IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 24, 2021, accepted December 1, 2021, date of publication December 20, 2021,
date of current version December 30, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3136816

Ernie: Scalable Load-Balanced Multicast Source Routing for Cloud Data Centers

JARALLAH ALQAHTANI[1,2], BECHIR HAMDAOUI[1], (Senior Member, IEEE), AND RAMI LANGAR[3,4]

[1]School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR 97331, USA
[2]College of Computer Science and Information Systems, Najran University, Najran 55461, Saudi Arabia
[3]LIGM CNRS-UMR 8049, University Gustave Eiffel, 77420 Marne-la-Vallée, France
[4]Software and IT Engineering Department, ÉTS, Montréal, QC H3C 1K3, Canada

Corresponding author: Jarallah Alqahtani (alqahtaj@eecs.oregonstate.edu)

This work was supported in part by National Science Foundation under Award 2003273.

ABSTRACT Nowadays, most applications hosted on public cloud data centers (DCs) disseminate data from a single source to a group of receivers for service deployment, data replication, software upgrade, etc. For such one-to-many data communication paradigm, multicast routing is the natural choice as it reduces network traffic and improves application throughput. Unfortunately, recent approaches adopting IP multicast routing suffer from scalability and load balancing issues, and do not scale well with the number of supported multicast groups when used for cloud DC networks. Furthermore, IP multicast does not exploit the topological properties of DCs, such as the presence of multiple parallel paths between end hosts. Despite the recent efforts aimed at addressing these challenges, there is still a need for multicast routing protocol designs that are both scalable and load-balancing aware. This paper proposes Ernie, a scalable load-balanced multicast source routing for large-scale DCs. At its heart, Ernie further exploits DC network structural properties and switch programmability capabilities to encode and organize multicast group information inside packets in a way that minimizes downstream header sizes significantly, thereby reducing overall network traffic. Additionally, Ernie introduces an efficient load balancing strategy, where multicast traffic is adequately distributed at downstream layers. To study the effectiveness of Ernie, we extensively evaluate Ernie's scalability behavior (i.e., switch memory, packet size overheads, and CPU overheads), and load balancing ability through a mix of simulation and analysis of its performances. For example, experiments of large-scale DCs with 27k+ servers show that Ernie requires a downstream header sizes that are 10× smaller than those needed under state-of-the-art schemes while keeping end-host overheads at low levels. Our simulation results also indicate that at highly congested links, Ernie can achieve a better multicast load balancing than other existing schemes.

INDEX TERMS Cloud data center networks, multicast routing, multicast scalability, multi-tenancy, network virtualization.

I. INTRODUCTION

The past decade has witnessed a rapid boom of cloud computing services. Large cloud service providers, such as Amazon AWS [1], Microsoft Azure [2] and Google Cloud Platform [3], host hundreds of thousands of tenants, each of which possibly running hundreds of applications. Many of these applications [4]–[6] are rife with one-to-many communication patterns, making multicast the choice for supporting this type of communication, as it greatly conserves network bandwidth and reduces server overhead. IP multicast can meet these requirements; however, it gives rise to two major challenges: scalability and load balancing. Scalability limitations arise in both the control and the data planes of the network. For example, switches can only support limited multicast states in their forwarding tables (e.g., thousands to a few tens of thousands [7]). Furthermore, today's data center (DC) networks operate under a single administrative domain and no longer require decentralized protocols like PIM [8] and IGMP [9]. In terms of load balancing, IP multicast-based protocols like PIM are principally designed for arbitrary network topologies and do not utilize topological structure of modern DC networks to take full advantage of the multi-path property. For instance, such protocols usually choose a

The associate editor coordinating the review of this manuscript and approving it for publication was Zhenzhou Tang.

random single core switch on a Fat Tree as the rendezvous point (RP) to build the multicast tree. When multiple groups use the same core switch simultaneously, traffic bursts and network congestion may occur. Unfortunately, these two keys obstacles of IP-multicast have forced some cloud providers to use application-based or overlay-multicast [10] approaches as an alternative method. In such approaches, where all packet replications occur at the host instead of switches, bandwidth and end-host CPU overheads are inflated.

SDN-based approaches [11], [12], on the other hand, have strived to handle these needs, but still present many challenges. For example, network switch resources are exhausted due to large numbers of flow-table entries, as well as high numbers of switch entry updates. Another challenge is the high computation complexity when maintaining real-time congestion of all network links to balance the multicast traffic [13], [14]. This impedes the deployment potential in large-scale networks of these approaches and as such, it is suitable only for small two-tier Leaf-Spine topologies.

Recently proposed source-routed multicast schemes for cloud DCs, such as Elmo [15] and Bert [16], address the scalability limitations and are shown to scale well with millions of multicast groups. They do so by exploiting both the DC network topology symmetry and hardware switch programmability to efficiently encode multicast routing information inside packets. However, these schemes still have some key shortcomings. For instance, Elmo [15] incurs extra overhead when the multicast group is large in size or dispersed across the network. Although Bert [16] aims to alleviate network overhead incurred by Elmo, it imposes bandwidth and end-host CPU overheads when the number of clusters (packet replications at the source) is large. Furthermore, these schemes neglected the multicast traffic load balancing and relied on underlying multipathing protocols (e.g., ECMP [17]). In fact, these load balancing protocols are mainly designed for unicast traffic and are not suitable for multicast.

Given the above inefficiencies, we revisit the multicast in DCs and propose Ernie, scalable, load-balanced source-routed scheme for large-scale data center networks. Ernie reconsiders possible solutions by further leveraging the topological properties of modern DC architectures. It scales to much larger numbers of multicast groups, while minimizing network overhead (i.e., switch memory and packet size overheads) and with an eye towards downlinks loads (highly congested links). Ernie does so by leveraging the structural property of multi-rooted Clos topology as well as the programmability and configurability of DC switches to encode forwarding headers inside multicast packets to substantially compact downstream packet headers. More specifically, this paper makes the following contributions:

- We propose a novel multicast routing technique that scales well with both the number and size of multicast groups, while keeping network overhead (i.e., switch memory and packet size overheads) minimal. Specifically, Ernie appropriately constructs and organizes multicast header information inside packets in a manner that minimizes downstream network traffic.
- We introduce an effective multicast traffic load balancing technique for downstream links that assigns multicast groups to core switches in a way that ensures the evenness of load distribution across the downstream links.
- We extensively evaluate the proposed techniques in terms of scalability and load balancing ability through a series of simulation experiments. In multi-tenant datacenters with 27k servers, our experiments show that the proposed techniques require a downstream header size that is 10× and 3× smaller than that needed under Elmo and Bert, respectively, and achieve between 25 and 65% load balancing improvement over other schemes for highly congested network links.

The rest of this paper is organized as follows. In Section II, we discuss related works. Section III briefly describes the network architecture, techniques of modern DCs, and the limitations of prior related state-of-the art works. We present Ernie, the proposed multicast routing scheme, in Section IV. In Section V, we study and evaluate the performances of Ernie and compare them with those achieved under existing schemes. Finally, we conclude the paper in Section VI.

II. RELATED WORKS

The field of large-scale DCs has recently been the focus of many researchers addressing various DC challenges, including, but not limited to, resource allocation [23]–[26], traffic load balancing [27]–[30], security and privacy [31]–[35], and power consumption [36]–[39]. Over the years, multicast also has been the focus of a large body of research to address issues related to scalability [21], [40], reliability [41]–[43], security [44], and congestion control [45], [46]. In this section, we briefly discuss related works, particularly those related to DC multicast scalability and load balancing. Multicast routing for wide-area networks is different in significant ways from that for DC networks. Thus, we restrict our focus in this paper on related works for DC multicast.

A. SCALABILITY

Scalability of multicast routing in DC networks has been a real concern and attracted considerable attention in the research community. For instance, SDN-based multicast solutions [11], [12] suffer from a high number of switch updates as well as limited switch group-table capacities. For example, in [11], a centralized controller partitions the address space, and local address aggregation is implemented when the table space in switches is not enough. This approach suffers from exhausting network switch resources with a large number of flow-table entries, as well as a high number of switch entry updates. On the other hand, several other proposals seek to increase the number of multicast groups that can be supported by encoding forwarding states inside multicast packets, as in [21], [22], [47], [48], which

TABLE 1. Summary and comparison between Ernie and prior DC multicast routing schemes. (*Switch size is 5K entries.)

	Scheme Type	Switch Storage Overhead*	Network Traffic Overhead	Scalability		Multicast Load Balancing	End-host Overhead (Packet Replication)
				# of Groups*	Group Size Limits		
Overlay Multicast [10]	Unicast	No	High	Huge (1M+)	No	Yes	High
IP Multicast	Decentralized	High	Minimal	Small (5K)	No	No	No
Miniforest [18]	Decentralized	High	Minimal	Small (5K)	No	Yes	No
iRP [13]	SDN	High	Minimal	Small (5K)	No	Yes	No
IP Multicast Scaling [11]	SDN	High / Medium with rule agg.	Minimal / Low with rule agg.	Medium(70K) / High (500K) with rule agg.	No	No	No
RDNA [19], COXcast [20]	Source-Routed	No	Medium	Huge (1M+)	Yes	No	No
LIPSIN [21], ESM [22]	Source-Routed	No	Medium	Huge (1M+)	Yes	No	No
Elmo [15]	Source-Routed	Medium / Low (with rule agg.)	Low / Medium (with rule agg.)	Huge (1M+)	No	No	No
Bert [16]	Source-Routed	No	Low	Huge (1M+)	No	No	Low
Ernie **(proposed scheme)**	Source-Routed	No	Low	Huge (1M+)	No	Yes	No

does so through the use of bloom filters. The overhead of these approaches arises from unnecessary traffic leakage (unnecessary multicast transmissions) due to high false positive forwarding. Moreover, these schemes support only small-sized groups (i.e., less than 100 receivers [49]). In [19], [20], division (modulo) operation is used to encode forwarding states inside the packets. In these approaches, a route between source and destination(s) is defined as the remainder of the division between a route-ID and switch-IDs. These approaches are only suitable for small DCs (micro data centers), and do not scale beyond a few tens of switches. Recent source-routed schemes, like Elmo [15] and Bert [16], address both control and data planes scalability limitations by exploiting DC topology symmetry as well as hardware switch reconfigurability. Although, these are shown to scale well with millions of multicast groups, they still present some major issues. For instance, Elmo [15] incurs some overhead when the multicast group is large in size or dispersed across the network. It behaves poorly under these scenarios by increasing network overhead (i.e., switch memory and packet size overheads). On the other hand, Bert [16] imposes bandwidth and end-host CPU overheads when the number of clusters (packet replications at the source) is large.

