Received December 19, 2020, accepted January 9, 2021, date of publication January 14, 2021, date of current version January 22, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3051874

Clustered Multicast Source Routing for Large-Scale Cloud Data Centers

JARALLAH ALQAHTANI[1], HASSAN H. SINKY[2], AND BECHIR HAMDAOUI[1], (Senior Member, IEEE)
[1]School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR 97331, USA
[2]College of Computer and Information Systems, Umm Al-Qura University, Makkah 24381, Saudi Arabia

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

This work was supported by the Deputyship for Research and Innovation, Ministry of Education, Saudi Arabia, under Grant UQU-IF-P2-20-001.

ABSTRACT The multi-tenancy concept in cloud data center (DC) networks paves the way towards advancements and innovation in the underlying infrastructure such as network virtualization. Multicast routing is essential in leveraging multi-tenancy to its full potential. However, traditional IP multicast routing is not suitable for DC networks due to the need to support a massive amount of multicast groups and hosts. State-of-the-art DC multicast routing approaches aim to overcome these scalability issues by, for instance, taking advantage of the symmetry of DC topologies and the programmability of DC switches to compactly encode multicast group information inside packets, thereby reducing the overhead resulting from the need to store the states of flows at the network switches. Although these approaches scale well with the number of multicast groups, they do not perform well with group sizes and, as a result, yield substantial traffic control overhead and network congestion. In this article, we present Bert, a scalable source-initiated DC multicast routing approach that scales well with both the number and size of multicast groups through the clustering of multicast group members where each cluster employs its own forwarding rules. Compared to the state-of-the-art approach, Bert yields much less traffic control overhead by significantly reducing packet header sizes and eliminating switch memory usage across the switches.

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

I. INTRODUCTION

Modern data center infrastructures have shifted from traditional on-premise physical servers to virtual networks where data and services exist and are connected across pools of data centers, both on-premises and in the cloud. Cloud computing is an emerging service model where massive data centers are built by cloud providers to offer services to tenants. Today's cloud data centers (DCs) host hundreds of thousands of tenants [1], [2], with each tenant possibly running hundreds of workloads supported through thousands of virtual machines (VMs) running on different servers [3]–[5]. These workloads often involve one-to-many communications among the different servers running VMs supporting the same workload/application [6]–[8]. Therefore, to enable efficient communication and data transfer among different servers, multicast routing protocol designs must be revisited to suit today's cloud data center network topologies. In this article, we study a critical requirement of DC topologies, i.e., multicast scalability. Traditional IP multicast routing is primarily designed for arbitrary network topologies and Internet traffic, with a focus on reducing CPU and network bandwidth overheads, and hence is not suitable for DCs due to the need for supporting large numbers of groups in commodity switches with limited memory capability. In other words, DC switches will have to maintain per-group routing rules for all multicast addresses, because they cannot be aggregated on per prefix basis.

That said, there have been few research efforts devoted to overcome this scalability issue [9]–[15]. For instance, Elmo [12], a recently proposed source-initiated multicast routing approach for DCs, overcomes the scalability issue and is shown to support millions of multicast groups with reasonable overhead in terms of switch state and network

The associate editor coordinating the review of this manuscript and approving it for publication was Thanh Ngoc Dinh.

traffic. Elmo does so by taking advantage of programmable switches [16] and the symmetry of DC topologies to compactly encode multicast group information inside packets, thereby reducing the overhead resulting from the need to store the states of flows at the network switches. However, although Elmo scales well with the number of multicast groups, it does not do so with multicast group sizes. When considering large multicast group sizes, Elmo header can carry on several hundreds of bytes extra, which increases traffic overhead in the network. In addition, the number of extra transmissions Elmo incurs due to compacting of packet rules increases significantly with the size of multicast group, yielding higher traffic congestion in the DC's downlinks. To overcome Elmo's aforementioned limitations, we propose in this article Bert, a source-initiated multicast routing for DCs. Unlike Elmo, Bert scales well with both the number and the size of multicast groups, and does so through clustering, by dividing the members of the multicast group into a set of clusters with each cluster employing its own forwarding rules. In essence, Bert yields much lesser multicast traffic overhead than Elmo by (1) significantly reducing the forwarding header sizes of multicast packets, (2) avoiding spurious downstream packet transmissions and (3) eliminating switch memory usage across DC switches.

