Learning Multicast Patterns for Efficient BIER Forwarding with P4

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

***Abstract*—**
Bit Index Explicit Replication (BIER) is an efficient domain-based transport mechanism for IP multicast (IPMC) that indicates receivers of a packet through a bitstring in the packet header. Recently, BIER forwarding has been implemented on 100 Gbit/s per port hardware using the P4 programming language. However, the implementation requires packet recirculation to iteratively serve one next-hop after another. The objective of this paper is to reduce this inefficiency.

Static multicast groups can be configured on P4 switches so that traffic can be sent to all next-hops without recirculation. We leverage that feature to make BIER forwarding more efficient. However, only a limited number of static multicast groups can be configured on a switch, which is not sufficient to cover all potential port patterns. In a first step, we develop efficient BIER forwarding that utilizes static multicast groups derived from so-called configured port clusters. Then, we design port clustering algorithms that observe multicast patterns and compute configured port clusters which are more efficient than randomly selected port clusters. These methods are based on Spectral Clustering, an unsupervised machine learning technique. We perform simulations that underline the effectiveness of this approach to reduce inefficient packet recirculations. We further implement the new forwarding behaviour on programmable hardware and provide a controller that samples BIER packets on the switch, runs the port clustering algorithms, and updates the configured static multicast groups. We validate this open source implementation in a testbed and show that the experimental results are in line with the simulation results.

***Index Terms*—Software-Defined Networking, Bit Index Explicit Replication, Multicast, Resilience, Scalability, Unsupervised Machine Learning**

I. Introduction

IP multicast (IPMC) is an efficient way to distribute one-to-many traffic. It is organized into multicast groups that are identified by unique IP addresses. Traffic of a multicast group is sent to all subscribers along a distribution tree, i.e., nodes replicate and forward packets to specific neighbors towards the subscribers. Therefore, only one packet is sent over each involved link, which reduces the load in comparison to unicast. To that end, core nodes store for each multicast group the neighbours that should receive a packet copy. As a result, traditional IPMC has two scalability issues. First, whenever the composition of an IPMC group changes, signaling to core nodes is necessary to update the neighbors that should receive packet copies. Second, link or node failures, and topology changes may affect multiple multicast groups, which puts high signaling and processing load on core devices.

The IETF proposed Bit Index Explicit Replication (BIER) [1] as an efficient and stateless domain-specific transport mechanism for IPMC traffic. Ingress routers equip an IPMC packet with a bitstring in the BIER header which contains all destinations of the packet within the domain. Core nodes replicate and forward the BIER packet according to its bitstring and the paths from the interior gateway protocol (IGP) which is called routing underlay. Egress routers remove the BIER header, and IPMC processing continues. With BIER, only ingress and egress routers of a domain know IPMC groups and are involved in signalling, but not the core routers.

Recently, we presented an open source BIER implementation for 100 Gbit/s per port in P4 for the Tofino ASIC hardware [2]. This implementation is inefficient as it requires one pipeline iteration per next-hop of a BIER packet as packets are transmitted iteratively instead of simultaneously. Therefore, a packet with n next-hops requires $n-1$ recirculations. On the one hand this is due to the fact that packet replication to a dynamic set of outgoing ports is not supported on the specific hardware device. On the other hand, it is difficult[1] to derive the set of outgoing ports from the bitstring within a single pipeline iteration, which is a general challenge for all switch architectures.

In this paper we present an efficient BIER implementation in P4 for the Tofino ASIC. First, we propose a forwarding algorithm that utilizes static multicast groups to simultaneously forward BIER packets to many outgoing ports. However, the number of configurable static multicast groups is limited and does not suffice to cover all port combinations on a 32-port switch. Therefore, the algorithm leverages subsets of ports, so-called “configured” port clusters. All port combinations within a configured port cluster are configured as static multicast groups. This allows efficient BIER forwarding within a very few pipeline iterations (at most 3 or 4 on the Tofino). To further improve the efficiency, we suggest to choose configured port clusters such that they contain ports over which BIER packets are frequently forwarded together. To that end, we propose port clustering algorithms that learn port patterns from sampled BIER traffic and compute configured port clusters that reduce the number of required forwarding cycles. The methods are based on Spectral Clustering which is an unsupervised

The authors acknowledge the funding by the Deutsche Forschungsgemeinschaft (DFG) under grant ME2727/3-1. The authors alone are responsible for the content of the paper.

[1]BIER bitstrings consist of at least 256 bits and each of them identifies a receiver. This results in 2^{256} possible bit combinations which need to be mapped to up to 2^{32} outgoing port combinations on a switch with 32 ports. This is a challenge for naive table matching.

machine-learning technique. In practice, a controller applies port clustering from time to time on recently sampled BIER traffic and updates the configured port clusters on the switch.

The paper is structured as follows. In Section II and III we describe related work and give an introduction to Bit Index Explicit Replication (BIER). Sections IV and V give a primer on the programming language P4 and cover important aspects of the existing P4-based BIER implementation. Section VI proposes the efficient BIER forwarding algorithm and shows simulation results for arbitrarily selected configured port clusters. Section VII suggests various port clustering methods. Their performance is compared by simulation in Section VIII and by hardware experiments in Section IX. Finally, we conclude the paper in Section X.

II. Related Work

In this section we first review related work on traditional multicast and resilience. Then, we present work related to both SDN- and BIER-based multicast. Finally, we review clustering approaches.

A. Traditional Multicast

Islam et al. [3] and Al-Saeed et al. [4] investigate related work for traditional multicast. The majority of cited papers aim to improve the scalability of traditional IPMC. They present intelligent tree-building mechanisms for multicast to make it more efficient, e.g., by reducing required state, or signaling.

Elmo [5] encodes topology information of data centers in packet headers to improve the scalability of IPMC. It leverages characteristic properties of those topologies to reduce the size of the forwaring information base (FIB) of core routers. The Avalanche Routing Algorithm (AvRA) [6] follows a similar approach where it optimizes link utilization for multicast by leveraging topology characteristics of data center networks. Dual-Structure Multicast (DuSM) [7] separates forwarding structures for high-bandwidth and low-bandwidth traffic to improve scalability and link utilization in data centers. Li et al. [8] optimize the FIB to improve the scalability of traditional multicast in data center networks. To that end, they propose to partition the multicast address space and aggregate those at bootleneck switches.

Application layer multicast (ALM) [9] monitors the traffic on application-specific distribution trees to optimize their structures for the corresponding group objective. Mokhtarian et al. [10] construct minimum-delay trees to reduce latency for delay sensitive data with different requirements like min-average, min-maximum, real-time requirements, etc. Adaptive SDN-based SVC multicast (ASCast) [11] follows a similar approach. The authors describe an integer-linear program to build optimal distribution trees and fast forwarding tables to optimize multicast forwarding in terms of latency and delay for live streaming.

Kaafar et al. [12] present a building scheme for efficient overlay multicast trees based on location-information of subscribers. Boivie et al. [13] propose small group multicast (SGM) which aims at avoiding management and set up overhead for multicast groups with a small number of receivers. To that end, the multicast packets of such groups carry the distribution information in their headers, which avoids signaling in the core. Simple explicit multicast (SEM) [14] stores multicast information only on branching nodes of the distribution tree. Non-branching nodes forward packets to the next-branching node via unicast. Jia et. al. [15] leverage prime numbers and the Chinese remainder theorem to efficiently organize the FIB. They reduce the size of the FIB in core devices and facilitate implementation.

Steiner trees [16] are tree structures that are used to build efficient multicast trees. Many research papers modify Steiner trees to build multicast trees optimized with regard to a specific metric, e.g., link costs [17], delay [18], number of hops [19], number of branch nodes [20], retransmission efficiency [21], or optimal placement of IPMC sources [22].

B. Resilience for Multicast

Shen et al. [23] extend Steiner trees so that distribution trees contain recovery nodes. Such nodes cache multicast traffic for retransmission to cut off receivers after recomputation of the FIB. The authors of [24] investigate resilience of several multicast algorithms against node failures. Kotani et al. [25] deploy primary and backup multicast trees that are identified by a field in the packet header. After failure detection, the source sends its packet over a working backup tree by indicating the backup path in the packet header. Pfeiffenberger et al. [26] propose that each node in a distribution tree is also the root of a backup tree that reaches all downstream destinations over paths that do not include the failed link/node. Nodes switch packets on a backup tree by setting a VLAN tag in the packet header.

C. SDN-Based Multicast

Rückert et al. [27], [28] propose and extend Software-Defined Multicast (SDM) which is an OpenFlow-based multicast platform to facilitate management. It focuses on overlay-based live streaming services for P2P video live streaming. The authors of [29] describe address translation in OpenFlow switches to reduce the number of multicast-dependent forwarding entries in near-to-leaf nodes. To that end, the forwarding action from the last hop towards the receivers is done with a unicast address. Lin et al. [30] implement shared multicast trees between different IPMC groups on OpenFlow switches. Thereby, the number of forwarding entries is reduced. The authors of [31] leverage bloom filters to reduce the number of TCAM-entries that is required for SDN-based multicast.

D. BIER Multicast

In [32], [33] we presented an early prototype of a BIER implementation in P4 for the software switch bmv2 [34]. However, bmv2 yields only low throughput (900 Mbit/s) [35]. Therefore, we developed a P4 implementation of BIER and BIER-FRR for the P4-programmable switching ASIC Tofino [2] with a switching capacity of 3.2 Tb/s, i.e., 100 Gbit/s per port in a 32-port switch. We demonstrated its technical feasibility and performance limits.

Giorgetti et al. [36], [37] presented an OpenFlow implementation of BIER. However, it requires extensive state or controller interaction for efficient BIER forwarding. Furthermore, it is capable of addressing only 20 receivers per packet due do the limited size of MPLS labels which are used to implement arbitrary header fields.

Desmouceaux et al. [38] investigate the retransmission efficiency of BIER. That is, when subscribers signal missing packets, BIER allows to retransmit packets to only specific subscribers while still forwarding only one packet copy per link. Traditional multicast retransmits either via unicast or to the entire multicast group. The evaluations show that BIER is significantly more efficient than traditional multicast, i.e., it causes fewer retransmitted packets and achieves better link utilization.

BIER with tree engineering (BIER-TE) [39] encodes the entire distribution tree in the packet header to have more control of the paths. Carrier grade minimalist multicast (CGM2) [40] is a novel derivate of BIER-TE. It encodes the distribution tree in a recursive manner in the packet header. Thereby, it can scale to larger networks than BIER-TE. However, CGM2 has not been implemented, yet, and is still under development.