B. LOAD BALANCING

Abundant research on DC traffic load balancing has mostly focused on unicast traffic [27]–[29], [50]–[53]. These approaches mainly rely on TCP characteristics (e.g., SYN/ACK and ECN), and are not suited for multicast. Moreover, there has been scarce research efforts on multicast load balancing traffic in DCs. For example, in [13], [14], [54], multicast traffic load balancing is done by maintaining bandwidth information for all the paths between all ToR switches. These approaches work well only in small 2-tier, leaf-spine topologies and do not scale well for large 3-tier, Clos topologies, which are widely deployed in production DCs. In large 3-tier topologies, collecting real-time congestion information for all links is quite expensive to implement. The dual-structure Multicast (DuSM) proposed in [12] classifies multicast groups into two kinds of multicast flows—Mice and Elephant flows—based on a threshold flow rate of the multicast group. DuSM forwards small flows using unicast rules on switches and uses multiple trees for large flows. Unfortunately, this approach gives rise to other challenges such as sacrificing network bandwidth. Miniforest [18], on the other hand, is a distributed multicast framework that aims to balance multicast traffic by evenly assigning multicast groups into root switches. Unfortunately, as discussed in Section IV-B1, inadequate assignment to root switches can lead to inefficient load balancing of other network layers. Moreover, continuous contact between Administrative Nodes (NA) in each pod to update leave/join requests, routing table, etc., may increase the network overhead, especially in large scale topologies.

To the best of our knowledge,Ernie is the first source-routed scheme that takes into account both scalability and load balancing aspects of multicast traffic in large scale DCs. Ernie addresses the multicast scalability limitations while keeping both the network and end-host overheads at low levels. In Table 1, we present the drawbacks of prior solutions, thus motivating our design of Ernie.

III. BACKGROUND

A. DC TOPOLOGIES

Large-scale production DCs typically are multi-rooted tree-based topologies (e.g., fat-tree [55] and its variants [56]–[58]). These types of topologies are highly connected and scalable to support a massive number of servers and applications with high demands. The servers are tree leaves, which are physically connected to (leaf/edge) switches, called Top-of-Rack (ToR) switches. General fat-tree topologies organize the network into three layers of switches, leaf, spine, and

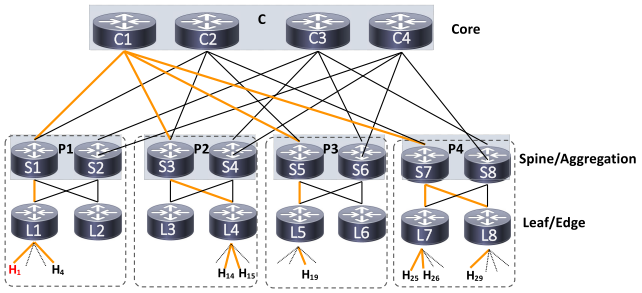


FIGURE 1. A three-tier multicast Clos tree topology with four pods. In this Example, there are 4 hosts under each leaf switch. H_1 is the multicast source, and H_4, H_{14}, H_{15}, H_{19}, H_{25}, H_{26}, and H_{29} are the destinations of the multicast group.

core, from bottom to top, as shown in Fig 1. The lower two layers are separated into k pods, with each pod containing $k/2$ leaf (aka edge) switches and $k/2$ spine (aka aggregation), which form a complete bipartite graph in between. There are $k^2/4$ core switches, constituting the top/root layer, each of which is connected to each of the k pods. These types of topologies provide large numbers of parallel paths to support high bandwidth, low latency, and non-blocking connectivity among servers.

B. MULTI-TENANT DCs

Multi-tenancy is one of the key features of cloud DCs. In Multi-tenant DCs (like Microsoft Azure [2], Amazon Web Services (AWS) [1], and Google Cloud Platform [3]), a fraction of computing resources (e.g., CPUs, memory, storage, and network) are rented to customers/tenants (e.g., commercial, government, individual) by means of virtualization technology. On the other hand, in multi-tenant cloud computing, infrastructures, applications, and databases are shared among all tenants. By moving towards multi-tenant DCs, tenants can lower their operational costs of maintaining private infrastructure, meet scalability demands with changing workload, and withstand disasters. For example, Netflix, the world's leading online video streaming service provider, uses AWS for nearly all its computing and storage needs [59].

C. VIRTUALIZATION IN DCs

In multi-tenant DCs, computing and network resources are virtualized. Typically, this is done by using software or firmware called a hypervisor [60]. The hypervisor allows one host computer to support multiple guest VMs by virtually sharing its resources, such as memory and processing. A virtual switch in the hypervisor, called the vswitch [61], manages routing traffic between VMs on a single host, and between those VMs and the network at large. Moreover, these DCs employ tunneling protocols (like VXLAN [62]) to guarantee resource isolation and fair share of network resources among tenants.

D. PROGRAMMABLE SWITCHES

Emerging programmable switch ASICs (e.g., Barefoot Tofino [63]) render flexible packet parsing and header manipulation through reconfigurable match-action pipelines that allow network operators to customize the behavior of physical switches. Network operators can program these switches using high-level network-specific languages like P4 [64]. P4, a language for Programming Protocol Independent Packet Processors, is a recent innovation providing an abstract model suitable for programming the network data plane.

E. SOURCE-ROUTED MULTICAST

In multicast source-routing approaches, a multicast tree is encoded into the header of each packet, and switches can read this information to make forwarding decisions. Recent source-routed schemes [15], [16] address both control and data planes scalability limitations by exploiting DC topology symmetry as well as hardware switch reconfigurability. Topology symmetry of DC networks (e.g. Fat-tree) is utilized to efficiently compact forwarding header size whereas emerging programmable switches are exploited to parse and process packets header efficiently. By doing so, the need for network switches to store routing information is minimized, and the burden on the controller is alleviated. Now we present two recently proposed source-routed multicast routing schemes:

1) ELMO

Elmo [15] primarily focuses on how to efficiently encode and compact a multicast forwarding information in the packet header. A multicast forwarding information is encoded in a packet header as a list of packet rules (p-rules for short). Exploiting the nature of DC networks topology, e.g. most servers are within five hops of each other, packet header consists of five p-rules, one for each hop. Each p-rule is comprised of set of output ports encoded as a bitmap that network switches use to forward the packet. Each multicast packet's journey can be explained in two phases: upstream path (from leaf switches up to core switches), and downstream path (from core switches down to leaf switches). In the upstream path, when the packet arrives at the upstream leaf switch, the switch forwards it to the given downstream ports as well as multipathing it to the upstream spine switch using an underlying multipath routing scheme; e.g. ECMP [17]. Using Fig. 2a for illustration, when leaf switch $L1$ receives the packet, it first removes its p-rules ($0001-M$) from the packet, and then forwards it to the host $H4$ as well as multipathing it to any spine switch (e.g $P1$). In the same manner, the upstream spine switches will forward the packet to the core switches. In the downstream path, the p-rules for the core, spine, and leaf switches consist of downstream ports, and switch IDs for spine, and leaf switches. A core switch replicates the packet and forwards it to each destination pod (spine switch), which in turn replicates the packet and forwards it down to the destination leaf switches. The leaf switches do the same to deliver the packet to the destination hosts.

2) BERT

Bert [16] overcomes Elmo's aforementioned limitations by alleviating traffic congestion at downstream paths as it

reduces both the packet header sizes and the number of extra packet transmissions, as well as eliminates switch memory utilization. It does so by adequately splitting multicast group members into multiple disjoint multicast clusters and encodes forwarding information for each cluster in the packet header. For example, as illustrated in Fig. 2b, Bert clusters the destination members into two clusters, $R1$ and $R2$. As for multicast tree encoding, Bert [16] follows Elmo [15].

IV. Ernie: THE PROPOSED MULTICAST Scheme

Ernie adequately encodes forwarding states inside each multicast packets with an eye towards downlinks traffic loads. In this section, we describe in detail Ernie, our proposed source-routed multicast scheme for DCs. We first describe how Ernie scales well, by capitalizing on the minimizing of packet header overhead at downstream paths. Then we introduce Ernie's load balancing technique that efficiently distributes multicast traffic across downstream (highly congested) layers.

A. SCALABILITY

1) MOTIVATION

An efficient scalable multicast source routing design should aim to minimize (1) network overhead (i.e., switch memory and traffic overhead) and (2) CPU computation. These two aims can be obtained by reducing packet header size and by avoiding packet replication at the source.