The rest of this article is organized as follows. In Section II we discuss related works. Related state-of-the-art works and limitations are described in Section III. We present Bert, the proposed multicast routing scheme, in Section IV. We study and evaluate the performances of Bert compared to those obtained under Elmo in Section V. Finally, we conclude the paper and discuss future directions in Section VI.

II. RELATED WORK

Multicast technology enables group communication where data is addressed to multiple destinations simultaneously allowing for a source to efficiently send to a group of destinations using a single transmission. Improving IP multicast routing protocols such as IGMP [17] and PIM [18] has been the focus of a large body of research. These protocols were originally designed for irregular topologies and Internet traffic which differ significantly from DC networks. Thus, we restrict this section to works that are related to DC networks.

DC multicast has been studied from various perspectives. For example, frameworks proposed in [19]–[21] studied the resource allocation and embedding of multicast virtual networks. They focus mainly on how to place and restore VMs to provide high-performance non-blocking multicast virtual networks while reducing hardware costs in fat-tree DCs. Other works, including ours [22], focus on other multicast routing problems such as scalability and load balancing in DCs. These works can be classified as decentralized [11], [23], SDN-based [10], [24], [25], or source-routed approaches [26]–[29].

A. SOURCE ROUTED MULTICAST

In [26]–[29], bloom filters are used to encode forwarding state inside packets. These approaches import unnecessary traffic leakage (unnecessary multicast transmissions). Moreover, these approaches can't either support large group sizes or large network topology. On the other hand, Elmo [12] exploits programmable switches and the symmetry of DC topologies to compactly encode forwarding states inside packets. However, when considering large multicast group sizes, Elmo header can carry on several hundreds of bytes extra, which increases traffic overhead in the network. Moreover, Elmo incurs many unneeded multicast packet transmissions, yielding higher traffic congestion in the DC's downlinks. Contrary to these approaches, Bert improves scalability with regards to the number and size of multicast groups in DC networks as detailed in Section IV.

B. SDN-BASED MULTICAST

In [10], a centralized controller partitions the address space, and local address aggregation are 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. [9], [30]–[32] focus, on the other hand, on multicast tree reconstruction, but without addressing any scalability issues.

C. DECENTRALIZED MULTICAST

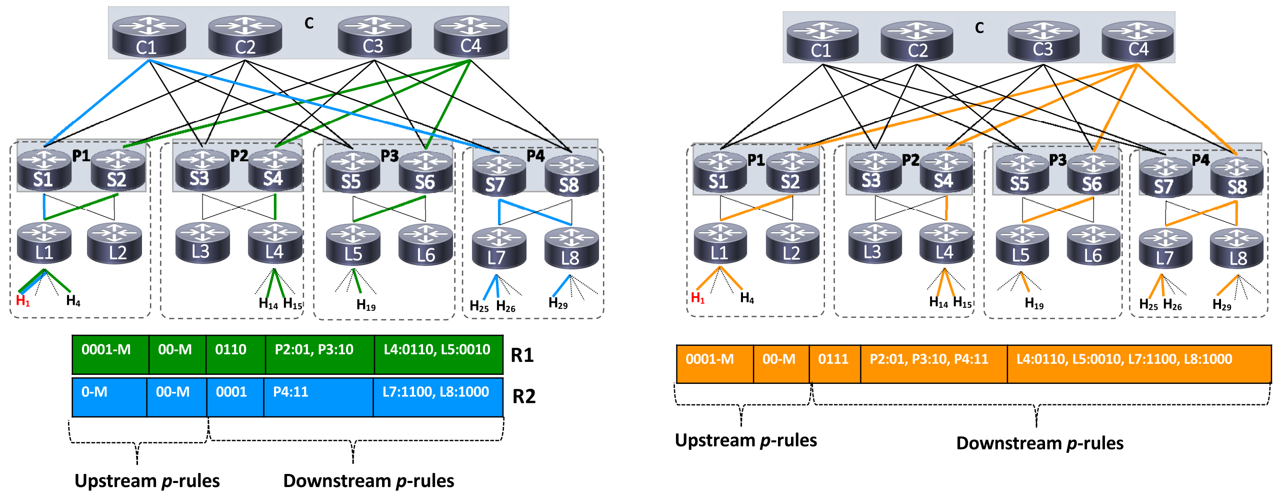
In [11] and [23], the authors present new address mapping schemes and discuss decentralized load balancing strategies, whereas [33] presents prioritized multicast scheduling in DCs. These approaches, however, do not scale well with large numbers of groups due to their reliance on network switches to store and forward multicast packets. Moreover, today's DCs operate under a single administrative domain and no longer require decentralized protocols like PIM and IGMP.