Braun et al. [41] propose 1+1 protection for BIER where traffic is transported on two disjoint trees. As a result, traffic is delivered successfully to receivers even when a failure interrupts one tree.

The state of the art for BIER multicast with P4 is limited due to the following reasons. First, existing implementations require additional forwarding capacity as shown in [2] which may significantly reduce the usable physical ports of a switch. Second, other BIER implementations require either exponential state or significant controller interaction (as in native IP multicast), which is contrary to the stateless nature of BIER. In this work, we present a novel approach for efficient BIER forwarding that leverages static multicast groups to reduce the required forwarding capacity by eliminating excessive recirculation. Further, we optimize the selection of the static multicast groups with unsupervised machine learning to further reduce the required recirculation and therefore forwarding capacity.

E. Clustering

Clustering is an unsupervised machine learning technique that solves the problem of identifying clusters of data points in a multidimensional space. Given a set of D-dimensional points $\{x_1, ..., x_N\}$, the goal of clustering is to partition the data into groups/clusters such that points in the same cluster are similar and points in different groups are dissimilar.

k-Means [42] is one of the most applied clustering algorithms. Its incentive is to find an assignment of data points to k cluster centers such that the sum of the squares of the distances of each data point to its cluster center is minimized.

DBSCAN [43] is a density-based clustering algorithm that can form arbitrary clusters and is especially suited for outlier detection. It is not suited for high-dimensional data sets as it uses the euclidean distance as similarity measure.

Spectral Clustering [44] is a clustering algorithm that is based on graph properties. It uses the normalized Laplacian[2] of the similarity matrix of the data points to build k clusters. Data points are embedded in $\mathbb{R}^k$ through the so-called spectral embedding. Thereby, the first k eigenvectors of the Laplacian are computed and used to project the data points. Finally, the embedded data points are clustered with a simple clustering algorithm, e.g, k-Means.

[2]The normalized Laplacian of a graph with weight matrix W and degree matrix D is given as $L = D^{-1/2}(D - W)D^{-1/2}$ [44].

III. Bit Index Explicit Replication (BIER)

In this section we give a short primer on BIER. BIER is a domain-based transport mechanism for multicast traffic. It can be explained with three layers as shown in Figure 1. On

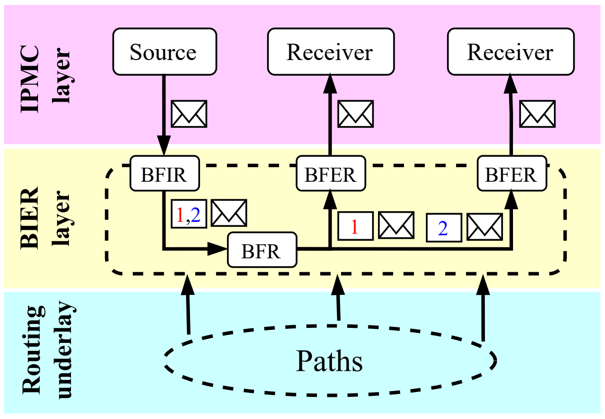


Figure 1: Layered BIER architecture according to [33].

the IPMC layer sources and receivers send and receive IPMC packets. The BIER layer is responsible for the transport of the IPMC packets from the IPMC sources to the IPMC receivers along paths from the unicast routing, i.e., the routing underlay, through the so-called BIER domain.

The BIER domain consists of three types of BIER devices. First, bit forwarding ingress routers (BFIRs) are the entry points to the BIER domain. They encapsulate IPMC packets with a BIER header for forwarding within the BIER domain. The BIER header contains a bit string that indicates all destinations of the BIER packet. That is, each bit position corresponds to a specific destination. A bit is activated if the corresponding destination should receive a copy of the packet. Second, bit forwarding routers (BFRs) forward BIER packets towards their destinations according to the activated bits in the BIER header. That is, a BFR sends a packet copy to the first next-hop over which at least one destination is reached. It leaves only those bits activated in the bit string of the packet copy which correspond to destinations that are reached via that next-hop, and clears all other bits to prevent duplicates at the receivers. The BFR repeats this procedure until all destinations are served. As a result, the forwarding path of a BIER packet is a tree whose links carry only a single packet copy. Third, bit forwarding egress routers (BFERs) remove the BIER header and pass the IPMC packet to the IPMC layer.

Next-hops on the BIER distribution tree may not be reachable due to link or node failures. In this case, downstream destination nodes do not receive any BIER traffic until BIER forwarding tables are updated. Therefore, two BIER fast reroute (BIER-FRR) concepts have been proposed [45] to forward BIER traffic over backup paths from the detection of the failure until BIER forwarding tables have been updated. The methods have been compared in [46] and tunnel-based BIER-FRR has been implemented in [2].

IV. Introduction to P4

In this section we give an overview of P4, explain the P4 processing pipeline, packet cloning, packet recirculation, and multicast groups in P4.

A. P4 Overview

P4 (programming protocol-independent packet processors) [47] is a high-level programming language to describe the data plane of P4-programmable devices. It is applied in a wide range of applications and research [48]. Target-specific compilers map the P4 programs to the programmable processing pipeline of the target devices which are also called targets. The P4 compiler also generates a data plane API that can be used by a control plane to manage runtime behavior, e.g., by writing forwarding entries. P4 targets follow a certain architecture that may vary between different targets, e.g., Intel Tofino implements the TNA architecture whereas some P4 capable SmartNICs may implement the PSA architecture. Packet cloning, multicast, and recirculation are common features that are supported by most P4 architectures. We implemented the subsequently presented mechanism for the Intel Tofino, which follows the TNA architecture. Therefore, most explained P4 related concepts are done with the TNA in mind. However, the presented concepts and mechanisms can be implemented similarly in other architectures.

B. P4 Pipeline

Figure 2 shows the abstract pipeline model of a P4 programmable device [47]. A programmable parser deserializes

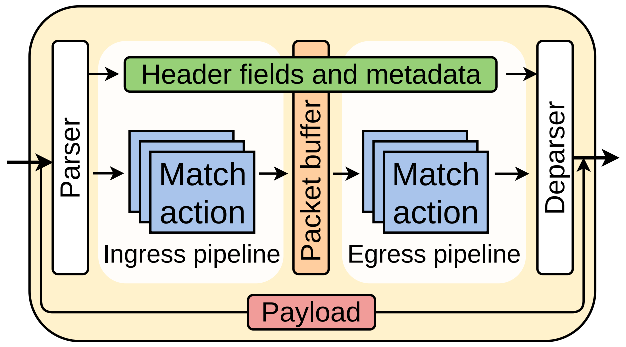


Figure 2: Abstract forwarding model according to [47].

the packet header and stores the information in so-called header fields. The header fields are carried through the pipeline together with packet-specific metadata fields which are comparable to variables from other high-level programming languages. Only header fields and metadata are processed afterwards in the ingress pipeline, i.e., the payload of the packet remains untouched. The ingress pipeline consists of one or more match-action-tables (MATs) that map header fields or metadata to actions. Examples for actions are changing header fields or metadata, e.g., setting the egress port of the packet. After processing in the ingress pipeline, the packet is temporarily buffered so that it can be processed by the egress pipeline which works similarly to the ingress pipeline. Finally, the deparser serializes the possibly changed header fields, forwards the packet through the designated egress port, and discards the metadata.

C. Packet Cloning

P4 has an operation for packet cloning. Depending on the architecture, different clone operations are defined. In the following, we explain ingress to egress (I2E) cloning which we used for the implementation. With I2E cloning, a set metadata flag indicates that the packet should be cloned after its processing in the ingress pipeline has concluded. However, the header fields and metadata of the clone resemble the packet that is initially parsed before the ingress pipeline. Figure 3 shows the concept. After the ingress pipeline has finished,

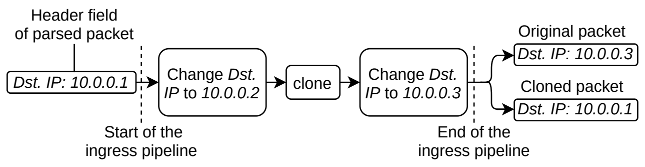


Figure 3: When a packet is cloned, the copy is created only after the ingress pipeline and its header fields are reset to the initial value after the packet has been parsed.

the packet is cloned and both the original packet, i.e., with header changes, and the packet copy, i.e., without header changes, enter the egress pipeline where they are processed independently of each other. Some architectures allow to carry additional information during cloning. In the case of the TNA, this is done through a so-called mirror header.

D. Packet Recirculation

Packet recirculation in P4 allows a packet to be processed a second time by the entire pipeline, i.e., by ingress and egress pipeline. To recirculate a packet, its egress port, i.e., a special metadata field, is set to a particular port ID that corresponds to a switch-intern recirculation port. The recirculation port functions as a regular port of the switch with the exception that it has no physical connector, i.e., only the switch itself can send to and receive traffic from the recirculation port.

After the packet has been processed by both the ingress and egress pipeline, it is sent to the recirculation port. Afterwards, the packet is processed again as if it has been received on a physical port.

E. Static Multicast Groups

P4 allows controllers to configure multicast groups on forwarding devices. A multicast group consists of a tuple of multicast group identifier and a set of egress ports. In addition, there is a special metadata field that allows the ingress pipeline to assign a multicast group identifier to a packet. After the ingress pipeline has completed, the packet is replicated to the pre-defined set of egress ports.

In the following we refer to those configured multicast groups as “static multicast groups” to differentiate them from multicast groups of IPMC. A static multicast group is a local mechanism on a switch to simultaneously forward a packet to multiple egress ports.

V. Simple P4-Based BIER Implementation

In this section we review the simple P4-based BIER implementation of [2]. The target is the Intel Tofino high-speed switching ASIC [49]. It is used for a prototype on the Edgecore Wedge 100BF-32X [50] with 32 100 Gbit/s ports. The implementation makes heavy use of packet recirculation, which causes capacity issue, e.g., 100 Gb/s incoming multicast traffic with 5 next-hops requires 400 Gb/s additional forwarding capacity for recirculation purpose, i.e., it requires #next-hops - 1 recirculations. The efficient BIER implementation in Section VI builds upon the simple implementation and greatly reduces the need for recirculations.