Regardless of the group size or placement strategy, clearly there is no packet replication when multicast packets traverse through the upstream path. Conversely, packet replications occur at the downstream path, and depend on group size and placement strategy. As a result, for multicast flows, the downstream path has a much heavier load than the upstream path. For example, in Fig. 1, the multicast flow uses two upstream links (from leaf to core switches) and 8 downstream links (from core to leaf switches). Without loss of generality, if the weight of this flow is 0.5, the flow will contribute 1 (i.e., 0.5×2) traffic load to the upstream path and 3.5 (i.e., 0.5×7) traffic load to the downstream path. Moreover, when using source-routed multicast, conveying a large header inside each multicast packet will tend to exacerbate traffic congestion at the downstream paths. In Elmo and Bert, each destination pod (downstream spine and leaf switches) receives unneeded information by receiving all/some other destination pods' routing information. For example, in Figs. 2a and 2b, the black p-rules received by each destination pads are unneeded. This unneeded information results in a large header that increases traffic overhead at downstream paths. If each pod is prevented from receiving other pods' p-rules, we can further reduce the header size and traffic overhead of a multicast group. Furthermore, we can reduce the number of bits needed to identify downstream spine and leaf switches. In Elmo and Bert, switch IDs are globally assigned to the downstream switches. This is because any downstream switch in one pod might receive other downstream switches' information

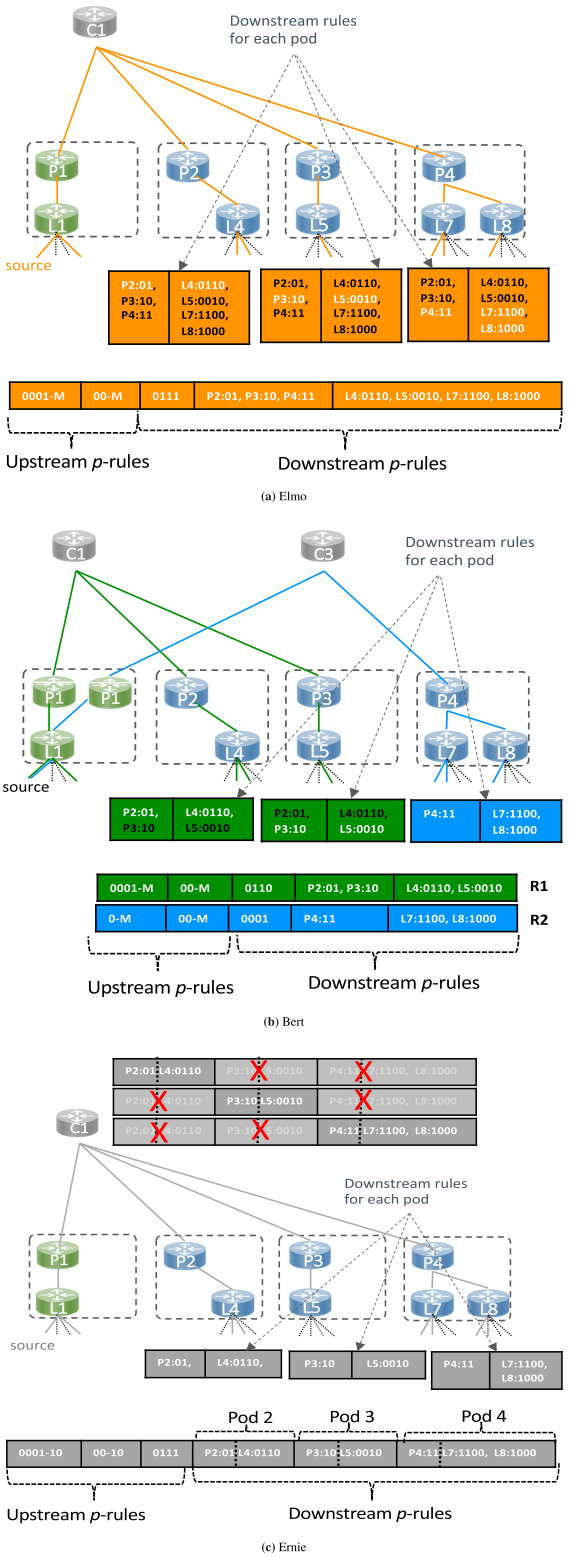


FIGURE 2. An example of multicast tree on a three-tier Clos topology with four pods. We use the same multicast tree in Fig.1. Unused switches and paths are eliminated for clarification. We show the header format of Elmo (a), Bert (b), and Ernie (c). Black downstream rules received by each pod are unneeded (redundant).

of other pods. By preventing this, we can locally (per pod) assign switch IDs. For example, in Figs. 2b and 2a, to identify

switches, Bert and Elmo use three bits for each of the spine and leaf switches. Hence, in Bert (Fig. 2b), the average size of downstream *p*-rules received by each destination pod is 22 bits whereas with Elmo (Fig. 2a) is 43 bits. On the other hand, in Fig. 2c only one bit is needed to identify each of the spine and leaf switches. Thus, the average size of downstream *p*-rules received by each destination pod is only 10 bits.

2) DESIGN CHOICE

One natural question that arises here is how do we achieve minimal header overhead in downstream paths? One way is through clustering as in Bert [16]. For example, a multicast group is clustered into a set of disjoint clusters such that each cluster contains only the members of one pod. However, as mentioned before, end hosts will have to pay a price for this by observing an increased CPU overhead. The proposed scheme, Ernie, efficiently reduces header size by further exploiting the structural property of 3-tier Clos DC network topologies. For example, inter-pod traffic must pass through core switches where all inter-pod packets are sent to the core (root) switches, and then routed down to the host destinations. In other words, core switches are the intersection point between the upstream and downstream paths. Ernie also exploits the programmable capability of the DC switches which allows to parse and manipulate packet header at line rate. Ernie, first, encodes and orders the downstream *p*-rules inside a packet by pods, and then chooses the core switches to handle redundancy at the downstream paths.

3) SYSTEM COMPONENTS

1) Controller: A network controller is a software that orchestrates network functions. In Ernie, for each multicast group, the controller is responsible for calculating the forwarding header. First, the controller calculates a multicast tree and encodes *p*-rules as a bitmap. Then, it installs these *p*-rules in the hypervisor of the multicast group source.
   Header format: In Ernie, Fig. 2c, multicast forwarding information is encoded as a sequence of *p*-rules. Each *p*-rule comprises of upstream and downstream ports encoded in bitmap format, with 1 meaning forward packet through corresponding port and 0 otherwise. Upstream *p*-rules are grouped by layers: upstream leaf, upstream spine, and core. Downstream *p*-rules, however, are grouped by pods instead of layers (e.g. Pod 2, Pod 3, and Pod 4). Furthermore, each pod downstream *p*-rules consist only of spine and leaf *p*-rules for the corresponding pod only. These *p*-rules consist of downstream ports and switch IDs. Organizing downstream *p*-rules in such way allows us to: 1) further reduce header size and traffic overhead at downstream path, and 2) facilitate header processing procedures at core switches.
2) Hypervisor Switch: A hypervisor switch, such as Open vSwitch (OVS), runs on each server and manages routing traffic between VMs on a single host, and between those VMs and the network at large. In Ernie, the hypervisor intercepts each multicast packet generated from multicast sources, and adds the *p*-rules to the packet header.
3) Network Switches: Unlike traditional switches, where the data plane is designed with fixed functionalities, programmable switches can be configured by network operators to enable and customize the functionality of the networking switches without additional hardware upgrades. Moreover, these switches are capable of processing packets at high speed. Ernie assumes that DCs deploy and run P4 [64] programmable switches ASICs (e.g., Barefoot Tofino [63]). When network switches receive a multicast packet, they parse, replicate (if needed), and forward the packet to the corresponding output ports. In Fig. 2c, when *leaf switch* $L1$ receives the multicast packet with *p*-rules ($0001 - 10$), it forwards the packet down to the fourth port (e.g.H_4) as well as up through first port to spine switch (e.g. $P1$). Unlike, Elmo and Bert, upstream ports are predetermined in Elmo due to load balancing awareness (to be explained later). Similarly, when the packet arrives to *spine switch* $P1$ with *p*-rules ($00 - 10$), $P1$ only forwards the packet up to the core switches (e.g. $C1$). When *core switch* $C1$ receives the packet with *p*-rule (0111), it first generates multiple copies of the packet (i.e. 3 copies), and then for each copy,
   - it removes other pods' *p*-rules from the header except the one that corresponds to the egress port. For example, in Fig. 2c, for the first copy, $C1$ keeps the *p*-rule (Pod 2), which corresponds to the second egress port, and removes the last two *p*-rules (Pod 3 and Pod 4), which correspond to the third and fourth ports, respectively.
   - it routes the packet to the corresponding egress port.

   As intended, each destination pod only receives its *p*-rules. For example, Pod 2 only receives its spine ($P2$) and leaf ($L4$) switches information. Each of these *p*-rules consists of downstream ports and switch IDs. Note that switch IDs here are locally assigned because there is no inter-pod switches' information received by the destination pod. The packet arriving at *downstream spine switches* (e.g. $P2$) is forwarded down to leaf switches (e.g. $L4$) using the *p*-rule: ($P2 : 01$). The *downstream leaf switches* do the same to deliver the packet to the destination hosts. For example, $L4$ forwards packet down through the second and third port (e.g. to H_{14} and H_{15}) using the *p*-rule: ($L4 : 0110$).

B. LOAD BALANCING

1) MOTIVATION

We use the example given in Fig. 3 to illustrate the impact of the lack of even distribution of multicast traffic in multi-rooted Clos topologies DCNs. Suppose there are two

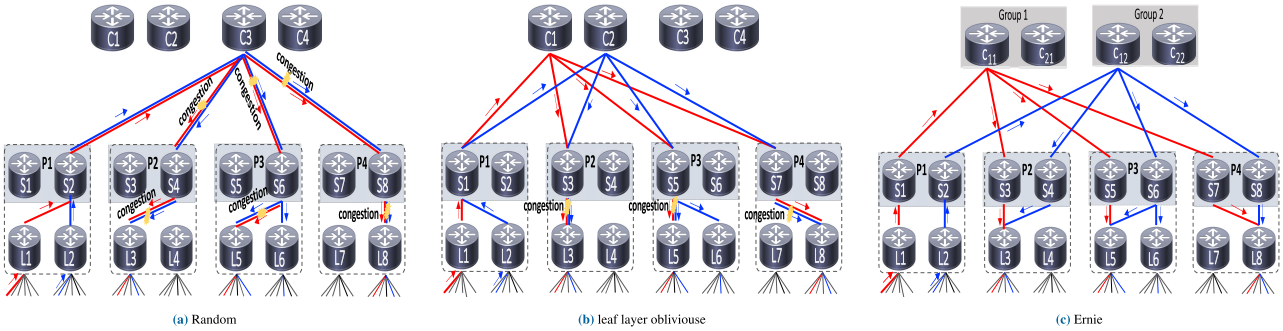


FIGURE 3. An example of two multicast groups (Red "R" and Blue "B") traffic distribution on a three-tier Clos topology with four pods. We show the strategy of Random (a), leaf layer oblivious. e.g., RR and Miniforest (b), and Ernie (c). Unused links omitted for clarity.

multicast groups, R (Red) and B (Blue), each having a source server and multiple destination servers randomly located across the network. Here, unevenly distributing multicast groups across the core switches results in jamming a small number of the core switches, yielding poorly unbalanced traffic loads. This situation arises when using: 1) PIM-SM [8], which randomly chooses a single core switch as the rendezvous point (RP) or 2) ECMP [17], which balances traffic loads among multiple upstream paths through multipathing. For instance, as shown in Fig 3a, due to the randomness nature, group R and group B can end up being routed through the same core switch (i.e., $C3$), thereby causeing heavy congestion among downstream paths, core and spine downlinks, respectively.