III. DATA CENTER NETWORKING AND TECHNIQUES

Cutting edge cloud DC networks consist of massive numbers of servers enabling multi-tenant occupancy, network virtualization and programmable switches. Next, we discuss state-of-the-art components and current limitations with modern cloud DCs solutions.

1) DC TOPOLOGIES

Large-scale DCs typically are multi-rooted, tree-based topologies (e.g., fat-tree [34] and its variants [35]–[37]). These types of topologies provide large numbers of parallel paths to support high bandwidth, low latency, and non-blocking connectivity among servers. The servers are tree leaves, which are connected to top-of-rack (ToR) (edge/leaf) switches. In general, DCs contain three types of switches, leaf, spine, and core, with each type residing in one layer, as shown in Figure 1. At the lowest layer, leaf (aka edge) switches are interconnected through spine


C
C1
C2
C3
C4
P1
P2
P3
P4
S1
S2
S3
S4
S5
S6
S7
S8
L1
L2
L3
L4
L5
L6
L7
L8
H1
H4
H14 H15
H19
H25 H26
H29
0001-M | 00-M | 0110 | P2:01, P3:10 | L4:0110, L5:0010 | R1
0-M | 00-M | 0001 | P4:11 | L7:1100, L8:1000 | R2
0001-M | 00-M | 0111 | P2:01, P3:10, P4:11 | L4:0110, L5:0010, L7:1100, L8:1000
Upstream p-rules
Downstream p-rules

(a) Bert (b) Elmo

FIGURE 1. An example of a three-tier multicast Clos tree topology with four pods. In this topology, there are 4 hosts under each leaf switch (ToR). 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.

(aka aggregation) switches, which constitute the second layer of switches. The core switches, constituting the top/root layer, serve as connections among the spine switches. With such a DC topology, every server can communicate with any other server using the same number of hops.

2) MULTI-TENANT DCs

In Multi-tenant DCs (like Microsoft Azure [3], Amazon Web Services (AWS) [5], and Google Cloud Platform [4]), a fraction of computing resources (e.g., CPUs, memory, storage, and network) are rented to customers/tenants (e.g., commercial or government organization, or an individual) by means of virtualization technology. These multi-tenant DCs need to guarantee resource isolation and sharing of bandwidth among different tenants. By moving towards multi-tenant DCs, tenants can lower their operational cost 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 [38].

3) VIRTUALIZATION IN DCs

In multi-tenant data centers, computing and network resources are virtualized. Typically, this is done by using software or firmware called a hypervisor [39]. 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 [40], 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 [41]) to guarantee resource isolation and fair share of network resources among tenants.

4) PROGRAMMABLE SWITCHES

Emerging programmable switch ASICs (e.g., Barefoot Tofino [42]) 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 [43]. P4, a language for Programming Protocol Independent Packet Processors, is a recent innovation providing an abstract model suitable for programming the network data plane.

IV. CLUSTERED MULTICAST SOURCE ROUTING

The proposed multicast routing protocol, Bert, expands on Elmo while addressing the multicast scalability issues. Bert adequately splits multicast group destinations into a set of disjoint multicast clusters and encodes forwarding information for each cluster to reduce packet header sizes and eliminate switch memory utilization.

A. ELMO

Elmo [12] is a recently proposed DC multicast routing protocol that scales well with the number of multicast groups. Elmo is a promising source-based routing protocol suitable for modern cloud DCs through its use of virtualization and programmable switches. In Elmo, packet headers are encoded with packet forwarding state/rules to limit the flow state information maintained at DC switches. Elmo exploits the programmable capability of DC switches and the symmetry of DC topologies to compactly encode multicast group information in packets, thereby reducing packet header overhead and, consequently, network traffic load. Furthermore, Elmo's use of programmable switches in multicast environments avoids the need for additional network hardware. These benefits are essential for high performance modern cloud DCs.

