and measured values (Gb/s).

C. Illustration of Required Recirculation Ports

For illustration purposes, we consider a P4 switch with 32 physical (external) ports and one virtual (internal) port in loopback mode for recirculations. If the capacity of that single virtual recirculation port does not suffice for recirculations, physical ports need to be turned into loopback mode as well and be used for recirculation. All recirculation ports are utilized in round-robin manner to ensure equal utilization among them.

Thus, the number of normal ports m plus the number of recirculation ports m' must be at most 33, i.e., 32 physical ports and 1 virtual port. Therefore, we find the smallest m' according to Equation 3, so that $m + m' \leq 33$ while maximizing m. The number of physical recirculation ports is $m' - 1$ as the virtual port can also be used for recirculations. Figure 17 shows the number of physical recirculation ports depending on the recirculation factor R and the utilization ratio U. Thereby, R and U are fractional numbers. While the number of recirculations for each packet is an integral number, the average number of recirculations per packet R is fractional. The number of physical recirculation ports increases with the

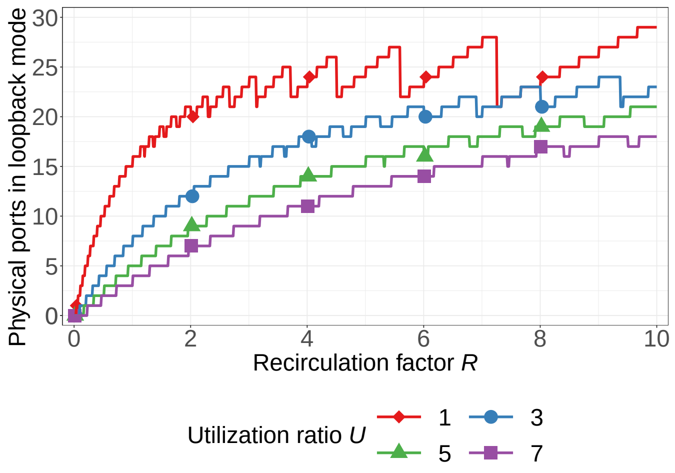

Figure 17: Number of physical ports in loopback mode.

recirculation factor R. Due to the fact that both m and m' are integers, the number of physical recirculation ports $(m-1)$ is not monotonously increasing because for some R and U the sum $m+m'$ amounts to the maximum 33, and to lower values for other R and U.

The various curves show that the number of required physical recirculation ports decreases with increasing utilization ratio U. With a large recirculation factor $R \geq 3$ and a low utilization Ratio $U \leq 3$, half of the ports of the 32 port switch or even more need to be used for recirculation, which is expensive. However, with small $R < 1$ and large $U > 3$ the number of required physical recirculation ports is low because most of the traffic does not require packet recirculation, and due to the large utilization ratio U, the recirculation ports can cover significantly more traffic than normal ports. It is even possible that no physical recirculation port is needed if the recirculation capacity of the internal recirculation port can cover the recirculation load.

D. Application of the Provisioning Method to Traffic Mixes with BIER

In this section we make predictions for m', the number of recirculation ports, for traffic mixes with typical multicast portions. We assume different portions of multicast traffic $a \in \{0.01, 0.025, 0.05, 0.1, 0.2\}$ and different average numbers of BIER NHs $n \in \{0, 2, 4, ..., 16\}$, i.e., each BIER packet is recirculated $n - 1$ times on average. Then, we calculate $R = a \cdot (n - 1)$, and assume $U = 4$. Again, we calculate the smallest m', i.e., like in Equation 3, so that $m + m' \leq 33$ while maximizing m. Figure 18 shows the number of physical recirculation ports depending on the average number of multicast NHs n and the fraction of multicast traffic a. If the fraction of multicast traffic is low

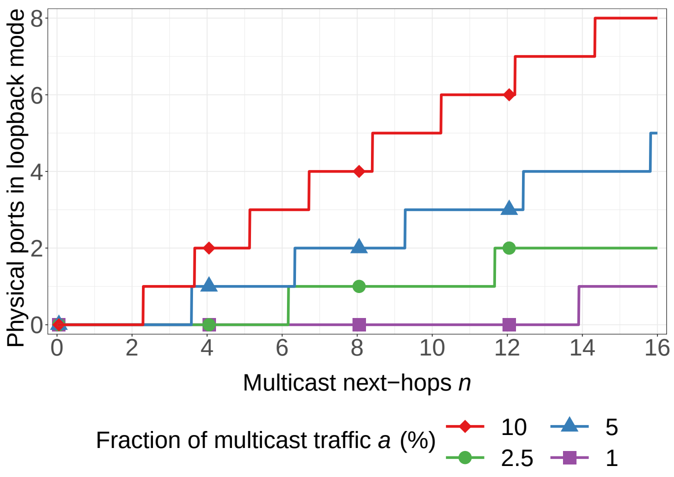

Figure 18: Physical ports in loopback mode for traffic mixes with realistic multicast portions.