To improve load balancing, one may consider evenly assigning multicast groups to core switches. For example, multicast groups are randomly divided into several disjoint sets, and each set is routed through a specific root switch [18]. Another approach would be to assign multicast groups to core switches in a round-robin manner. Though, these approaches can locally balance downstream traffic (only from core to downstream spines), it may still introduce large traffic congestion in the downstream leaf switches. Referring to Fig.3b for illustration, suppose multicast groups R and B are assigned to two consecutive core switches $C1$ and $C2$, respectively. The downlinks of core switches for both groups are disjoint. However, because $C1$ and $C2$ are in the same core group (e.g., group 1), the traffic between downstream spine and leaf switches for both groups go through the same links.

2) DESIGN CHOICE

To overcome all these challenges, Ernie proposes an effective load balancing strategy that evenly distributes downstream multicast traffic by exploiting the symmetric property of fat-tree topologies. In fat-tree topologies, there are $(k/2)^2$ core switches that are divided into $k/2$ groups $j_1, j_2, \ldots, j_{k/2}$, each of which contains $k/2$ cores. Each core group connects to all pods through different spine switches. For example, in Fig.3c, the core switches in the first group are connected to the first spine switches in each pod, and the core switches in the second group are connected to the second spine switches in each pod, and so on. This property of the fat-tree topology ensures that the path between each core group and any leaf switch is disjoint. Hence, Ernie assigns multicast groups sequentially per core group such that no successive multicast groups go through the same core group. Let C_{ij} indicate the i^{th} core switch in the j^{th} core group, where $1 \leq i \leq k/2$ and $1 \leq j \leq k/2$. Generally, to assign multicast groups to core switches, Ernie iterates through all i^{th} switches in each group j in consecutive order. For example, in Fig.3c, multicast groups R and B are assigned to route through core switches C_{11} and C_{12}, respectively. To further explain this, suppose there are four multicast groups in this example; so based on Ernie's technique, the first multicast group goes through C_{11}, and the second to fourth multicast groups go through C_{12}, C_{21}, C_{22}, respectively. Intuitively, Ernie provides disjoint and periodic connections for successive multicast groups at all downstream layers. In general, in fat-tree topologies with k pods, when using Ernie's load balancing technique, there are no $(k/2)^i$ consecutive multicast groups go through the same downstream links in layer i, where $1 \leq i \leq 2$.

In fact, Ernie achieves a good load balancing while ensuring scalability at not cost. At its heart, to scale well, Ernie efficiently encodes the entire multicast tree in the packet. Unlike other source-routed schemes, Ernie also considers balancing downstream traffic (heavy load traffic) during the encoding process by routing the multicast packets through a proper core switch. Particularly, this process is done only when upstream switch ports (output ports of upstream leaf and spine) are encoded. Furthermore, Ernie's load balancing strategy does not need to continually pursue all the possible paths utilization information [13], [14], nor maintain the number of assigned multicast groups for each core switch [18].

C. KEY FEATURES OF Ernie

1) REDUCING TRAFFIC OVERHEAD

We have shown that for multicast traffic, downstream paths are much heavier and are always the main bottleneck of the network (due to high number of packet replications). In Ernie, header size for the downstream packet is minimal

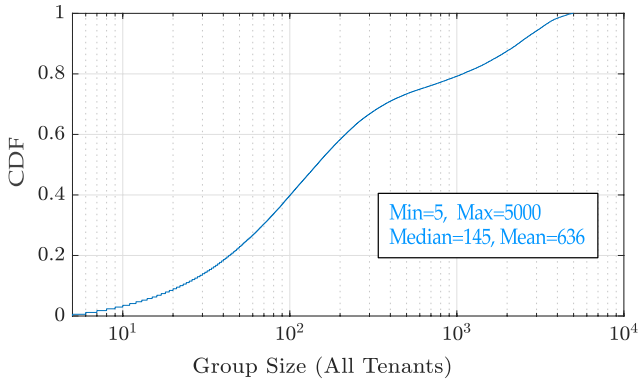


FIGURE 4. Group size follows random distribution for each tenant.

compared to other approaches (e.g. Elmo [15] and Bert [16]), as shown in Section IV-A1. As a result of reduced downstream header size, the overall traffic (upstream and downstream) incurred by Ernie is significantly reduced and is lesser than that incurred by Elmo or Bert.

2) REDUCING END-HOST CPU OVERHEAD

In Ernie, end hosts need to send one single copy of the packet to the network, regardless of multicast group members' size or distribution. Note that Elmo also does the same and reduces CPU overhead but at the price of increasing traffic overhead and switch memory usage. On the other hand, end hosts in Bert might send multiple copies of the packet based on the number of clusters, resulting in increased CPU overhead.

3) ELIMINATING SWITCH MEMORY UTILIZATION

In Elmo, in order to reduce traffic overhead caused by large header sizes in downstream paths, it uses switch memory to store some forwarding rules at downstream switches. Ernie, on the other hand, does not use switch memory.

4) BALANCING DOWNSTREAM MULTICAST TRAFFIC

In addition to minimizing high network and CPU overheads, Ernie offers good balancing of multicast traffic load in the downstream links. Ernie wisely and uniformly assigns multicast groups to core/root switches. To the best of our knowledge, Ernie is the first source-routed multicast scheme that addresses scalability while providing good load balancing of traffic in the downstream layers (highly congested layers).

V. PERFORMANCE EVALUATION

In this section, we assess the effectiveness of Ernie vis-a-vis of its ability to improve scalability and load balancing. We simulate cloud DC topologies, mimicking the experiment setup used in [15], [16] and consist of large fat-tree topologies with 48 pods, each having 576 hosts for a total of 27,648 hosts. We consider multi-tenant environments with varying number of tenants, number of VMs per tenant, VM placement strategies, and numbers and sizes of multicast groups.

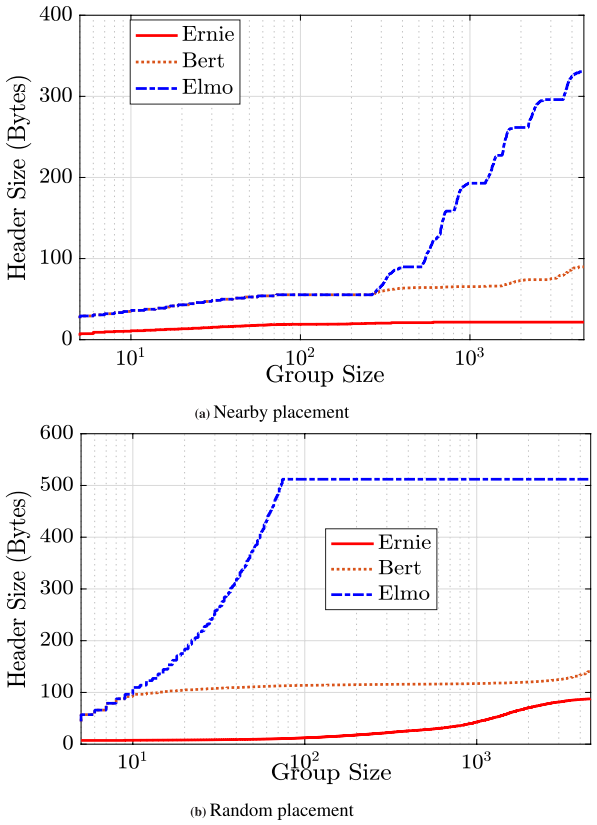


FIGURE 5. Header size overhead in the downstream path as function of group size. Group size varying from 5-5000 members.

The simulated cloud DC networks are populated by 3000 tenants, with the number of VMs per tenant being exponentially distributed between 10 and 5000, with mean=178. Each physical server can host up to 20 VMs and VMs belonging to the same tenant cannot be placed in the same server. We evaluate two types of VM placement policies: (i) a tenant's VMs are placed on hosts that are located next to one another (Nearby), and (ii) tenant's VMs are distributed across the network uniformly at random (Random). We generate 100K multicast groups, and the number of groups assigned to each tenant is proportional to the size of the tenant. Each tenant's group sizes are uniformly distributed between five (minimum group size) and the entire tenant size. Such a distribution is shown in Fig. 4, where the average group size is 646.

A. SCALABILITY ANALYSIS

We compare the scalability performance of Ernie to that of state-of-the-art source routed schemes, Elmo and Bert, discussed in Section III. We focus on four metrics/aspects (defined in Section IV-C): header sizes, traffic overhead, switch memory cost, and source packet replications. We also compare Ernie to other switch-based multicast schemes, like iRP and Miniforest, in terms of traffic overhead and switch memory cost.

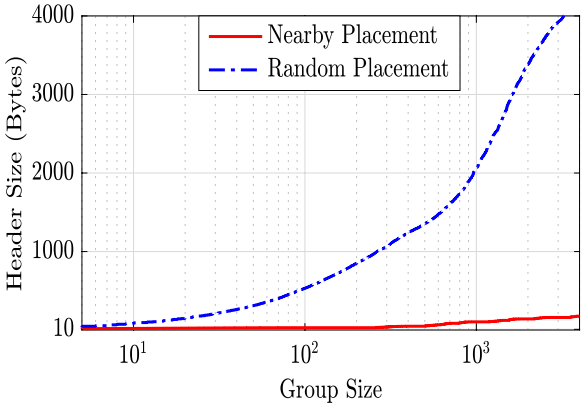


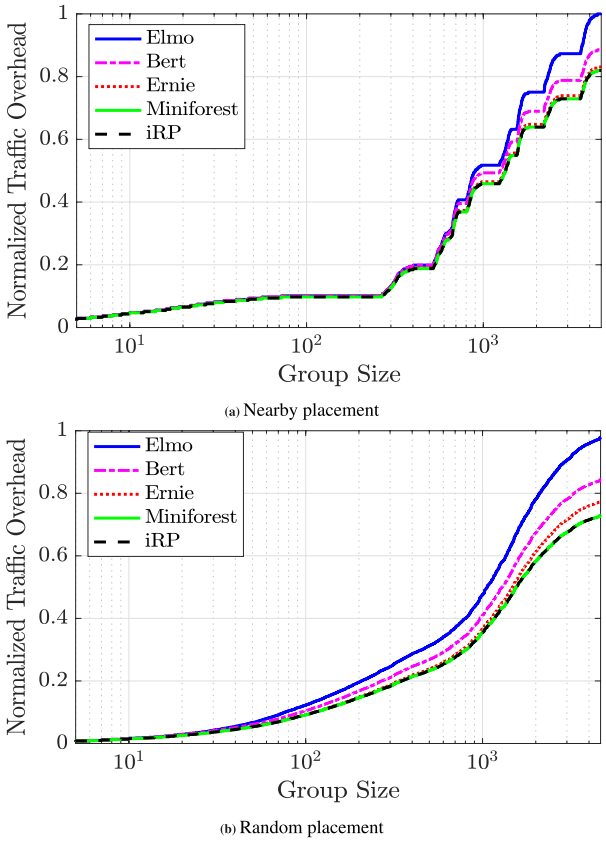
FIGURE 6. Ernie's upstream header size.