A. BIER Processing

BFRs leverage the Bit Index Forwarding Table (BIFT) to determine the next-hops of a BIER packet. We implement the BIFT as common match-action table in P4. For each BFER there is one entry in the BIFT. The match key is a bitstring with only the single bit activated for the corresponding BFER. The other entry fields, i.e., the corresponding action with its parameters, are a next-hop and a forwarding bitmask (FBM). The FBM is a bit string similar to the BIER bitstring and it indicates the BFERs with the same next-hop. When a packet arrives, the BFR first copies the bitstring of the packet to a temporary metadata field which we call “remaining bits”. The remaining bits indicate the BFERs that still have to be served. Then, the least-significant activated bit in the remaining bits is matched against the BIFT using a ternary match operation. The match-action table entry returns the corresponding next-hop and FBM for that BFER. The BFR clears all bits in the bitstring of the packet that are not activated in the FBM. Thus, only the bits of BFERs that are reached through this next-hop remain in the BIER bitstring. The BFR further clears all bits in the remaining bits that are activated in the FBM as they have already been served. Afterwards, the clone operation is applied. Figure 4 shows the processing flow of the original and cloned BIER packet. The original packet is sent through

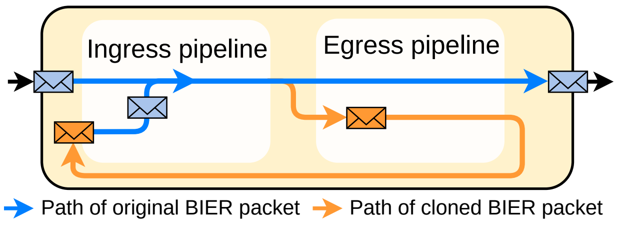


Figure 4: The original BIER packet is sent through an egress port while the packet copy is recirculated.

the appropriate egress port to reach the selected next-hop. The packet copy is cloned to the egress pipeline and recirculated to a recirculation port. Within the egress pipeline, the BIER bitstring of the packet copy is set to the remaining bits so that only the remaining BFERs are served in the next pipeline iteration. Further details about the original BIER forwarding implementation can be found in [2].

B. Recirculation Capacity and Problem Statement

The Tofino ASIC has a switch-intern recirculation port which has the same packet processing capacity as regular ports. If its capacity does not suffice, packet loss occurs. To increase the recirculation capacity, physical ports may be turned into loopback mode, and recirculation traffic may be distributed over the internal ports and the loopback ports in a round-robin manner [2]. As these ports cannot be utilized for other traffic, recirculations are costly.

The simple BIER implementation requires $n-1$ recirculations for BIER packets with n next-hops. This approach obviously does not scale well with increasing number of next-hops and traffic rate. The objective of this paper is a more efficient P4-based implementation that requires fewer recirculations per BIER packet (see Section VI) and an optimized configuration thereof using clustering methods (see Section VII).

VI. Efficient BIER Forwarding with P4

We explain how static multicast groups can be leveraged to make BIER forwarding using P4 more efficient, and how BIER-FRR can be integrated. To demonstrate the efficiency of the new forwarding algorithm, we present a simulative performance study.

A. Efficient BIER Forwarding with Static Multicast Groups

We first explain how BIER forwarding can profit from configured port clusters consisting of static multicast groups such that multiple next-hops can be served within a single pipeline iteration. Then we explain how the forwarding algorithm determines a port cluster and the appropriate static multicast group for a BIER packet, and forwards it.

The presented algorithm is specific to P4 and the architecture of the Tofino ASIC. However, efficient forwarding algorithms for any switch architecture need to determine the set of egress ports for a BIER packet. This is a difficult task as bitstrings are at least 256 bits large. Therefore, the presented approach may also be a useful base for efficient BIER forwarding on other switch architectures.

1) Use of Static Multicast Groups: The idea to make BIER forwarding more efficient is the use of static multicast groups so that multiple egress ports can be simultaneously served instead of using recirculation.

A naive solution is configuring static multicast for all possible combinations of egress ports. When a packet arrives, the set of egress ports is determined and the corresponding static multicast group forwards the packet to all needed egress ports without packet recirculation. However, on a 32 port switch this requires $2^{32} = 4294967296$ static multicast groups, which exceeds the number of configurable static multicast ports.

We propose now a more sophisticated approach which requires fewer static multicast groups. We define a set $\mathcal{C} = \{C_1, ..., C_k\}$ of so-called "configured port clusters" (or port sets) C_i such that all configured port clusters together cover all ports of a switch. Configured port clusters may be disjoint or overlapping. For each port set C_i, static multicast groups $M_j \in \mathbb{P}(C_i)$ are configured for all sets of ports in the powerset of C_i. Thus, a configured port cluster C implies

$$m(C) = 2^{|C|} - |C| - 1 \qquad (1)$$

static multicast groups that need explicit configuration on the switch; the empty group and groups with only a single destination do not need to be configured. On a 32-port switch three port clusters with 10, 11, and 11 ports may be configured, which requires in total 5085 explicitly configured static multicast groups. This is well feasible on a switch like the Tofino which supports up to $2^{16} = 65536$ static multicast groups[3]. With this approach, a BIER packet needs to be sent to at most $|\mathcal{C}|$ static multicast groups, which requires $|\mathcal{C}| - 1$ recirculations instead of $n_h - 1$ with n_h being the number of next-hops of a BIER packet. Moreover, the administrator may set a threshold m_{max} on the number of static multicast groups usable for efficient BIER forwarding.

2) Forwarding Procedure: We first give a forwarding example. Then, we describe how the forwarding procedure selects a set of configured port clusters $\mathcal{S}_i \subseteq \mathcal{C}$ for BIER forwarding, and then we present how the appropriate static multicast group is chosen from a selected configured port cluster $C_j \in \mathcal{S}_i$.

a) Forwarding Example: Figure 5 illustrates an example for a 8-port switch with three configured port clusters, $\mathcal{C} = \{C_1 = \{1,2,3\},\ C_2 = \{4,5,6\},\ C_3 = \{6,7,8\}\}$. For each configured port cluster, all port combinations are configured as static multicast groups. The empty group and groups with only a single port do not need to be configured. We consider all subsets of configured port clusters $\mathcal{S}_i \subseteq \mathcal{C}$. A packet destined for ports 1, 3, and 4 requires the subset $\mathcal{S}_4 = \{C_1, C_2\}$ for forwarding, i.e., it will be served by the multicast group $\{1,3\}$ from C_1 and the multicast group $\{4\}$ from C_2, which requires a single recirculation. Note that groups with a single destination do not need to be configured as static multicast group.

b) Selection of Set of Configured Port Clusters: Now we explain how the appropriate subset $\mathcal{S}_i$ for forwarding is determined in the data plane. C-FBM($\mathcal{S}_i$) is the combined forwarding bitmask of a subset of configured port clusters and indicates all BFERs that are reachable through $\mathcal{S}_i$. We set up a match-action table with one entry per subset $\mathcal{S}_i$ in increasing order with regard to subset size $|\mathcal{S}_i|$, as shown in Figure 5. The entry ¬C-FBM($\mathcal{S}_i$) is the complement of C-FBM($\mathcal{S}_i$). The objective is to find the smallest subset of configured port clusters that serves all BFERs of a BIER packet. To that end, the bitstring of a packet is bitwise ANDed with the complement of the C-FBM in the match-action table. We define a match if the result of that operation is zero. Then, all BFERs of the BIER packet are served by the corresponding subset $\mathcal{S}_i$. This operation is done through a ternary match. A ternary match in P4 is defined by a source value s_v, e.g., a header field, and a ($mask$, $value$) pair. The corresponding table entry matches when s_v & $mask = value$. The source value is given by the BIER bitstring, the mask is the complement of the C-FBM and the value is 0. Due to the order within the match-action table, the first match $\mathcal{S}_i$ is the smallest subset with that property. The first configured port cluster C_j in that subset $\mathcal{S}_i$ is selected for the remainder of the forwarding process.

We consider the example of Figure 5, where 8 BFERs are reachable over ports 1-8. For simplicity, BFER i corresponds to the i-th bit in the BIER bitstring and is reachable over port i, i.e., BFER 1 corresponds to the least significant bit and is reachable over port 1. The C-FBMs for all subsets $\mathcal{S}_i \subseteq \mathcal{C}$ are given in Table 1.

Table 1: Subsets $\mathcal{S}_i \subseteq \mathcal{C}$ with corresponding C-FBMs.

Subset	C-FBM	¬C-FBM
$\mathcal{S}_1 : \{C_1\}$	00000111	11111000
$\mathcal{S}_2 : \{C_2\}$	00111000	11000111
$\mathcal{S}_3 : \{C_3\}$	11100000	00011111
$\mathcal{S}_4 : \{C_1, C_2\}$	00111111	11000000
$\mathcal{S}_5 : \{C_1, C_3\}$	11100111	00011000
$\mathcal{S}_6 : \{C_2, C_3\}$	11111000	00000111
$\mathcal{S}_7 : \{C_1, C_2, C_3\}$	11111111	00000000

Again, we assume a BIER packet to be destined for ports 1, 3, and 4, i.e., towards BFERs 1, 3, and 4 with a bitstring $bs = 00001101$; then only $\mathcal{S}_4$ or $\mathcal{S}_7$ can cover all BFERs of the packet[4]. Due to the order within the match-action table, the first match is $\mathcal{S}_4$ and C_1 is selected for the remainder of the forwarding process.

c) Selection of the Static Multicast Group: Only a single static multicast group M_h of the configured port cluster C_j will be used for forwarding. We now determine that $M_h \in \mathbb{P}(C_j)$ and take a similar approach as in Section VI-A2b for that purpose. We set up a match-action table for C_j which has an entry for any static multicast group $M_h \in \mathbb{P}(C_j)$. The entries are sorted by increasing group size $|M_h|$ and contain the complement of the C-FBM of the corresponding multicast group. Single ports are also considered as static multicast groups although they do not require explicit configuration on the switch. The bitstring of a BIER packet is first ANDed with the C-FBM of the selected configured port cluster C_j. This excludes all BFERs from the bitstring that cannot be served by C_j. The result is bitwise ANDed with the complement of the C-FBM(M_h) of the multicast groups in the table entries. We define a match if the result is zero. This is done with a ternary match operation as with the selection of a configured port cluster. Due to the increasing order of entries in the match-action table, the first match refers to the smallest static multicast group M_h within the configured port cluster that covers all relevant BFERs.

d) Forwarding and Bitstring Adaptation: At the end of the ingress pipeline, the original BIER bitstring is stored in a transient metadata header. The activated bits in C-FBM(M_h)

[3]The actual usable number of available resources depends on the program complexity.