like 1%, the capacity of the internal port suffices to serve up to 13 NHs on average. Moderate fractions of 2.5% multicast traffic require no physical recirculation port for up to 5 NHs, 1 physical recirculation port for up to 11 NHs, and 2 physical recirculation ports for 12 and more NHs. With 5% multicast traffic, the number of required physical recirculation ports increases almost linearly from zero to 5 with an increasing number of NHs. Large fractions of multicast traffic, like 10%, require up to 8 recirculation ports if the number of NHs is also large like 16. Under such conditions, 25% of the physical ports cannot be used for normal traffic forwarding as they are turned into loopback mode. However, the assumptions seem rather unlikely as multicast traffic typically makes up only a small proportion of the traffic.

VIII. Conclusion

The scalability of traditional IPMC is limited because core devices need to maintain IPMC group-dependent forwarding state and process lots of control traffic whenever topology or subscriptions change. Therefore, BIER has been introduced by the IETF as an efficient transport mechanism for IPMC traffic. State in BIER core devices is independent of IPMC groups, and control traffic is only sent to border nodes, which increases scalability in comparison to traditional IPMC significantly. In addition, there are fast-reroute (FRR) mechanisms for BIER to minimize the effect of network failures. However, BIER cannot be configured on legacy devices as it implements a new protocol with a complex forwarding behavior.

In this paper we demonstrated a P4-based implementation of BIER with tunnel-based BIER-FRR, IP unicast with FRR, IP multicast, and Ethernet forwarding on existing P4-enabled hardware. The target platform is the P4-programmable switching ASIC Tofino which is used in the Edgecore Wedge 100BF-32X, a 32 100 Gb/s port high-performance P4 switch. To implement BIER forwarding, the implementation requires multiple packet recirculations, which may limit the throughput of BIER traffic if the switch-intern recirculation port is overloaded. As a remedy, more recirculation capacity can be added by turning physical ports into loopback mode.

We evaluated the performance of the prototypical hardware implementation. On the one hand, we showed that BIER-FRR significantly reduces the restoration time after a failure, and in combination with IP-FRR, the restoration time is extremely reduced. We confirmed that the number of recirculation ports may limit BIER throughput, depending on the number of BIER next-hops, next-hops and port utilization. We modelled BIER forwarding, predicted limited throughput due to missing recirculation capacity, and validated the results by experimental values. Furthermore, we proposed a simple method for the provisioning of physical recirculation ports. We applied that method to different traffic mixes including multicast traffic and showed that only very few physical recirculation ports already suffice to handle realistic loads of multicast traffic with BIER.

References

[1] I. Wijnands *et al.*, *RFC 8279: Multicast Using Bit Index Explicit Replication (BIER)*, https://datatracker.ietf.org/doc/rfc8279/, Nov. 2017.

[2] D. Merling *et al.*, "P4-Based Implementation of BIER and BIER-FRR for Scalable and Resilient Multicast," *Journal of Network and Computer Applications (JNCA)*, 2020, Under submission. [Online]. Available: https://atlas.informatik.uni-tuebingen.de/~menth/papers/Menth19-Sub-1.pdf.

[3] P. Bosshart *et al.*, "P4: Programming Protocol-Independent Packet Processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, 2014.

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

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

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

[7] S. Islam *et al.*, "A Survey on Multicasting in Software-Defined Networking," *IEEE Communications Surveys Tutorials (Comst)*, vol. 20, 2018.

[8] Z. Al-Saeed *et al.*, "Multicasting in Software Defined Networks: A Comprehensive Survey," *Journal of Network and Computer Applications (JNCA)*, vol. 104, 2018.

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

[10] J. Rueckert *et al.*, "Flexible, Efficient, and Scalable Software-Defined Over-the-Top Multicast for ISP Environments With DynSdm," *IEEE Transactions on Network and Service Management (TNSM)*, vol. 13, 2016.

[11] T. Humernbrum *et al.*, "Towards Efficient Multicast Communication in Software-Defined Networks," in *IEEE International Conference on Distributed Computing Systems Workshops*, 2016.

[12] C. A. S. Oliveira *et al.*, "Steiner Trees and Multicast," *Mathematical Aspects of Network Routing Optimization*, vol. 53, 2011.

[13] L. H. Huang *et al.*, "Scalable and Bandwidth-Efficient Multicast for Software-Defined Networks," in *IEEE GLOBECOM*, 2014.