1) HEADER SIZES

We first evaluate the packet header overhead in the downstream path. Figs. 5a and 5b show the header size as a function of group size under the Nearby and Random placement strategies. Note that the header size of Elmo increases as the size of the group increases in each placement strategy. On the other hand, Bert and Ernie adapt to the group size and do not blow up as the header size increases. For the Nearby placement, Ernie requires a header size that is more than 10× and 3× smaller than that needed under Elmo for large group sizes (e.g. 1000 members or more) and small group size (e.g. 100 members or less), respectively. Compared to Bert, Ernie requires a header sizes that is at least 3× smaller regardless of the group size. For the Random placement, compared to Elmo, Ernie requires about 40× smaller header sizes for group sizes between 50 and 150 members and about 6× for large group sizes (e.g. 1000 or more). When compared to Bert, Ernie requires about 10× and 1.6× smaller header size for small and large group sizes, respectively.

As explained in Section IV-A, Ernie significantly achieves this improvement by obviating unneeded p-rules received on each destination pod as well as reducing the number of bits used for switch IDs. On the other hand, compared to Elmo, Bert reduces the forwarding header by clustering the multicast groups, at the cost of increasing the number of packet replications at the host (see below). Note that Elmo has bounded the packet header sizes to 512 bytes, and opted for storing the extra forwarding information in switch memories. So, we apply this restriction to Elmo Header. We evaluate switch memory usage in Section V-A3.

Fig. 6 shows that the header of Ernie traverses through upstream links (from source leaf to core switches). For the Nearby placement, the header size marginally increases as the size of the group increases. For example, it increases from 20 bytes to 150 bytes when the group size is increased from 10 to 5000 members. For the Random placement strategy, the header size dramatically increases as the size of the group increases. Because group members are dispersed across pods and leaf switches, forwarding information is too large to encode, especially for large group sizes.



(a) Nearby placement

(b) Random placement

FIGURE 7. Traffic overhead as function of group size for large packet payload of 1500 bytes. Group size varying from 5-5000 members.

Nevertheless, as will be shown in this section, the overall traffic, switch memory, and end-host CPU overheads achieved by Ernie are minimal.

2) TRAFFIC OVERHEAD

In this experiment, we evaluate the overall traffic overhead incurred in the upstream (from leaf switches to core switches) and downstream (from core switches to leaf switches) paths. We compare Ernie with the source-routed (Elmo and Bert) and with the switch-based schemes (Miniforest and iRP). In switch-based schemes, multicast forwarding states/information are stored in switches, so the header overhead is zero. Figs. 7a and 7b show the network traffic overhead of the Nearby and Random placement strategies for packet payload of 1500 bytes. The results are normalized to the values achieved by Elmo. Observe that Ernie performs better than Elmo and Bert for both placement strategies, especially at large group sizes. For small group sizes (i.e. less than 100 members), the performance achieved under all three schemes is close to that achieved under the switch-based schemes (zero header overhead). This is because when the group size is small, both the header size and the number of downstream replications are small. For large group sizes, Ernie still performs effectively compared to the switch-based schemes. However, Elmo contributes 20% and 27% more traffic compared to them for the Nearby and Random

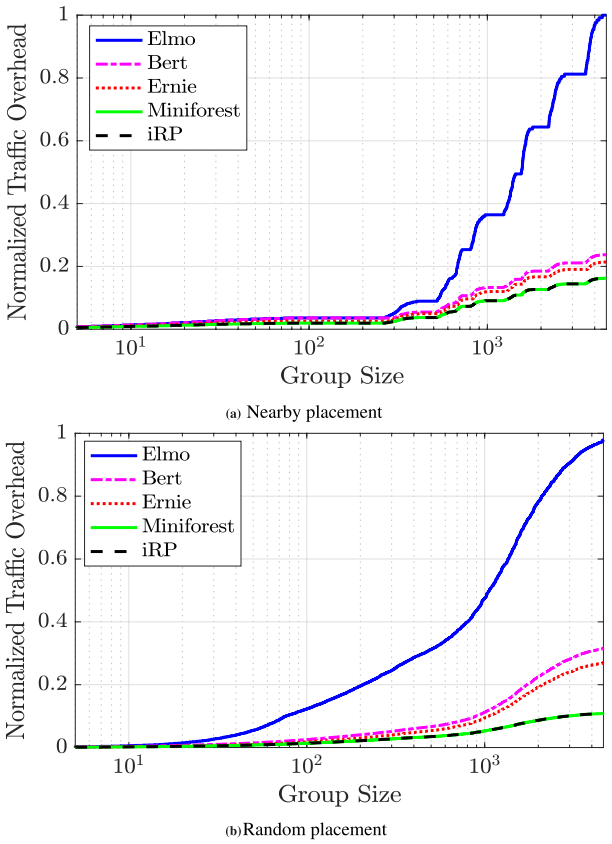


FIGURE 8. Traffic overhead as function of group size for small packet payload sizes (64 bytes). Group size varying from 5-5000 members.

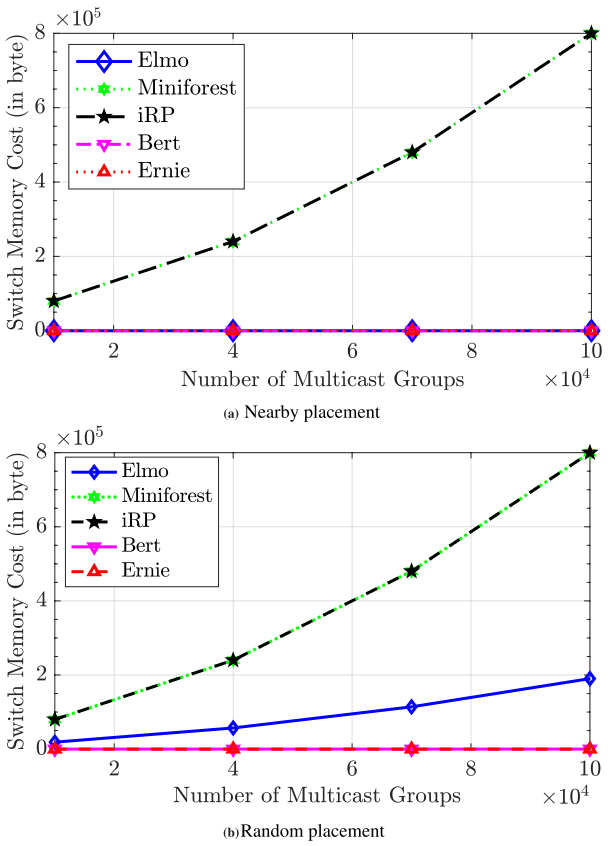


FIGURE 9. Switch memory costs.

placement strategies, respectively. Bert contributes about 9% more traffic compared to the switch-based schemes for both placement strategies.

We also assess the network traffic overheads of Nearby and Random placement strategies in Figs. 8a and 8b for small packet payload sizes (i.e. 64 bytes). Compared to switch-based schemes, source-routed schemes contribute more traffic when packet payload is small for large group sizes. This is because the header size for large group sizes is large when compared to the payload size. Elmo traffic overhead rises quickly when the group size is increased. Again, this is because in Elmo, the header size is too large compared to the payload size for large group sizes. Bert does better than Elmo here by dividing the members of the multicast group into a set of clusters. Again, this is at the price of increasing the number of packet replications at the host. Compared to Elmo and Bert, Ernie achieves significant reductions when smaller payload sizes are used. For example, Ernie achieves 4× lower traffic overhead compared to Elmo and 1.2× better compared to Bert for large group sizes (i.e. 5000 members) in the two placement strategies.

In summary, Ernie yields less traffic overhead when compared to the source-routed schemes, Elmo and Bert, while performing competitively when compared to the switch-based schemes (zero header overhead).