Although Elmo has shown to scale well with the number of multicast groups, it still suffers from scalability issues in terms of incurred traffic overhead when facing large group sizes. For example, a packet header can carry on several hundreds of bytes to encode all p-rules (packet rules) [12], incurring excessive network traffic overhead and link congestions. Elmo tries to overcome this by: (1) removing per-hop p-rules from the header as packets traverse the network switches; unfortunately, the downstream spine and leaf switches, which happen to consume most of the header space, are removed last, causing most of the traffic overhead to disseminate over the network topology. (2) Switches in the downstream paths having same or similar bitmaps are mapped to a single bitmap. For example, as shown in Figure 1a, at the leaf layer, L_7 and L_8 can share one p-rule; i.e. L_7, L_8 : 1100, yielding one extra transmission in L_8. However, sharing bitmaps results in extra packet transmissions, which can increase traffic overhead.

In order to overcome the aforementioned challenges of Elmo, we propose Bert, which first clusters the set of multicast destination members into multiple subgroups, then encodes multicast information in packet headers for each of these clusters.

B. MOTIVATING EXAMPLE

To illustrate the limitations of Elmo and motivate the design of the proposed scheme, Bert, we present a detailed example in Figure 1. At a high level, for each multicast group, the controller first computes a multicast tree and corresponding forwarding rules, then installs these rules in the hypervisor of the multicast group source. The hypervisor intercepts each multicast packet and appends the forwarding rules to the packet header. Elmo essentially focuses on how to efficiently encode a multicast forwarding policy in the packet header. Conversely, Bert, in addition to efficiently encoding the forwarding rules, aims to alleviate traffic overhead caused by header size and extra packet transmissions in the downstream paths. The forwarding header consists of a succession of p-rules that include rules for upstream leaf and spine switches, as well as for the downstream core, spine, and leaf switches. Each switch in the multicast tree will remove its p-rules from the header when forwarding the packet to the next layer. For both Elmo and Bert, each multicast packet's journey can be explained in two main phases:

1) UPSTREAM PATH

This path involves *leaf-to-core* switches. The p-rules for upstream switches (leaf and spine) consist of downstream ports and a multipath flag. 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; i.e. ECMP [44]. In Elmo, only one packet goes through upstream paths. Using Figure 1b for illustration, leaf switch L_1 first removes its p-rules $(0001 - M)$ from the packet, then forwards it to the host H_4 as well as multipathing it to any spine switch P_1. The upstream spine switches will do the same to forward the packet to the core switches. Our proposed Bert, on the other hand, first clusters the destination members of the multicast group into multiple (two in the example) clusters, and then sends multiple (two in the example) copies of the packet (with different headers but same payload), one for each cluster; more detail on the clustering part will be provided later. The first packet has the same upstream p-rules as Elmo; i.e. $R1$, while the second packet (i.e. $R2$) does not have any downstream rules for the leaf and spine switches to avoid any extra transmissions. In Bert, although packet duplication incurs some extra (minor) traffic in upstream paths, it results in substantial traffic reduction in downstream paths when compared to Elmo. That is, the overall traffic of both upstream and downstream paths is significantly reduced under Bert when compared to Elmo.

2) DOWNSTREAM PATH

This path involves *core-to-leaf* switches. The p-rules for the core, spine, and leaf switches in the downstream path consist of downstream ports and switch IDs. In the downstream path, the core switches forward the packet to the given pod based on the core switch p-rules. In Elmo, one core switch sends the packet to the spine switches, which in turn forward it (based on the spine switch p-rule) to the leaf switches. The leaf switches do the same to deliver the packet to the destination hosts. Note that because of topology symmetry, any core switch can forward the packet to the destination pods. Referring to the example in Figure 1b again, in Elmo, core switch C sends the packet to $P2$, $P3$ and $P4$ switches (three packets in total), and once the packet arrives at the downstream spine switch, it is then forwarded based on the spine switch p-rules to the leaf switches. These leaf switches do, in turn, the same to deliver the packet to the destination hosts. In Bert, $C4$ forwards the first packet (i.e. $R1$) to $P2$ and $P3$, while $C1$ forwards the second packet (i.e. $R2$) to $P4$ (see Figure 1a). Note that the number of core-pod packets, which is three in the example, is the same in both Elmo and Bert.