[4]This is ensured through the ternary match operation: bs & ¬C-FBM(S_i) == 0.

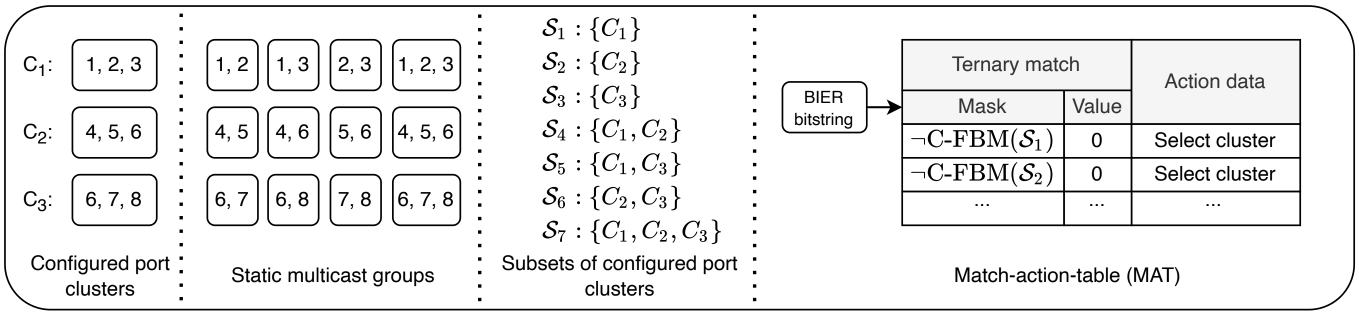

C1:
1, 2, 3
C2:
4, 5, 6
C3:
6, 7, 8
Configured port clusters
1, 2
1, 3
2, 3
1, 2, 3
4, 5
4, 6
5, 6
4, 5, 6
6, 7
6, 8
7, 8
6, 7, 8
Static multicast groups
$\mathcal{S}_1 : \{C_1\}$
$\mathcal{S}_2 : \{C_2\}$
$\mathcal{S}_3 : \{C_3\}$
$\mathcal{S}_4 : \{C_1, C_2\}$
$\mathcal{S}_5 : \{C_1, C_3\}$
$\mathcal{S}_6 : \{C_2, C_3\}$
$\mathcal{S}_7 : \{C_1, C_2, C_3\}$
Subsets of configured port clusters
BIER bitstring
Ternary match
Action data
Mask
Value
$\neg$C-FBM($\mathcal{S}_1$)
0
Select cluster
$\neg$C-FBM($\mathcal{S}_2$)
0
Select cluster
...
...
...
Match-action-table (MAT)

Figure 5: Configured port clusters C_i together cover all ports of a switch. All port combinations within a configured port cluster are configured as static multicast groups. A match-action table chooses a minimum subset of configured port clusters and selects its first configured port cluster for forwarding.

are deactivated in this transient metadata header which represents the remaining bits that need processing; if the bitstring is not zero, the packet is recirculated and the BIER bitstring is restored through the transient metadata header in the egress pipeline of the recirculation port[5]. In addition, a copy of the original packet is sent to all egress ports of the selected static multicast group M_h. The egress pipelines of these ports clear all bits in the packet's bitstring that are not activated in the FBM of the corresponding port and then they transmit the packets.

B. Integration of BIER-FRR

The proposed efficient forwarding scheme is compatible with BIER-FRR if BIER-FRR is integrated as follows. First, the switch processes the egress ports that are affected by a failure, i.e., a failed link or a failed node. To that end, the BIER packets are forwarded by regular BIER forwarding but over alternate ports. When all affected egress ports have been served, the BIER packet is recirculated and the remaining ports, i.e., working ports, are processed by the presented, efficient forwarding algorithm. This approach prevents duplicates at subscribers and unnecessary double transmissions of the same packet over one link. Details are given in [2].

C. Simulative Performance Evaluation

We evaluate the concept of static multicast groups for efficient BIER forwarding through the following experiment. We examine different numbers of disjoint configured port clusters $k \in \{1, 2, 4, 8, 16, 32\}$. With k configured port clusters and a 32 port switch, each configured port cluster contains $\frac{32}{k}$ ports. Further, we simulate BIER packets with $n_h \in \{1, 2, 4, 8, 16, 32\}$ random next-hops. They are processed by the different configured port clusters. Figure 6(a) and Figure 6(b) show the average number of recirculations per packet and the required static multicast groups. Although multicast traffic with random next-hops is unrealistic (see Section VIII-A1), it serves as a good baseline for a performance evaluation of the efficient BIER forwarding mechanism. The average number of recirculations increases with the number of next-hops n_h and the number of configured port clusters k. In fact, the number of recirculations is bound by $k - 1$. For $k = 32$, the results are equivalent to the simple (original) BIER forwarding. Higher values of k lead to smaller configured port clusters, and hence, to fewer next-hops that can be served in one shot. The number of required static multicast groups decreases with the number of configured port clusters k. To keep the number of recirculations low, larger configured port clusters should be preferred. However, the number of available static multicast groups may be limited due to technical reasons or based on administrative decisions.

[5]This restores all bits from the original BIER bitstring that have not been processed yet.

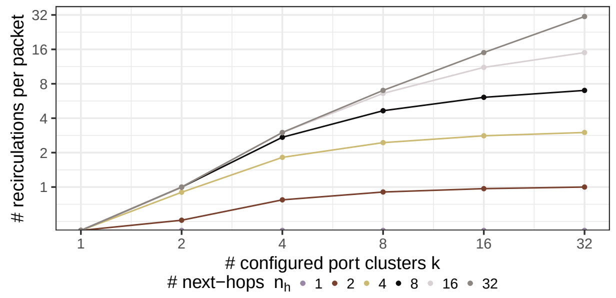
In the given traffic model, we randomly selected next-hops for BIER packets. This is not a realistic model for multicast traffic. The next-hops of subsequent BIER packets are likely to be correlated and so are the ports over which the packets are sent. Therefore, some configured port clusters reduce the average recirculation more than others. To effectively minimize the number of recirculations, it is necessary to form meaningful configured port clusters that take the current traffic model into account.

VII. Port Clustering Algorithms for Efficient BIER Forwarding

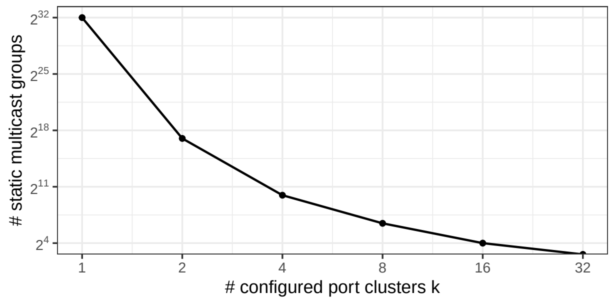
In this section, we first illustrate the optimization potential of efficient BIER forwarding through configuration of appropriate port clusters. Then, we present three clustering algorithms to reduce the average recirculations per packet: random port clustering (RPC) as a simple baseline, port clustering based on Spectral Clustering (PCSC), and recursive clustering with overlaps (RPCO) which also leverages Spectral Clustering for subroutines. For the latter two algorithms we present a graph embedding method that turns ports of sampled packets into a graph structure from which the algorithms learn correlated port clusters.

A. Optimization Potential and Approach

The bits in the BIER header require a packet to be sent to a certain set of next-hops, and, thereby, to specific ports of a switch. To be brief, we talk about "ports of a packet". In the previous section we showed how multiple ports of a BIER packet can be served at once to speed up the forwarding process. For example, port clusters $\{1, .., 8\}$, $\{9, .., 16\}$,



(a) Average number of recirculations per packet for $n_h \in \{1, 2, 4, 8, 16, 32\}$ random next-hops.



(b) Number of required static multicast groups.

Figure 6: Average number of recirculations and number of static multicast groups for $k \in \{1, 2, 4, 8, 16\}$ configured port clusters.

$\{17, .., 24\}$, and $\{25, .., 32\}$ may be configured. Then, a BIER packet needs to be processed at most four times, i.e., it must be recirculated at most three times, no matter how many BFERs are set in the BIER header. If a packet has only ports in the range $\{1, .., 8\}$, the packet does not need to be recirculated at all. However, if a packet has ports $\{1, 9, 17, 25\}$, it still requires three recirculations. The worst-case performance of the presented mechanism is therefore the number of configured port clusters $|\mathcal{C}|$ - 1 instead of #next-hops - 1. This also holds for the subsequently presented optimization algorithms RPC, PCSC, and RPCO.

We now assume that ports of a packet are not random but correlated. That is, certain ports sets tend to occur together. We call them correlated port clusters. We propose to learn these correlated port clusters from sampled traffic and to utilize them as configured port clusters. Then it is likely that BIER packets can be forwarded with fewer processing steps and, thereby, the number of recirculations may be reduced. In practice, a controller can continuously sample multicast traffic from a switch, learn the correlated port clusters of the sampled multicast traffic, and adjust the configured port clusters on the switch.

Large configured port clusters require lots of static multicast groups, but they have the potential to effectively reduce the number of recirculations. A constraint is the maximum number m_{max} of static multicast groups usable for configured port clusters which may be a technical limit or defined by the administrator.

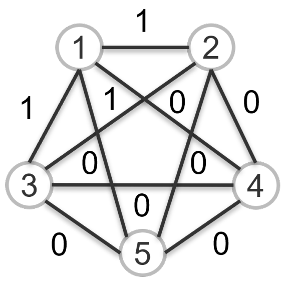
B. Random Port Clustering (RPC)

With RPC, n_p ports are randomly partitioned into approximately k equal-size clusters. The number of clusters k is determined such that the resulting number of configured static multicast groups is at most m_{max}. As the algorithm is trivial, we do not provide any further details. The method will serve as a baseline for a performance comparison.

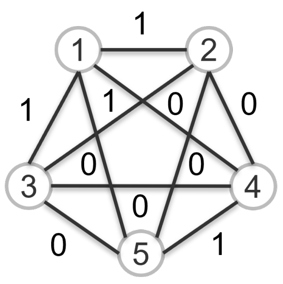
C. Port Clustering based on Spectral Clustering (PCSC)

We first present a graph embedding method that turns ports of sampled packets into a graph structure from which the algorithms learn correlated port clusters. Then we present the PCSC algorithm which is based on Spectral Clustering. It partitions n_p ports into approximately equal-size port clusters.