3) SWITCH MEMORY COST

We next study the switch memory cost for all schemes in both placement strategies. First, in Fig. 9 we study the effects of the different number of multicast groups (e.g., 10K-100K multicast groups) on the switch memory for Nearby placement. Regardless of the number of groups and placement strategies, Ernie and Bert show drastic improvements in switch memory utilization, as there is no need to store any forwarding information and hence switch memory usage is zero for both schemes. On the other hand, in Miniforest and iRP, switch memory cost is dramatically increased when the number of multicast groups is increased, regardless of the placement strategy. For example, switch memory cost increases between 100K- 800K bytes when the number of groups varies from 10K – 100K. Note that real switching hardware supports only a limited number of multicast group table sizes, typically thousands to a few tens of thousands of entries nowadays [7]. This shows the scalability obstacle of switch-based multicast schemes in today's public clouds. In Elmo, switch memory cost depends on the placement strategies and the number of multicast groups. Elmo stores some forwarding information in switch's multicast table when the forwarding headers reach their maximum size (i.e., 512 bytes). In the Nearby placement, the header size of Elmo does not exceed the threshold for all numbers of the multicast group, yielding a zero

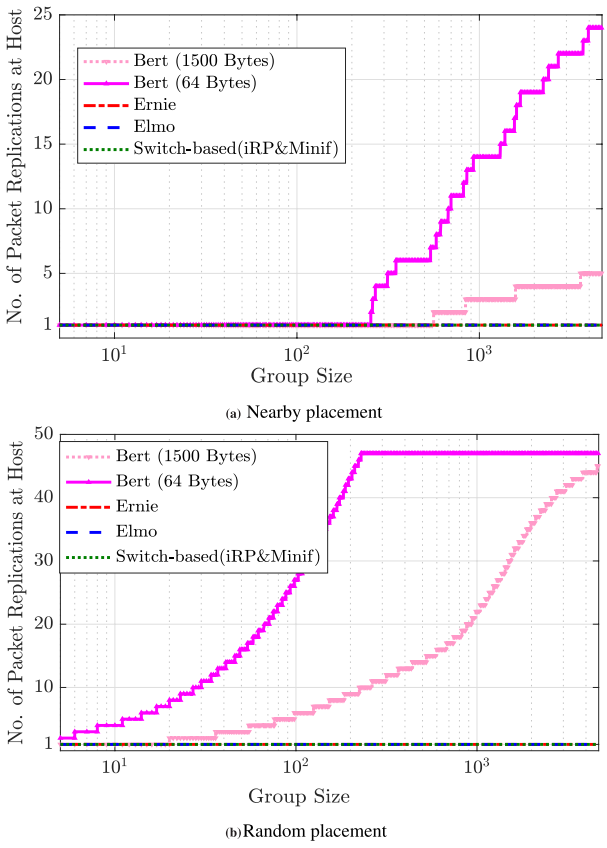


FIGURE 10. Number of packet replications at the host as a function of group size.

switch memory usage. However, for the Random placement, Elmo's switch memory cost increases as the group sizes increase. For example, when the number of groups varies from 10K-100K, the switch memory cost increases between 10K to 180K bytes. However, it is much less when compared to switch-based multicast schemes.

4) SOURCE PACKET REPLICATIONS

In this section, we assess the CPU overhead of the studied schemes, and do so by capturing the number of packet replications the source needs to make, which captures various aspects of CPU overhead like processing delay, CPU power consumption, storage cost, etc. Here, the higher the number of packet replications is, the higher the CPU overhead is, and vice versa. In Fig 10, regardless of the multicast group size or placement strategy, Ernie, Elmo, Miniforest, and iRP schemes maintain minimal overheads since only one packet replication takes place at the source. On the other hand, in Bert, packets are replicated per cluster where the number of packet replications depends on the group sizes, placement strategies, and packet payload size. Bert clusters/divides multicast groups to reduce the packet header sizes but at the cost of increasing the CPU overhead. For example, in Fig. 10a, for the Nearby placement with a large packet payload, the number of replications is 1 for small group sizes, and it slightly increases and reaches 5 when the multicast group size is large (e.g 5000 members). However, in Fig. 10b, for the Random placement with a small packet payload, the number of replications dramatically increases when the group size is increased.

B. LOAD BALANCING ANALYSIS

We now turn our attention to the study of Ernie's load balancing ability, and in doing so, we compare it with the following existing load balancing schemes:

- iRP [13]: multicast groups are assigned to least-congested core switches. Controller maintains bandwidth information for all core links and chooses the least-congested one.
- Miniforest (minif) [18]: Multicast groups are randomly divided into several disjoint sets, and each set is routed through a unique core switch.

In addition, we compared Ernie to the following three baseline approaches:

- Least-Congested Link (LCL): distributes packets/flows through an optimum link by considering the current link load.
- Round-Robin (RR): Multicast groups are assigned to core switches in a round-robin manner.
- Random (Rand): multicast groups are randomly assigned to a core switches.

For this experiment, the traffic load on each link is randomly chosen from 100 – 1000 bytes for all DC links. We consider 10K multicast groups that generate traffic with multicast group sources are each sending one packet of size 1500 bytes. We calculate the amount of traffic for all links in each layer. For ease of illustration, we visualize the traffic load for all links in each layer using boxplot. In boxplot, the central red mark is the median; whiskers represent the min and max value; boxes show the 25*th*, 50*th* (red line), and 75*th* percentiles. We also calculate the standard deviation (SD) to show the evenness of load distribution across the links in each layer; here the lower the standard deviations are, the closer the loads of links are to one another.

In Figs. 11 and 12, we observe that downstream traffic (core-to-spine and spine-leaf) in both placement strategies is much heavier than upstream traffic (leaf-to-spine and spine-to-core), and this is due to packet replications. Furthermore, downstream traffic load of the Random placement strategy is much heavier than that of the Nearby placement. Simply the reason is that for the Random placement strategy, where group members are dispersed across leaf switches, the number of packet replications is higher than that of the Nearby placement. For example, in the Nearby placement, traffic of core-to-spine (Fig. 11c) and spine-to-leaf (Fig. 11d) is about 3× and 50× more than spine-to-core (Fig. 11b) and leaf-to-spine (Fig. 11a), respectively. For the Random placement, traffic of core-to-spine (Fig. 12c) and spine-to-leaf (Fig. 12d) is about 28× and 240× more than spine-to-core (Fig. 12b) and leaf-to-spine (Fig. 12a), respectively.

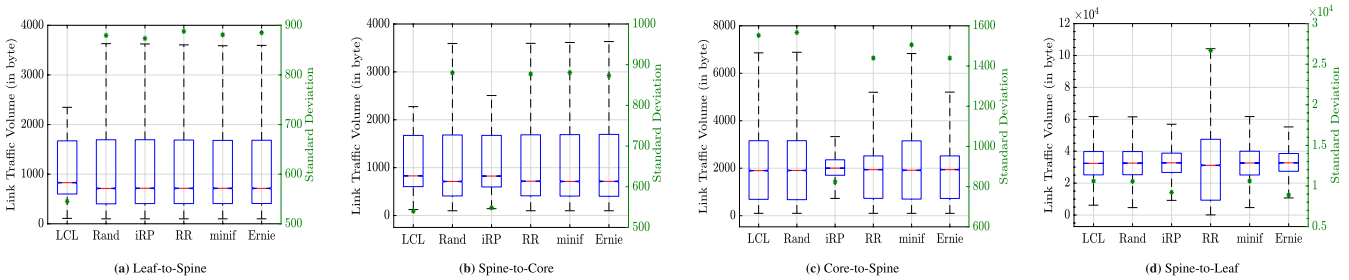


FIGURE 11. Load Balancing simulation for Nearby placement. In the boxplot, the central red mark is the median; whiskers represent the min and max value; boxes show the 25*th*, 50*th* (red line), and 75*th* percentiles. The right y-axis and the green star is for the Standard Deviation (SD).

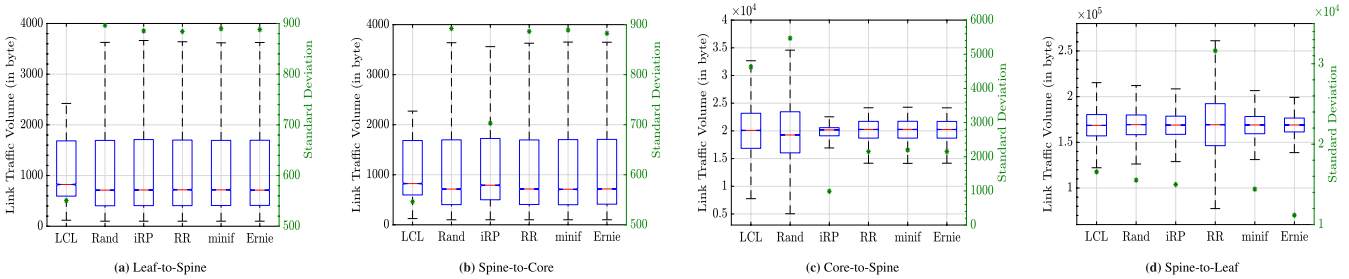


FIGURE 12. Load Balancing simulation for Random placement. In the boxplot, the central red mark is the median; whiskers represent the min and max value; boxes show the 25*th*, 50*th* (red line), and 75*th* percentiles. The right y-axis and the green star (*) is for the Standard Deviation (SD).

In spine-to-leaf layer (Figs. 12d and 11d), Ernie outperforms other schemes and returns the lowest SD (standard deviation) of link utilizations in both placement strategies. For example, SD (right Y-axis) achieved by Ernie in the Random placement strategy (Fig.12d) is about 65% less than RR, 32% less than Random, LCL and iRP, and 25% less than Miniforest. This is as expected because as we have shown in Section IV-B2, Ernie takes all downstream traffic (core-to-spine and spine-leaf) into account when assigning multicast groups to core switches to avoid link congestion. In the Nearby placement strategy (Fig. 11d), Ernie still achieves the best load balancing with an SD of about 67% lesser than RR, 17% lesser than Randam, LCL, and Miniforest, and 4% lesser than iRP. Interestingly, the performance of the RR approach is the worst among all schemes at spine-to-leaf layer (Figs. 11d and 12d) in both placement strategies. The reason is as shown in Section IV-B1 Fig. 3b; due to topological property of fat-tree structures, when multicast groups are assigned to core switches of the same core group in round-robin manner, the same (spine-to-leaf) links repeatedly exploited by multiple consecutive multicast groups, while other links are not used at all.

In the core-to-spine layer (Figs. 12c and 11c), Ernie achieves better load balancing when compared to LCL and Randam schemes with SD being lesser than 60% and 10% of these schemes in the Random and Nearby placement strategies, respectively. Furthermore, Ernie performs similarly to RR and Miniforest in both placement strategies. This is because these three schemes evenly distribute multicast groups among all core switches. On the other hand, iRP outperforms all schemes in the two placement strategies due to link congestion visibility of iRP in this layer. However, the cost of this achievement is quite high. For example, in this experiment, iRP needs to monitor and compare about 10^8 links to choose the best core.