[14] Z. Hu *et al.*, "Multicast Routing with Uncertain Sources in Software-Defined Network," in *IEEE/ACM International Symposium on Quality of Service*, 2016.

[15] S. Zhou *et al.*, "Cost-Efficient and Scalable Multicast Tree in Software Defined Networking," in *Algorithms and Architectures for Parallel Processing*, 2015.

[16] J.-R. Jiang *et al.*, "Constructing Multiple Steiner Trees for Software-Defined Networking Multicast," in *Conference on Future Internet Technologies*, 2016.

[17] B. Ren *et al.*, "The Packing Problem of Uncertain Multicasts," *Concurrency and Computation: Practice and Experience*, vol. 29, 2017.

[18] Y.-D. Lin *et al.*, "Scalable Multicasting with Multiple Shared Trees in Software Defined Networking," *Journal of Network and Computer Applications (JNCA)*, vol. 78, 2017.

[19] A. Iyer *et al.*, "Avalanche: Data Center Multicast using Software Defined Networking," in *International Conference on Communication Systems and Networks*, 2014.

[20] W. Cui *et al.*, “Scalable and Load-Balanced Data Center Multicast,” in *IEEE GLOBECOM*, 2015.

[21] W. K. Jia *et al.*, “A Unified Unicast and Multicast Routing and Forwarding Algorithm for Software-Defined Datacenter Networks,” *IEEE Journal on Selected Areas in Communications*, vol. 31, 2013.

[22] M. J. Reed *et al.*, “Stateless Multicast Switching in Software Defined Networks,” in *IEEE International Conference on Communications*, 2016.

[23] S. H. Shen *et al.*, in *Reliable Multicast Routing for Software-Defined Networks*, 2015.

[24] M. Popovic *et al.*, “Performance Comparison of Node-Redundant Multicast Distribution Trees in SDN Networks,” *International Conference on Networked Systems*, 2017.

[25] D. Kotani *et al.*, “A Multicast Tree Management Method Supporting Fast Failure Recovery and Dynamic Group Membership Changes in OpenFlow Networks,” *Journal of Information Processing (JIP)*, vol. 24, 2016.

[26] T. Pfeiffenberger *et al.*, “Reliable and Flexible Communications for Power Systems: Fault-tolerant Multicast with SDN/OpenFlow,” in *IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, 2015.

[27] A. Giorgetti *et al.*, “First Demonstration of SDN-based Bit Index Explicit Replication (BIER) Multicasting,” in *IEEE European Conference on Networks and Communications (EuCNC)*, 2017.

[28] ——, “Bit Index Explicit Replication (BIER) Multicasting in Transport Networks,” in *International Conference on Optical Network Design and Modeling (ONDM)*, 2017.

[29] Y. Desmouceaux *et al.*, “Reliable Multicast with B.I.E.R.,” *Journal of Communications and Networks*, vol. 20, 2018.

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

[31] W. Braun *et al.*, “Performance Comparison of Resilience Mechanisms for Stateless Multicast using BIER,” in *IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, 2017.

[32] D. Katz *et al.*, *Bidirectional Forwarding Detection (BFD)*, https://datatracker.ietf.org/doc/rfc5880/, Jul. 2004.

[33] Q. Xiong *et al.*, *BIER BFD*, https://datatracker.ietf.org/doc/draft-hu-bier-bfd/, Oct. 2017.

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

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

[36] *P416Portable Switch Architecture (PSA)*, https://p4.org/p4-spec/docs/PSA-v1.1.0.pdf, Nov. 2018.

[37] Edge-Core Networks, *Wedge100BF-32X/65X Switch*, https://www.edge-core.com/_upload/images/Wedge100BF-32X_65X_DS_R05_20191210.pdf, 2019.

[38] J. Geng, J. Yan, and Y. Zhang, “P4QCN: Congestion Control Using P4-Capable Device in Data Center Networks,” *Electronics*, vol. 8, 2019.

[39] C. Wernecke, H. Parzyjegla, G. Mühl, P. Danielis, and D. Timmermann, “Realizing Content-Based Publish/Subscribe with P4,” in *IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2018.

[40] M. Uddin, S. Mukherjee, H. Chang, and T. V. Lakshman, “SDN-based Multi-Protocol Edge Switching for IoT Service Automation,” *IEEE Journal on Selected Areas in Communications (JSAC)*, vol. 36, 2018.

[41] G. Rétvári *et al.*, “IP fast ReRoute: Loop Free Alternates revisited,” in *IEEE International Conference on Computer Communications (INFOCOM)*, 2011.

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