1) Graph Embedding: We embed the port information of sampled packets into a graph which is needed by the algorithms for PCSC and RPCO. The nodes of the graph represent the ports of a switch. The graph is fully connected and the edges have weights. All weights are initially zero. The embedding iteratively processes the sampled packets. For every combination of two ports of a packet, the weight of the link between these ports is increased by one. Figure 7(a)-Figure 7(b) illustrate how two sampled packets with ports $\{1, 2, 3\}$ and $\{4, 5\}$, respectively, modify an embedded graph with 5 nodes whose edges are initially all zero.



(a) The edge weights between egress ports 1, 2, and 3 are increased by one.

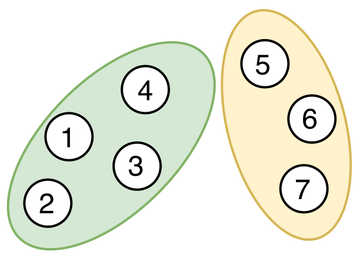


(b) The edge weights between egress ports 4 and 5 are increased by one.

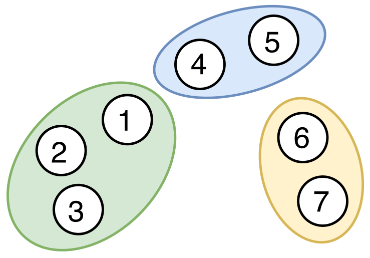
Figure 7: Graph embedding: a full-mesh graph is augmented by port information from sampled packets: high edge weights indicate port pairs that frequently occur together in a BIER packet.

2) PCSC Algorithm: We first develop a metric for port clusters that correlates with the number of recirculations needed for the sampled traffic. Then we propose pseudocode for PCSC that minimizes that number while respecting the number of usable static multicast groups.

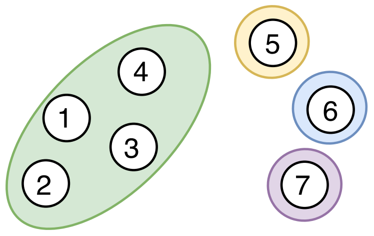
a) Metric: We consider two port clusters C_1 and C_2. The clustering is good if only a few BIER packets need to be sent through ports of C_1 and C_2. We identify a metric for the graph embedding that correlates with that number of



(a) Two almost equal-size port clusters need 15 static multicast groups.



(b) Three almost equal-size port clusters need 6 static multicast groups.



(c) The optimal port clusters have unequal size and need 11 static multicast groups.

Figure 8: Most BIER packets are sent to ports $\{1, 2, 3, 4\}$, $\{3, 4\}$, and $\{4, 5\}$ on a 7-port switch and the maximum number of usable static multicast groups is $m_{max} = 12$. PCSC produces equal-size port clusters while the optimum port clusters minimizing the overall number of recirculations has unequal size.

packets although it is not an exact measure for it. The function $cut(C_1, C_2)$ is the sum of the weights on the edges between any two nodes $v_1 \in C_1$ and $v_2 \in C_2$. It gives an upper bound on the number of packets with ports in both C_1 and C_2. It is an upper bound and not the exact number as a packet may have multiple ports from C_1 and/or C_2. To assess whether the clustering is good, we need to relate $cut(C_1, C_2)$ to the overall number of nodes in the considered clusters. This can be done with the so-called normalized cut (Ncut) and is given below, generalized for multiple clusters.

$$Ncut(C_1, ..., C_k) = \sum_{i=1}^{k} \frac{cut(C_i, \overline{C_i})}{vol(C_i)}$$

Thereby, $cut(C_i, \overline{C_i})$ measures the sum of the edge weights between nodes in C_i and nodes that are not in C_i ($\overline{C_i}$). The function $vol(C_i)$ sums up the edge weights of all nodes in C_i – as a result, edge weights between nodes within the cluster are counted twice, weights of outgoing edges are counted once. The objective is to find clusters $C_1, ..., C_k$ that minimize the normalized cut. Ncut is known to be NP hard and therefore cannot be solved efficiently. However, Spectral Clustering is a relaxation of Ncut. It yields a partition $\mathcal{C}$ with preferably equal-size[6] clusters $C_i \in \mathcal{C}$ and can be solved efficiently [44].

b) Pseudocode for PCSC: PCSC is described in Algorithm 1. It first performs the graph embedding for the set of sampled packets $\mathcal{S}$ and the given number of nodes n_p. Then, Spectral Clustering is called to provide a partition $\mathcal{C}$ of the n_p nodes into k clusters. This is performed in a loop, starting from a single cluster up to n_p clusters. As soon as a partition $\mathcal{C}$ is found that requires at most m_{max} static multicast groups, the algorithm stops and $\mathcal{C}$ is returned. It is the clustering with the lowest number of clusters that can be configured with m_{max} static multicast groups.

[6]This property is desirable as the number of static multicast groups increases exponentially with the number of nodes in a cluster.

Algorithm 1 PCSC

```
Input: samples: $\mathcal{S}$
       number of ports: $n_p$
       number of multicast groups: $m_{max}$

$graph$ = $graphEmbedding(n_p, \mathcal{S})$
for $k$ from 1 to $n_p$ do
    $\mathcal{C}$ = SpectralClustering($graph$, $k$)
    if number of multicast groups for $\mathcal{C} \leq m_{max}$ then
        return $\mathcal{C}$
    end
end
```

D. Recursive Port Clustering with Overlap (RPCO)

We first explain two major shortcomings of PCSC. Then we explain how RPCO solves these shortcomings. Finally, we give a high-level pseudocode description of RPCO.

1) Shortcomings of PCSC: PCSC has two major shortcomings. First, if the configured port clusters cannot be built, the number of clusters is increased by one. As a result, an important cluster that significantly reduces the number of recirculations may not be built although a less important cluster could be split to save static multicast groups.

We illustrate that with a 7-port switch and $m_{max} = 12$ usable static multicast groups. We assume that most multicast packets are sent to port clusters $\{1, 2, 3, 4\}$, $\{3, 4\}$, and $\{4, 5\}$. When PCSC is called with $k = 2$, the clusters in Figure 8(a) may be returned which require 15 static multicast groups, which exceeds m_{max} so that it is not a valid solution. Therefore, PCSC increases k to 3, which may return the clusters in Figure 8(b) which require only 6 static multicast group. As this is feasible, this clustering is PCSC's final result. However, the optimal clustering that minimizes the overall number of recirculations might be the one in Figure 8(c) with 4 unequal-size clusters. They require 11 static multicast groups, which is also feasible.

Second, PCSC creates disjoint clusters. This, however, may not be optimal. We illustrate that by a small example. We consider packets with ports $\{1, 2, 3\}$ and $\{2, 3, 4\}$ and

$m_{max} = 8$ usable static multicast ports. A single, large cluster $C = \{1, 2, 3, 4\}$ requires $m(C) = 11$ static multicast groups so that it cannot be configured. When working with smaller, non-overlapping clusters, it is not possible to cover the port sets of both packets with only a single port cluster. However, when working with overlapping port clusters $C_1 = \{1, 2, 3\}$ and $C_2 = \{2, 3, 4\}$, only 7 static multicast groups are needed[7], which is feasible. Moreover, the port sets of both packets can be covered. Thus, overlapping clusters may help to further reduce the number of recirculations with a limited number of usable static multicast groups.

2) Design Ideas: We discuss major design ideas of RPCO. If the number of usable static multicast groups m_{max} does not suffice to configure k clusters proposed by Spectral Clustering, RPCO selects the clusters that reduce recirculations in the most efficient way and recursively re-clusters the remaining clusters. To that end, we review and adapt the knapsack algorithm to select the clusters that reduce recirculations most efficiently. Given a clustering, we further suggest how to add nodes also to other clusters they are not yet part of, which facilitates cluster overlaps.

a) The Knapsack Algorithm: In the knapsack problem [51], a set of items is given, and each item has a weight and a value. The knapsack objective is to select items such that their overall weight is less than a given limit while their overall value is maximized.

We apply the knapsack algorithm as follows. The set of items is given as set of port clusters $\mathcal{C} = \{C_1, ..., C_k\}$. The value of a port cluster C_i is given by the number of recirculation it saves for the set of packets $\mathcal{S}$ which is evaluated by simulation. The weight of a port cluster C_i is given by its number of required static multicast groups $m(C_i)$. The limit is the number of usable static multicast groups. The algorithm selects those clusters that maximize the number of saved recirculations with the available static multicast groups.

b) Adding Single Nodes to Multiple Clusters: We first define the so-called port-cluster relevance $r(x, C)$ of a port x and a cluster C, $x \notin C$. Then, we explain how the port-cluster relevance is used to add single nodes to multiple clusters.

The port-cluster relevance measures the connectivity between port x and cluster C. It is the sum of the edge weights w between x and C, i.e., $r(x, C) = \sum_{y \in C} w(x, y)$.

Ports are initially assigned to a cluster with Spectral Clustering. However, ports may also be important for other clusters. The list of all port-cluster pairs sorted by decreasing port-cluster relevance suggests the order in which nodes should be additionally added to another cluster provided the remaining static multicast groups suffice. As a result, a partition of ports becomes a port clustering with overlaps.

3) Pseudocode for RPCO: We give a high-level pseudocode for RPCO and refer to the Github repository[8] for details.

Algorithm 2 describes the outer control loop of RPCO. First, the graph embedding of the samples $\mathcal{S}$ is computed and stored in $graph$. Then, the best clustering $\mathcal{C}_{best}$ is initialized with single node clusters. A graph with n_p nodes (number of ports on the switch) can be partitioned into up to n_p clusters. Therefore, the subsequent loop is called with k between 1 and n_p. Within the loop, the current clustering $\mathcal{C}$ is initialized empty and the number of remaining static multicast groups m_{left} is initialized with m_{max}. Both $\mathcal{C}$ and m_{left} are global variables so that they can be modified by subroutines. RecursiveClustering computes a partition of all nodes and stores it in $\mathcal{C}$. Details of the procedure will be explained later. Then, OverlapClusters utilizes remaining usable static multicast groups m_{left} to add nodes to other clusters they are not yet part of (see Section VII-D2b). This leads to overlapping clusters. Afterwards, the best clustering $\mathcal{C}_{best}$ is updated by $\mathcal{C}$ if $\mathcal{C}$ requires fewer recirculations than the best clustering. The function Recirculations($\mathcal{C}, \mathcal{S}$) computes the number of recirculations required for clustering $\mathcal{C}$ for the packets in $\mathcal{S}$. Finally, RPCO returns the best clustering of all switch ports that minimizes the number of recirculations for the samples $\mathcal{S}$.