For load balancing of the upstream traffic (Figs. 11a, 11b, 12a and 12b), Ernie, Rand, Miniforest, and RR perform closely to each other in both placement strategies. In these schemes, when a core switch is selected, a multicast packet traverses a fixed, predetermined path between source and destination(s). In other words, when a core switch is selected, there is only one path from source leaf switch to destination leaf switch. Hence, a congested upstream path may be picked to be the routing path. LCL performs well in the upstream path, and poorly on the downstream path in terms of load balancing. The reason is because when packet crossing layers in the upstream direction, multiple paths are possible and the least congested one is picked. However, once a core switch has been reached and the packet is sent down to destinations, only a single path is available (algorithm cannot work). Ernie is within 35% of LCL (ideal load balancing for the upstream traffic) in both placement strategies in each upstream layer. iRP achieves good load balancing at spine-to-core layer (Figs. 11b and 12b) traffic, and again this is due to the monitoring of enter-pod traffic.

C. DISCUSSION: PROS AND CONS

We discuss here some challenging scenarios that can limit the performance of Ernie. First, we have seen in Section V-A1 (Fig. 6) that in Random placement strategy, Ernie requires

larger header sizes than those required by Elmo and Bert at the upstream layers. This is because the multicast group members are dispersed across pods and leaf switches, thereby needing more encoded information. Note that this scenario is not only challenging for Ernie but also for both Elmo and Bert. In Elmo, in the downstream paths (highly congested paths), header sizes are much larger than those of Ernie (Fig. 5b), resulting in higher traffic overhead (4× greater than Ernie (Fig. 5b)). Moreover, for this scenario, Elmo needs to store some forwarding information in the switch's multicast table while Ernie does not use switch memory at all (Fig. 9b). In the case of Bert, Random placement requires more clusters (more packet replication at the source) which in turn increases end-host CPU overheads. For example, packet replications at the source ranges between 1 and 48 where group sizes range between 10 and 5000 members (Fig. 10).

Second, Ernie performs worse than some of the other schemes (Figs. 11and 12) in terms of load balancing at the upstream layers. The reason is, as discussed in Section V-B, that the multicast routing path in Ernie is predetermined, and unlike LCL and iRP, congested path cannot be avoided. However, at the highly congested links, Ernie achieves better load balancing when compared to all other schemes.

VI. CONCLUSION AND FUTURE DIRECTIONS

Multicast is a crucial communication primitive in today's cloud DC networks. Multicast benefits group communications in saving network bandwidth and increasing application throughput. The use of IP multicast has been traditionally curtailed due to scalability and load balancing limitations. Despite much progress in recent years towards addressing these challenges, an efficient scalable and load-balanced multicast method for cloud DCs has remained elusive. To this end, we present Ernie, a scalable load-balanced multicast source routing scheme suitable for large-scale cloud DCs. When compared to existing multicast routing schemes for DCs, Ernie is developed with two design goals in mind: 1) scalability; it scales well with the number and size of multicast groups in that it incurs minimal network overhead in terms of header size, network traffic, and switch memory, and 2) load balancing; it achieves good balance of traffic loads at highly congested links. With new and emerging developments in both programmable and virtualized networks, we believe that Ernie is qualified to be deployable in today's production DCs, as it neither requires new network hardware nor does it require changes to the applications running on the end host.

Current advancements in cloud DC networks unlock a variety of applications and open challenges. First, contemporary literature lacks real-world multicast studies related to cloud DCs. Most works focus on flow characteristics such as flow sizes, arrival rates, and distributions [6]. Thus, we believe there is a need for studies that analyze multicast behavior in detail such as the number of multicast groups, multicast group sizes, and group member distributions in real-world DCs. These studies would be useful in evaluating newly proposed multicast approaches for DCs. Second, in addition to scalability and load balancing, a reliable multicast protocol should easily with the lowest-cost handle common multicast group dynamics when members leave or join an existing group requiring forwarding information in switches or packet headers to be updated. This behavior, known as churn, exhausts network hardware resources and increases control plane overhead. Thus, stability and adaptivity against churn are of crucial importance for reliable multicast protocols in cloud DCs. Finally, Ernie with new features that consider multicast membership dynamics and respond immediately to the failure while achieving the lowest-cost solution are left for future work.

REFERENCES

[1] *Amazon AWS Kernel Description*. Accessed: Sep. 10, 2021. [Online]. Available: https://aws.amazon.com/

[2] *Microsoft Azure*. Accessed: Feb. 10, 2020. [Online]. Available: https://azure.microsoft.com/

[3] *Google Compute Engine*. Accessed: Sep. 10, 2021. [Online]. Available: https://cloud.google.com/compute/

[4] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *Proc. IEEE 26th Symp. Mass Storage Syst. Technol. (MSST)*, May 2010, pp. 1–10.

[5] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[6] J. Alqahtani, S. Alanazi, and B. Hamdaoui, "Traffic behavior in cloud data centers: A survey," in *Proc. Int. Wireless Commun. Mobile Comput. (IWCMC)*, Jun. 2020, pp. 2106–2111.

[7] *Cisco Nexus 5000 Series NX-OS Multicast Routing Command Reference*. Accessed: Sep. 29, 2021. [Online]. Available: https://www.cisco.com/c/en/us/td/docs/switches/datacenter/nexus5000/sw/command/reference/multicast/n5k-mcast-cr/n5k-igmpsnp_cmds_h.html

[8] B. Fenner, M. Handley, H. Holbrook, I. Kouvelas, R. Parekh, Z. Zhang, and L. Zheng, *Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification (Revised)*, document RFC 7761, 2016, pp. 1–137.

[9] H. Holbrook, B. Cain, and B. Haberman, *Using Internet Group Management Protocol Version 3 (Igmpv3) and Multicast Listener Discovery Protocol Version 2 (Mldv2) for Source-Specific Multicast*, document RFC 4604, Internet Engineering Task Force, 2006.

[10] *Overlay Multicast in Amazon Virtual Private Cloud*. Accessed: Oct. 11, 2021. [Online]. Available: https://aws.amazon.com/articles/overlay-multicast-in-amazon-virtual-private-cloud/

[11] X. Li and M. J. Freedman, "Scaling IP multicast on datacenter topologies," in *Proc. 9th ACM Conf. Emerg. Netw. Exp. Technol.*, 2013, pp. 61–72.

[12] W. Cui and C. Qian, "Scalable and load-balanced data center multicast," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2015, pp. 1–6.

[13] A. Latif, P. Kathail, S. Vishwarupe, S. Dhesikan, A. Khreishah, and Y. Jararweh, "Multicast optimization for CLOS fabric in media data centers," *IEEE Trans. Netw. Service Manage.*, vol. 16, no. 4, pp. 1855–1868, Dec. 2019.

[14] Z. Guo, J. Duan, and Y. Yang, "On-line multicast scheduling with bounded congestion in fat-tree data center networks," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 1, pp. 102–115, Jan. 2014.

[15] M. Shahbaz, L. Suresh, J. Rexford, N. Feamster, O. Rottenstreich, and M. Hira, "Elmo: Source routed multicast for public clouds," in *Proc. ACM Special Interest Group Data Commun.*, Aug. 2019, pp. 458–471.

[16] J. Alqahtani, H. H. Sinky, and B. Hamdaoui, "Clustered multicast source routing for large-scale cloud data centers," *IEEE Access*, vol. 9, pp. 12693–12705, 2021.

[17] C. Hopps, *Analysis of an Equal-Cost Multi-Path Algorithm*, document RFC 2992, 2000. [Online]. Available: https://www.rfc-editor.org/rfc/rfc2992.txt

[18] F. Fan, B. Hu, K. L. Yeung, and M. Zhao, "MiniForest: Distributed and dynamic multicasting in datacenter networks," *IEEE Trans. Netw. Service Manage.*, vol. 16, no. 3, pp. 1268–1281, Sep. 2019.

[19] A. Liberato, M. Martinello, R. L. Gomes, A. F. Beldachi, E. Salas, R. Villaca, M. R. N. Ribeiro, K. Kondepu, G. Kanellos, R. Nejabati, A. Gorodnik, and D. Simeonidou, "RDNA: Residue-defined networking architecture enabling ultra-reliable low-latency datacenters," *IEEE Trans. Netw. Service Manage.*, vol. 15, no. 4, pp. 1473–1487, Dec. 2018.

[20] W. K. Jia, "A scalable multicast source routing architecture for data center networks," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 1, pp. 116–123, Jan. 2014.

[21] P. Jokela, A. Zahemszky, C. E. Rothenberg, S. Arianfar, and P. Nikander, "LIPSIN: Line speed publish/subscribe inter-networking," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 195–206, Aug. 2009.

[22] D. Li, Y. Li, J. Wu, S. Su, and J. Yu, "ESM: Efficient and scalable data center multicast routing," *IEEE/ACM Trans. Netw.*, vol. 20, no. 3, pp. 944–955, Jun. 2012.

[23] J. Baek and G. Kaddoum, "Heterogeneous task offloading and resource allocations via deep recurrent reinforcement learning in partial observable multifog networks," *IEEE Internet Things J.*, vol. 8, no. 2, pp. 1041–1056, Jan. 2021.

[24] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, "Energy-efficient resource allocation and provisioning framework for cloud data centers," *IEEE Trans. Netw. Service Manage.*, vol. 12, no. 3, pp. 377–391, Sep. 2015.

[25] D. Saxena, I. Gupta, J. Kumar, A. K. Singh, and X. Wen, "A secure and multiobjective virtual machine placement framework for cloud data center," *IEEE Syst. J.*, early access, Jul. 20, 2021, doi: 10.1109/JSYST.2021.3092521.

[26] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, "Online assignment and placement of cloud task requests with heterogeneous requirements," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2015, pp. 1–6.

[27] S. Alanazi and B. Hamdaoui, "CAFT: Congestion-aware fault-tolerant load balancing for three-tier clos data centers," in *Proc. Int. Wireless Commun. Mobile Comput. (IWCMC)*, Jun. 2020, pp. 1746–1751.

[28] N. Katta, M. Hira, A. Ghag, C. Kim, I. Keslassy, and J. Rexford, "CLOVE: How I learned to stop worrying about the core and love the edge," in *Proc. 15th ACM Workshop Hot Topics Netw.*, Nov. 2016, pp. 155–161.

[29] P. Wang, G. Trimponias, H. Xu, H. Liu, and Y. Geng, "Luopan: Sampling-based load balancing in data center networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 1, pp. 133–145, Jan. 2019.