Algorithm 2 RPCO

```
Input: samples: $\mathcal{S}$
       number of ports: $n_p$
       max. number of multicast groups: $m_{max}$

$graph$ = GraphEmbedding($n_p, \mathcal{S}$)
$\mathcal{C}_{best} = \{\{1\}, ...\{n_p\}\}$
for $k \in [1, n_p]$ do
    $\mathcal{C} = \{\emptyset\}$
    $m_{left} = m_{max}$
    RecursiveClustering($graph, k$)
    OverlapClusters($graph$)
    if Recirculations($\mathcal{C}, \mathcal{S}$) < Recirculations($\mathcal{C}_{best}, \mathcal{S}$) then
        $\mathcal{C}_{best} = \mathcal{C}$
    end
end
return Best port clustering $\mathcal{C}_{best}$
```

RecursiveClustering is described in Algorithm 3. If the graph contains only a single node v, the node is added as a separate cluster to $\mathcal{C}$ and the recursion ends. Otherwise, Spectral Clustering is executed to produce clustering $\mathcal{C}'$ with the desired number of clusters k. Then, the cluster set $\mathcal{C}^*$ is identified which makes best use of the remaining static multicast groups m_{left} to reduce recirculations. All clusters in $\mathcal{C}^*$ are added to the current clustering result $\mathcal{C}$ and m_{left} is decreased by their number of required static multicast groups. The clusters not selected by knapsack ($\mathcal{C}' \setminus \mathcal{C}^*$) are recursively clustered. To that end, the corresponding embedded subgraph is computed. The recursion ends if either the recursion was called with a single node or if all clusters $\mathcal{C}'$ can be selected.

VIII. Simulative Performance Comparison

In this section we compare the performance of the three port clustering methods Random Port Clustering (RPC), Port Clustering based on Spectral Clustering (PCSC), and Recursive Port Clustering with Overlaps (RPCO). We first develop a model for correlated multicast traffic and explain the

[7]When working with overlapping port clusters, the static multicast groups required by multiple port clusters need to be configured only once on the switch.

[8]Github: https://github.com/uni-tue-kn/rpco

Algorithm 3 RecursiveClustering

```
Input: graph embedding: graph
       number of clusters: $k$
if $graph$ contains only the single node $v$ then
    $\mathcal{C} = \mathcal{C} \cup \{\{v\}\}$
    return
end
$\mathcal{C}'$ = SpectralClustering$(graph, k)$
$\mathcal{C}^* =$ knapsack$(\mathcal{C}', \mathcal{S}, m_{left})$
for $C \in \mathcal{C}^*$ do
    $\mathcal{C} = \mathcal{C} \cup \{C\}$
    $m_{left} = m_{left} - m(C)$
end
for $C \in \mathcal{C}' \setminus \mathcal{C}^*$ do
    $subgraph$ = subgraph of $graph$ limited to nodes in $C$
    RecursiveClustering$(subgraph, 2)$
end
```

performance evaluation methodology. Then, we compare the performance of the mentioned clustering methods for various correlated multicast traffic models. Finally, we compare the runtime of the algorithms and discuss their scalability properties.

A. Traffic Model and Evaluation Methodology

We define a simple model for correlated multicast traffic and explain the methodology for the subsequent comparison of the port clustering methods.

1) Model for Correlated Multicast Traffic: In Section VI-C we utilized a model for multicast traffic that assumes random ports for subsequent multicast packets. However, random ports are not realistic for two reasons. First, subsequent multicast packets belong to a set of active multicast groups and packets of a multicast group have identical ports as long as the groups do not change. Second, receivers of multicast groups are users or connected upstream aggregation points in specific time zones, geographical regions, or neighborhoods. Therefore, we assume the users have common interests for certain multicast content so that they belong to multicast groups with correlated receivers. We have not found any literature studying this issue and think this would be useful future work.

We propose a model for correlated multicast traffic for use in the subsequent performance comparison. We define a set of generating port clusters $\mathcal{C}_g = \{C_1, C_2, ..., C_k\}$ from which ports of a packet are preferentially chosen. First, we randomly choose one generating port cluster C_i; thereby all C_i have equal probability. Then, we determine a random number of ports which is equally distributed between 1 and the size $|C_i|$ of the chosen cluster. We draw these ports with a probability p from C_i (without duplicates) and with probability $1-p$ from ports outside C_i (without duplicates).

For $p = 1$, all ports of a sampled BIER packet are from a single, generating port cluster C_i. In that case, if the generating port clusters $\mathcal{C}_g$ are configured for efficient BIER forwarding, BIER packets can be forwarded without recirculation. As p decreases, a sampled BIER packet is likely to have increasingly more ports outside the selected generating port cluster C_i. That means, the resulting multicast traffic is more random and more recirculations are needed. We take p as a measure for port correlation in the generated multicast traffic.

2) Evaluation Methodology: The objective of port clustering algorithms for efficient BIER forwarding is the reduction of recirculations. Therefore, we take the average number of recirculations per packet as performance metric for the subsequent comparisons.

We generate 1000 BIER packets. Based on these packets we compute sets of port clusters for optimized configuration using the considered port clustering methods and various numbers of usable static multicast ports m_{max}. Then, we generate another 10000 packets and simulate efficient BIER forwarding using the optimized configuration. We count the number of recirculations and compute the average number of recirculations per packet. We conduct the experiments 100 times and produce 95% confidence intervals for the average number of recirculations. As they are very small, we omit them in the figures for the sake of readability.

B. Performance Comparison of Port Clustering Methods

We compare the efficiency of the port clustering algorithms for different traffic models. We consider disjoint and overlapping generating port clusters of equal and unequal size. We choose the models such that they all lead to 4.5 ports per BIER packet, which makes their results comparable.

1) Multicast Traffic Generated from Disjoint Port Clusters: We study correlated multicast traffic generated from disjoint generating port clusters. We consider symmetric and asymmetric generating port clusters.

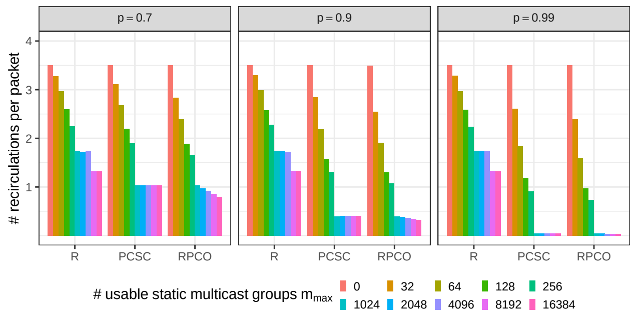
a) Symmetric Generating Port Clusters: We consider four symmetric, disjoint, generating port clusters of size 8: $C_1 = \{1, .., 8\}$, $C_2 = \{9, .., 16\}$, $C_3 = \{17, .., 24\}$, $C_4 = \{25, .., 32\}$. If they are used for configuration, $4 \cdot (2^8 - 8 - 1) = 988$ static multicast groups are needed.

Figure 9(a) shows the average number of recirculations per packet for traffic models with port correlation $p \in \{0.7, 0.9, 0.99\}$, for usable static multicast groups $m_{max} \in \{0, 32, 64, 128, 256, 10.24, 2048, 4096, 8192, 16384\}$, and for the port clustering methods RPC, PCSC, and RPCO.

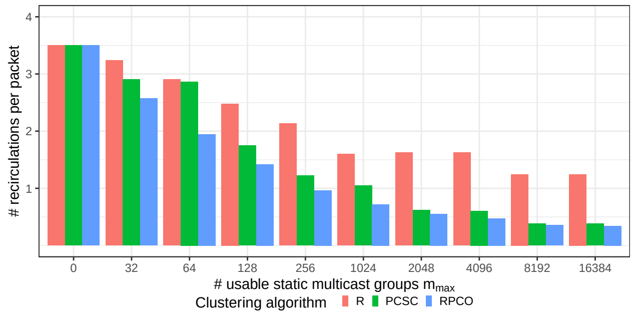
If no static multicast group is available for efficient BIER forwarding ($m_{max} = 0$), the port clustering is disabled, and the forwarding behaviour is the same as the one for simple BIER forwarding. Therefore, packets with 4.5 ports on average require 3.5 recirculations on average. Increasing the number of usable multicast groups m_{max} allows efficient BIER forwarding to decrease the average number of recirculations per packet. This holds for all traffic models and for all port clustering methods. However, if sufficient static multicast groups are available, the degree to which the average number of recirculations can be reduced depends on the port correlation p and the port clustering method.

If a packet with l ports is generated from a specific generating port cluster, all the ports are taken from that cluster with a probability of p^l. Setting $l = 4.5$ yields 20.1% for $p = 0.7$, 62.2% for $p = 0.9$, and 95.6% for $p = 0.99$.

Thus, the chosen traffic models are quite divers. For port correlation $p = 0.7$, the average number of recirculations are similar for all considered port clustering algorithms. The advanced port clustering algorithms hardly outperform the random method due to the lack of sufficient port correlation in the generated multicast traffic. For port correlation $p = 0.99$, most packets are entirely drawn from a single generating port cluster. As a result, the advanced packet clustering methods lead to significantly fewer packet recirculations than random clustering. With $m_{max} = 1024$ or more usable multicast groups, PCSC and RPCO reduce the average number of recirculations to almost zero. Apparently they are able to learn the right port clusters. The generating port clusters are optimal for configuration; as mentioned above, they require 988 static multicast groups. This explains why $m_{max} = 512$ or fewer static multicast groups require more recirculations, also with advanced port clustering methods. The results in Figure 9(a) show that PSCS and RPCO lead to about the same number of recirculations per packet for symmetric, disjoint, generating port clusters.



(a) Traffic sampled from four generating port clusters of size 8 with different port correlation p.

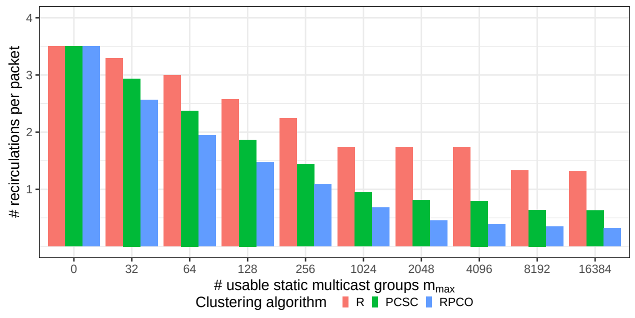


(b) Traffic sampled from four clusters of size 12, 10, 6, 4 with port correlation $p = 0.9$.