[30] M. Dabbagh, B. Hamdaoui, A. Rayes, and M. Guizani, "Shaving data center power demand peaks through energy storage and workload shifting control," *IEEE Trans. Cloud Comput.*, vol. 7, no. 4, pp. 1095–1108, Oct. 2019.

[31] J. Liu, X. Wang, S. Shen, Z. Fang, S. Yu, G. Yue, and M. Li, "Intelligent jamming defense using DNN Stackelberg game in sensor edge cloud," *IEEE Internet Things J.*, early access, Aug. 9, 2021, doi: 10.1109/JIOT.2021.3103196.

[32] J. Liu, X. Wang, S. Shen, G. Yue, S. Yu, and M. Li, "A Bayesian Q-learning game for dependable task offloading against DDoS attacks in sensor edge cloud," *IEEE Internet Things J.*, vol. 8, no. 9, pp. 7546–7561, May 2021.

[33] B. Wu, X. Chen, Z. Wu, Z. Zhao, Z. Mei, and C. Zhang, "Privacy-guarding optimal route finding with support for semantic search on encrypted graph in cloud computing scenario," *Wireless Commun. Mobile Comput.*, vol. 2021, pp. 1–12, Mar. 2021.

[34] Z. Wu, S. Shen, H. Zhou, H. Li, C. Lu, and D. Zou, "An effective approach for the protection of user commodity viewing privacy in e-commerce website," *Knowl.-Based Syst.*, vol. 220, May 2021, Art. no. 106952.

[35] Z. Wu, G. Li, S. Shen, X. Lian, E. Chen, and G. Xu, "Constructing dummy query sequences to protect location privacy and query privacy in location-based services," *World Wide Web*, vol. 24, no. 1, pp. 25–49, Jan. 2021.

[36] G. Portaluri, D. Adami, A. Gabbrielli, S. Giordano, and M. Pagano, "Power consumption-aware virtual machine placement in cloud data center," *IEEE Trans. Green Commun. Netw.*, vol. 1, no. 4, pp. 541–550, Dec. 2017.

[37] S. Alanazi, M. Dabbagh, B. Hamdaoui, M. Guizani, and N. Zorba, "Reducing data center energy consumption through peak shaving and locked-in energy avoidance," *IEEE Trans. Green Commun. Netw.*, vol. 1, no. 4, pp. 551–562, Dec. 2017.

[38] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, "An energy-efficient VM prediction and migration framework for overcommitted clouds," *IEEE Trans. Cloud Comput.*, vol. 6, no. 4, pp. 955–966, Oct. 2018.

[39] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, "Efficient datacenter resource utilization through cloud resource overcommitment," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Apr. 2015, pp. 330–335.

[40] S. Naribole and E. Knightly, "Scalable multicast in highly-directional 60-GHz WLANs," *IEEE/ACM Trans. Netw.*, vol. 25, no. 5, pp. 2844–2857, Oct. 2017.

[41] E. Tsimbalo, A. Tassi, and R. J. Piechocki, "Reliability of multicast under random linear network coding," *IEEE Trans. Commun.*, vol. 66, no. 6, pp. 2547–2559, Feb. 2018.

[42] Z. Tang, H. Wang, Q. Hu, and C. Li, "Performance analysis of multi-user multi-round linear network coded cooperation," *IEEE Commun. Lett.*, vol. 18, no. 10, pp. 1767–1770, Oct. 2014.

[43] L. Hai, H. Wang, J. Wang, and Z. Tang, "HCOR: A high-throughput coding-aware opportunistic routing for inter-flow network coding in wireless mesh networks," *EURASIP J. Wireless Commun. Netw.*, vol. 2014, no. 1, pp. 1–13, Dec. 2014.

[44] P. Judge and M. Ammar, "Security issues and solutions in multicast content distribution: A survey," *IEEE Netw.*, vol. 17, no. 1, pp. 30–36, Jan. 2003.

[45] J. Widmer and M. Handley, *TCP-Friendly Multicast Congestion Control (TFMCC): Protocol Specification*, document Rec. 4654, 2006.

[46] S. Islam, N. Muslim, and J. W. Atwood, "A survey on multicasting in software-defined networking," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 1, pp. 355–387, 1st Quart., 2018.

[47] S. Ratnasamy, A. Ermolinskiy, and S. Shenker, "Revisiting IP multicast," in *Proc. Conf. Appl., Technol., Archit., Protocols Comput. Commun.*, 2006, pp. 15–26.

[48] X. Gao, T. Chen, Z. Chen, and G. Chen, "NEMO: Novel and efficient multicast routing schemes for hybrid data center networks," *Comput. Netw.*, vol. 138, pp. 149–163, Jun. 2018.

[49] D. Li, M. Xu, Y. Liu, X. Xie, Y. Cui, J. Wang, and G. Chen, "Reliable multicast in data center networks," *IEEE Trans. Comput.*, vol. 63, no. 8, pp. 2011–2024, Aug. 2014.

[50] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. NSDI*, 2010, vol. 10, no. 8, pp. 89–92.

[51] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav, and G. Varghese, "CONGA: Distributed congestion-aware load balancing for datacenters," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 503–514, 2014.

[52] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal, "Fastpass: A centralized zero-queue datacenter network," *Comput. Commun. Rev.*, vol. 44, no. 4, pp. 307–318, 2014.

[53] K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, and A. Akella, "Presto: Edge-based load balancing for fast datacenter networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 465–478, 2015.

[54] V. Nithin, A. Rathod, V. Badarla, T. Humernbrum, and S. Gorlatch, "Efficient load balancing for multicast traffic in data center networks using SDN," in *Proc. 10th Int. Conf. Commun. Syst. Netw. (COMSNETS)*, Jan. 2018, pp. 113–120.

[55] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM Computer Commun. Rev.*, vol. 38, no. 4, pp. 63–74, Aug. 2008.

[56] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Vl2: A scalable and flexible data center network," in *Proc. ACM SIGCOMM Conf. Data Commun.*, 2009, pp. 51–62.

[57] V. Liu, D. Halperin, A. Krishnamurthy, and T. Anderson, "F10: A fault-tolerant engineered network," in *Proc. 10th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2013, pp. 399–412.

[58] J. Alqahtani and B. Hamdaoui, "Rethinking fat-tree topology design for cloud data centers," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2018, pp. 1–6.

[59] *Netflix on AWS*. Accessed: Nov. 10, 2021. [Online]. Available: https://aws.amazon.com/solutions/case-studies/netflix/

[60] *What is a Hypervisor?* Accessed: Nov. 10, 2021. [Online]. Available: https://www.vmware.com/topics/glossary/content/hypervisor

[61] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, and P. Shelar, "The design and implementation of open vSwitch," in *Proc. 12th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2015, pp. 117–130.

[62] M. Mahalingam, D. G. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright, *Virtual Extensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks Over Layer 3 Networks*, document Rec. RFC 7348, 2014, pp. 1–22.
[63] *Intel Programmable Ethernet Switch Products*. Accessed: Apr. 20, 2021. [Online]. Available: https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch.html
[64] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 87–95, Jul. 2014.

JARALLAH ALQAHTANI received the B.S. degree from King Khalid University, Abha, Saudi Arabia, and the M.S. degree from the Illinois Institute of Technology, Chicago, IL, USA. He is currently pursuing the Ph.D. degree in computer science with Oregon State University, Corvallis, OR, USA. He received academic scholarship from Najran University to support this Ph.D. research work, in 2012. His research interests include data center networking and cloud computing.

BECHIR HAMDAOUI (Senior Member, IEEE) received the M.S. degrees in ECE and in CS and the Ph.D. degree in ECE from the University of Wisconsin–Madison, in 2002, 2004, and 2005, respectively. He is currently a Professor with the School of Electrical Engineering and Computer Science, Oregon State University. His research interests include the general areas of networked systems, network security, and wireless communication networks, with a current focus on machine learning for wireless systems, edge cloud computing, autonomous systems, next-generation wireless networks, spectrum management, and datacenter networking. He is a Senior Member of IEEE Computer Society, IEEE Communications Society, and IEEE Vehicular Technology Society. He won several awards, including the ISSIP 2020 Distinguished Recognition Award, the ICC 2017 Best Paper Award, the IWCMC 2017 Best Paper Award, the 2016 EECS Outstanding Research Award, and the 2009 NSF CAREER Award. He also chaired/co-chaired many IEEE conference programs, including the 2021 GLOBECOM Testbeds4Wireless Workshop, the 2017 INFOCOM Demo/Posters Program, the 2016 IEEE GLOBECOM Mobile and Wireless Networks Symposium, and many others. He served as a Distinguished Lecturer for the IEEE Communication Society, in 2016 and 2017, and currently serves as the Chair for the IEEE Communications Society's Wireless Technical Committee (WTC). He serves/served as an Associate Editor for several journals, including IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, IEEE NETWORK, and IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY.

RAMI LANGAR is currently a Professor in computer science with University Gustave Eiffel, France, and the Head of the SNR (Software, Network, Real-time) Research Group, Gaspard-Monge Computer Science Laboratory. His research interests include resource management in future wireless systems, cloud-RAN, network slicing in 5G/5G+/6G networks, software-defined wireless networks, and mobile cloud offloading. He has been recently a coordinator or the technical leader for several projects in computer networking, including: 5G-INSIGHT (ANR, 2021–2024), Smart-Grid-5G (EDF-Labs, 2019–2022), FlashRAN (CNRS/Paris-Region, 2019–2021), SCORPION (FUI, 2017–2020); PODIUM (FUI, 2016–2019); and ELASTIC Networks (FUI, 2015–2018). He has been Area Editor of IEEE SYSTEMS journal, since November 2019, IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS (JSAC)—SERIES ON NETWORK SOFTWARIZATION & ENABLERS, since September 2018, and *Journal of Network and Service Management* (JONS), since January 2018. He was the Chair of the IEEE ComSoc Technical Committee on Information Infrastructure and Networking (TCIIN), from January 2018 to December 2019. He also chaired/co-chaired many IEEE/IFIP conference programs/symposia, including the 2020 IEEE GLOBECOM Next Generation Networking and Internet (NGNI) Symposium, the 2019 and 2016 IEEE ICC NGNI Symposium, the 2020 IFIP Networking Poster, and many others.