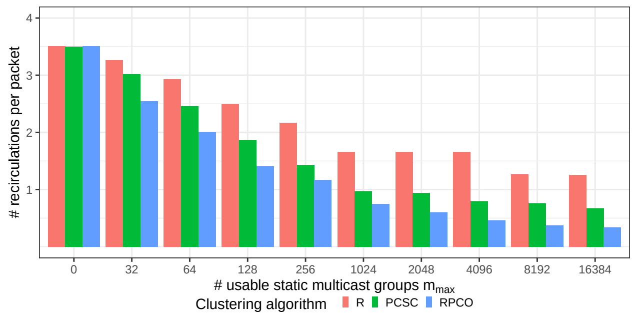
Figure 9: Impact of port clustering methods and number m_{max} of usable, static multicast groups on the average number of recirculations per packet; multicast traffic is sampled from disjoint generating port clusters.

In the following, we choose port correlation $p = 0.9$ as this generates sufficiently correlated multicast traffic with substantial port deviation from the generating port clusters.

b) Asymmetric Generating Port Clusters: We consider four asymmetric, disjoint, generating port clusters of size 12, 10, 6, 4: $C_1 = \{1, .., 12\}$, $C_2 = \{13, .., 22\}$, $C_3 = \{23, .., 28\}$, and $C_4 = \{29, .., 32\}$. If used for configuration, they require



(a) Traffic sampled from six clusters of size 8.



(b) Traffic sampled from six clusters of size 12, 10, 8, 8, 6, 4.

Figure 10: Impact of port clustering methods and number m_{max} of usable, static multicast groups on the average number of recirculations per packet; multicast traffic is sampled from overlapping, generating port clusters with port correlation $p = 0.9$.

$(2^{12}-12-1)+(2^{10}-10-1)+(2^{6}-6-1)+(2^{4}-4-1) = 5164$ static multicast groups.

Figure 9(b) illustrates the average number of recirculations per packet for port correlation $p = 0.9$. Again, more usable static multicast groups cause fewer recirculations. We now observe that RPCO reduces the average number of recirculations to lower numbers than PCSC, in particular for $m_{max} \leq 4096$. For larger m_{max}, PCSC and RPCO lead to almost equal results. This is in line with the design goal of RPCO: it makes better use of a limited number of static multicast groups than PCSC by proposing unequal-size port clusters. For $m_{max} = 64$, PCSC causes 3 recirculations per packet while RPCO causes only 2. For port correlation $p = 0.99$, which is not shown in the figure, both PCSC and RPCO reduce the average number of recirculations to almost zero for $m_{max} \geq 8192$.

2) Multicast Traffic Generated from Overlapping Port Clusters: We study the performance of the presented clustering algorithms for overlapping, generating port clusters.

a) Symmetric Generating Port Clusters: We consider six overlapping, generating port clusters of size 8: $C_1 = \{1, .., 8\}$, $C_2 = \{6, .., 13\}$, $C_3 = \{11, .., 18\}$, $C_4 = \{17, .., 24\}$, $C_5 = \{22, .., 29\}$, and $C_6 = \{28, .., 32, 1, .., 3\}$. Configuring them as port clusters requires $6 \cdot (2^8 - 8 - 1) - 4 \cdot (2^3 - 3 - 1) - 2 \cdot (2^2 - 2 - 1) = 1464$ static multicast groups.

Figure 10(a) indicates the average number of recirculations per packet for port correlation $p = 0.9$. Here, PCSC outperforms RPC, and RPCO outperforms PCSC for any number $m_{max} > 0$ of usable static multicast groups. While PCSC computes only disjoint port clusters, RPCO may yield overlapping port clusters. This can lead to fewer recirculations when frequently observed port groups of packets are partly overlapping. For port correlation $p = 0.99$, which is not shown in the figure, only RPCO reduces the average number of recirculations to almost zero for $m_{max} \geq 2048$.

b) Asymmetric Generating Port Clusters: We consider six overlapping, generating port clusters of size 12, 10, 8, 8, 6, 4: $C_1 = \{1, .., 12\}$, $C_2 = \{27, .., 32, 1, .., 4\}$, $C_3 = \{9, .., 16\}$, $C_4 = \{22, .., 29\}$, $C_5 = \{18, .., 23\}$, and $C_6 = \{16, .., 19\}$. Configuring them as port clusters requires 5630 static multicast groups.

Figure 10(b) illustrates the average number of recirculations per packet for port correlation $p = 0.9$. The results are very similar to those in Figure 10(a), only a few recirculations more are required. That means, PCSC clearly outpeforms RPC, and RPCO outperforms PCSC. For $p = 0.99$ and $m_{max} \geq 8192$, which is not shown here, RPCO even reduces the average number of recirculations to almost zero. That is, it is able to find optimal clusters for configuration even under challenging conditions (overlapping, unequal-size, generating port clusters).

C. Runtime

The presented clustering algorithms, especially RPCO, seem rather complex at first glance. We measure the runtime of the presented algorithms for the evaluation in Section VIII-B2b. The experiments are executed on a 2022 Mac Studio with M1 Max and 32 GB of RAM. Figure 11 compiles the results.

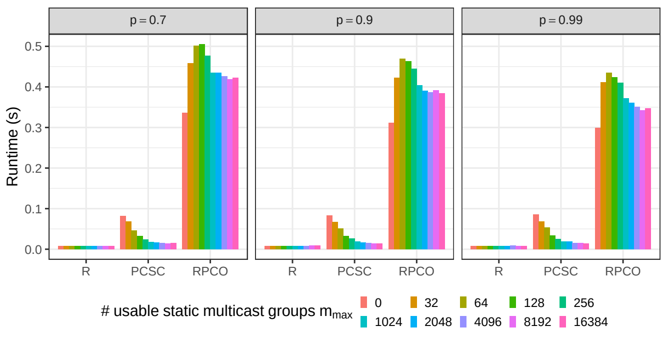


Figure 11: Average runtime in seconds for the clustering algorithms RPC, PCSC, and RPCO while performing experiments for Section VIII-B2b.

Random Port Clustering (RPC) has the shortest runtime with at most 9 ms. It partitions all ports into equal-size clusters and its runtime is therefore independent of port correlation p. PCSC reveals the second lowest runtime with up to 86 ms. It calls the Spectral Clustering subroutine at most n_p times where n_p is the number of ports. PCSC's runtime decreases with increasing m_{max} because larger values of m_{max} lead to fewer subroutine calls (return leaves the loop in Algorithm 1). RPCO has the longest runtime with up to 527 ms. It also performs n_p iteration steps but may call Spectral Clustering multiple times within a single iteration step. Its runtime primarily depends of the number of recursive calls. With decreasing p, RPCO's runtime decreases. Lower values of p lead to more uncorrelated packets, which leads to a blurred graph structure in the sense of more homogeneous edge weights. The Spectral Clustering subroutine tends to return larger clusters on a blurred graph. When not all clusters can be built, RPCO recursively re-clusters them. This is more likely with a blurred graph structure than with a sharp graph structure, i.e., a higher correlation between packets.

Although RPCO has the longest runtime, RPCO can be carried out sufficiently fast so that it can be well applied in practice as configured port clusters may be adapted rather on the time scale of minutes than seconds.

D. Scalability

In the following, we discuss the scalability properties of the presented mechanisms, i.e., how they behave in larger networks. First, the presented clustering algorithms leverage only local information for optimization, i.e., they only require sampled packets from a switch. Therefore, the optimization is preferably done by a controller running on the switch itself, which eliminates the need for additional control plane traffic in the network. Second, the used graph embedding (Section VII-C1) has a constant size per switch, i.e., it scales linearly with the number of switch ports. Therefore, the runtime is bounded by a small constant for a realistic number of maximal ports of a switch. As a consequence, the presented mechanisms are highly scalable and also suited for large networks.

IX. Experimental Performance Evaluation

In this section we perform experiments in a hardware testbed to demonstrate the practical feasibility of the proposed concepts and to validate the theoretical results from Section VIII. First, we explain the concept and the testbed setup. Then, we describe the performed experiments.

A. Concept

Figure 12 illustrates the concept for the hardware testbed.

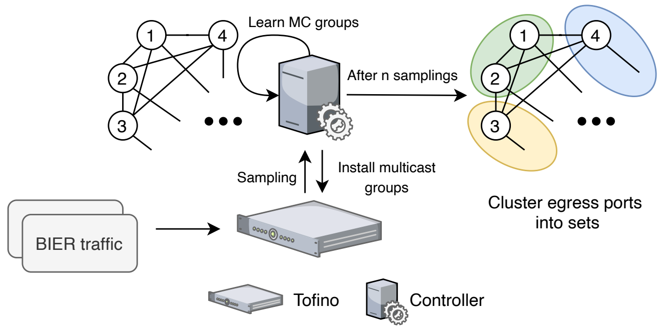


Figure 12: Concept for the hardware evaluation.

The Tofino [49], a P4-programmable switching ASIC, is the core of the hardware testbed. We utilize a Tofino-based Edgecore Wedge 100BF-32X switch [50] with 32 100 Gbit/s ports that runs the adapted BIER implementation as described in Section VI. BIER traffic is sampled at the Tofino with a rate of 0.1%, i.e., every 1000^{th} BIER packet. Sampled packets are sent to the controller and used for the graph embedding as described in Section VII. For 100 Gbit/s incoming multicast traffic, this amounts to 100 Mbit/s which can be efficiently handled by the controller. Alternatively, the number of sampled packets can also be limited through a Meter[9] instance. After 2^{10} samples, the controller applies the optimization heuristic and installs the static multicast groups of the configured port clusters. We measure the average recirculation traffic on the Tofino to assess the effectiveness of the presented optimization heuristics. To that end, packets on the recirculation port are cloned to a separate end host that measures the incoming bandwidth which equals the rate of the recirculation traffic.

B. Traffic Generation

Generating UDP traffic at high rate according to a given distribtion is a difficult task. We leverage Iperf [53] to generate homogeneous UDP traffic on an end host. It is sent to the Tofino which adapts it according to a specified distribution of BIER headers. When the Tofino receives a UDP packet generated by Iperf, it generates a random number between 0 and $2^w - 1$, where w is a parameter of the random extern on Tofino that generates a random number between 0 and $2^w - 1$. The generated random number is then used as index to a match-action table that maps the random number to a BIER header (see Figure 13). Then, the header of the UDP packet is substituted by the BIER header indicated in the table. Thereby, a UDP packet stream with any distribution of BIER headers can be generated.

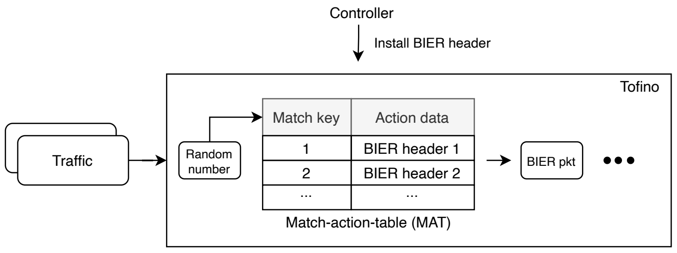


Figure 13: A match-action table is used to turn homogeneous UDP traffic into BIER traffic with headers following a desired distribution.

The match-action tables is populated a priori by a controller which has sampled 2^w BIER headers according to the traffic model in Section VIII-A1 for a given set of generating port clusters and a port correlation p. As a result, the Tofino turns homogeneous UDP traffic into BIER traffic whose headers follow a desired distribution.

[9]Intel Tofino supports 3-color metering as described in [52].

C. Experiment

We validate our hardware implementation by conducting the same experiments as in Section VIII-B2b. Thus, the traffic model consists of six overlapping generating port clusters of size 12, 10, 8, 8, 6, and 4. We choose port correlation $p = 0.9$, and use $w = 14$ to install 2^w sampled BIER headers of that distribution in the match-action table on the Tofino. We generate 5 Gbit/s UDP traffic via Iperf and send it to the Tofino which turns it into BIER traffic with the desired header distribution. We perform 5 runs per experiment and report average values.

The controller samples the BIER traffic and computes optimized port clusters for configuration on the Tofino. Thereby, different port clustering methods and different numbers m_{max} of usable static multicast groups are considered. Figure 14 shows the average recirculation traffic in Gbit/s.

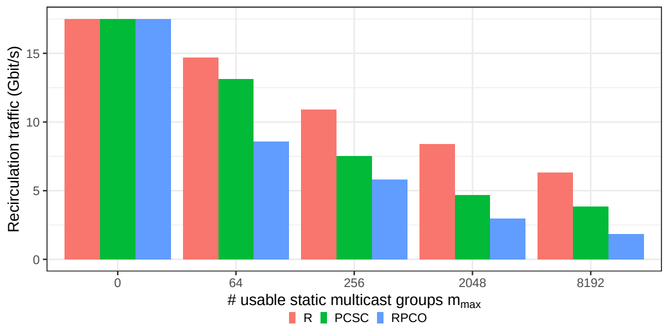


Figure 14: Average recirculation traffic for RPC, PCSC and RPCO and different numbers m_{max} of usable static multicast groups; the traffic model has six overlapping generating port clusters and port correlation $p = 0.9$; the results are to be compared with those in Figure 10(b).

If no static multicast group is available ($m_{max} = 0$), efficient BIER forwarding is essentially disabled, and the observed behaviour is the same as the one for simple BIER forwarding. Therefore, packets with 4.5 ports on average require 3.5 recirculations on average, which results in $3.5 \cdot 5$ Gbit/s $= 17.5$ Gbit/s recirculation traffic. This closely matches the results of Section VIII-B2b. An increasing number m_{max} of usable static multicast groups decreases the average number of recirculations per packet and therefore the recirculation traffic. Again, PCSC and RPCO clearly outperform RPC and RPCO performs better than PCSC (for $m_{max} > 0$). In fact, for $m_{max} = 8192$, RPCO reduces the recirculation traffic by 71% compared to RPC and 52% compared to PCSC. The experimental results in Figure 14 are in line with the simulation results in Figure 10(b) as they show the same proportions.

We performed this experiment with only 5 Gbit/s incoming traffic due to the lack of a fast generator for contant bit rate traffic. However, efficient BIER forwarding runs at line rate at the Tofino[10], i.e., it is capable of handling 32 × 100 Gbit/s incoming traffic.

[10]Every P4 program that compiles for the Tofino runs at line rate.

X. Conclusion

Bit Index Explicit Replication (BIER) forwards multicast traffic without signalling and states within BIER domains. Thereby, it greatly improves scalability for multicast in core networks. However, a simple implementation of that concept implies iterative packet transmission which requires additional processing capatity [2] on a single switch. In this paper we presented efficient BIER forwarding with static multicast groups such that a BIER packet can be sent to multiple next-hops in a single pipeline iteration. To that end, we configure port clusters on the switch and install all combinations of ports within each port cluster as static multicast group. Simple match-action operations choose the appropriate port clusters and therein the right static multicast group so that packets are transmitted to multiple next-hops in a single iteration step. As a result, a BIER packet can be processed in high-speed with a single or at most a few iteration steps. We demonstrated by simulation that randomly selected disjoint equal-size configured port clusters can decrease the required recirculations by 90% with only 1024 static multicast groups on a 32 port switch with 32 next-hops (Section VI-C) compared to simple iterative BIER forwarding. Further, we presented port clustering algorithms based on Spectral Clustering which learn the current BIER traffic pattern and compute port clusters for configuration. Recursive Port Clustering with Overlap (RPCO) reduces the required recirculations by up to 96% compared to randomly selected port clusters (Section VIII). We implemented efficient BIER forwarding on the Edgecore Wedge 100BF-32X, a 32 100 Gbit/s port high-performance P4 switch, and validated the simulation results in a hardware testbed.

The work comes with a few byproducts. We developed efficient BIER forwarding for data plane programming with the Tofino ASIC. Other switch architectures will also face the challenge to determine outgoing ports of a BIER packet with little effort and can benefit from the presented algorithms. We proposed a traffic model for the outgoing ports of multicast traffic on a switch for evaluation purposes. Future work may validate that traffic model based on measured data. Finally, we developed a simple method for data plane programming to modify traffic such that its headers correspond to a specific distribution. This may also be useful in other experimental work.

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.*, "Hardware-Based Evaluation of Scalable and Resilient Multicast With BIER in P4," *IEEE Access*, vol. 9, 2021.

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

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

[5] M. Shahbaz *et al.*, "Elmo: Source Routed Multicast for Public Clouds," in *ACM SIGCOMM*, 2019.

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

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

[8] X. Li *et al.*, "Scaling IP Multicast on Datacenter Topologies," in *ACM CoNEXT*, 2013.

[9] X. Zhang *et al.*, "A Centralized Optimization Solution for Application Layer Multicast Tree," *IEEE Transactions on Network and Service Management (TNSM)*, vol. 14, 2017.

[10] K. Mokhtarian *et al.*, "Minimum-delay multicast algorithms for mesh overlays," *IEEE/ACM Transactions on Networking*, vol. 23, 2015.

[11] S.-H. Shen, "Efficient SVC Multicast Streaming for Video Conferencing With SDN Control," *IEEE Transactions on Network and Service Management (TNSM)*, vol. 16, 2019.

[12] M. A. Kaafar *et al.*, "A Locating-First Approach for Scalable Overlay Multicast," in *IEEE INFOCOM*, 2006.

[13] R. Boivie, N. Feldman, and C. Metz, "Small Group Multicast: A New Solution for Multicasting on the Internet," *IEEE Internet Computing*, vol. 4, 2000.

[14] A. Boudani and B. Cousin, "SEM: A New Small Group Multicast Routing Protocol," in *International Conference on Telecommunications (ICT)*, 2003.

[15] 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 (JSAC)*, vol. 31, 2013.

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

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

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

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

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

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

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

[23] S. H. Shen, L. H. Huang, D. N. Yang, and W. T. Chen, "Reliable Multicast Routing for Software-Defined Networks," in *IEEE INFOCOM*, 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] 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.

[28] 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.

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

[30] 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.

[31] M. J. Reed *et al.*, "Stateless Multicast Switching in Software Defined Networks," in *IEEE International Conference on Communications (ICC)*, 2016.

[32] 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.

[33] D. Merling *et al.*, "P4-Based Implementation of BIER and BIER-FRR for Scalable and Resilient Multicast," *Journal of Network and Computer Applications (JNCA)*, vol. 169, 2020.

[34] p4lang, *Behavioral-model*, https://github.com/p4lang/behavioral-model, Accessed: 01.28.2021, 2021.

[35] A. Bas, *BMv2 Throughput*, https://github.com/p4lang/behavioral-model/issues/537\#issuecomment-360537441, Jan. 2018.

[36] 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.

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

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

[39] 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.

[40] T. Eckert and B. Xu, *Carrier Grade Minimalist Multicast (CGM2) using Bit Index Explicit Replication (BIER) with Recursive BitString Structure (RBS) Addresses*, https://datatracker.ietf.org/doc/html/draft-eckert-bier-cgm2-rbs-01, Feb. 2022.

[41] 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.

[42] J. B. MacQueen, "Some Methods for Classification and Analysis of MultiVariate Observations," in *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1, University of California Press, 1967.

[43] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, 1996.

[44] U. v. Luxburg, "A Tutorial on Spectral Clustering," *Statistics and Computing*, vol. 17, 2007.

[45] H. Chen, M. McBride, S. Lindner, *et al.*, "BIER Fast ReRoute," Internet Engineering Task Force, Internet-Draft draft-chen-bier-frr-04, Jan. 2022, Work in Progress, 31 pp. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-chen-bier-frr-04.

[46] D. Merling, S. Lindner, and M. Menth, "Comparison of Fast-Reroute Mechanisms for BIER-Based IP Multicast," in *2020 Seventh International Conference on Software Defined Systems (SDS)*, 2020.

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

[48] F. Hauser, M. Haeberle, D. Merling, *et al.*, "A Survey on Data Plane Programming with P4: Fundamentals, Advances, and Applied Research," *currently under submission*, 2021.

[49] Intel, *Intel Tofino*, https://www.intel.de/content/www/de/de/products/network-io/programmable-ethernet-switch/tofino-series.html, 2021.

[50] Edge-Core Networks, *Wedge100BF-32X/65X Switch*, https://www.edge-core.com/_upload/images/2021-048-DCS800_Wedge100BF-32X-DS-R08.pdf, 2021.

[51] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*. Springer, Berlin, Germany, 2004.

[52] J. Heinanen and R. Guerin, *RFC2698: A Two Rate Three Color Marker*, https://www.rfc-editor.org/info/rfc2698, Sep. 1999.

[53] iperf2 team, *iperf*, https://iperf.fr.