This example shows that Bert greatly reduces the header size. For instance, the header size of the first packet ($R1$) and second packet ($R2$) is 40 and 24 bits respectively. To identify switches, we use four bits for each of the spine and leaf switches. Hence, the average header size in Bert is about 32 bits per packet whereas with Elmo it is 62 bits (see Figure 1b). Thus, the average header size for the downstream packet in Bert is $\frac{1}{k}$ of that of Elmo's packet, where k is the number of clusters of the multicast group, a design parameter of Bert.

In Elmo, in order to reduce the header size, switches in downstream paths can share the same p-rules. Referring to the example in Figure 1b, when all leaf switches in the multicast tree share one p-rule—which should then be bitwise OR of all these leaf switches (i.e., L_4, L_5, L_7, L_8 : 1111), Elmo incurs 10 extra packet transmissions. This reduces Elmo's header size to 50 bits, which is still larger than Bert's header. However, these unnecessary packet transmissions can cause switch processing, network traffic, and power consummation overheads.

C. BERT ARCHITECTURE

Bert consists of three main components: a centralized controller, hypervisor switches, and network switches. Figure 2 illustrates our design architecture and is summarized below:

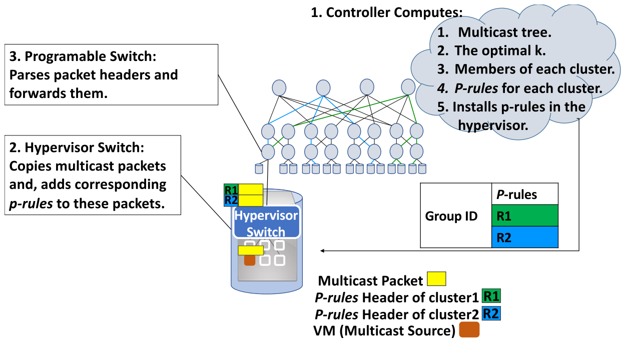

1. Controller Computes:
1. Multicast tree.
2. The optimal k.
3. Members of each cluster.
4. P-rules for each cluster.
5. Installs p-rules in the hypervisor.
3. Programable Switch: Parses packet headers and forwards them.
2. Hypervisor Switch: Copies multicast packets and, adds corresponding p-rules to these packets.
R1
R2
Hypervisor Switch
Group ID
P-rules
R1
R2
Multicast Packet
P-rules Header of cluster1 R1
P-rules Header of cluster2 R2
VM (Multicast Source)

FIGURE 2. Bert's architecture. A multicast group is clustered into two clusters, blue and green. The forwarding rules for each cluster are installed in the multicast source's hypervisor. The hypervisor copies each multicast packet and adds the *p*-rules R1 and R2 to the original and copied packet respectively.

1) CONTROLLER

The controller is responsible for calculating the multicast tree, the traffic cost, and the optimal number of clusters for each multicast group. It also encodes the forwarding rules (p-rules) for each cluster and installs them in the source hypervisor.

2) HYPERVISOR SWITCH

The hypervisor software switch, which is deployed at the end server, is in charge of maintaining p-rules for the multicast groups whose multicast source resides in that server. The hypervisor switch intercepts each multicast packet generated from multicast sources and matches the multicast IP address to the p-rules at the forwarding table. Based on the number of clusters of the corresponding multicast group, the hypervisor switch makes copy/copies of the multicast packet. Finally, the hypervisor switch adds the corresponding p-rules to each copy/copies of the multicast packet.

3) PHYSICAL SWITCHES (NETWORK SWITCHES)

As in Elmo, we assume DCs are running P4 programmable switches [43], which allow for parsing up to 512 bytes of the packet's header size [12]. Bert uses the network switches (programmable switches) only to parse and forward the multicast packets. However, in addition to that, Elmo uses these switches to store some forwarding rules, s-rules, when the size of a multicast group is large.

D. OPTIMAL NUMBER OF CLUSTERS

In Bert, although the number of clusters, k, is a tunable design parameter, there exists an optimal k value that yields the highest performance improvement in overall traffic savings, and in this section we aim to determine it. To gain some insights on the impact of the value of k on the overall (payload plus header) incurred traffic, we show in Figure 3 how the amount of traffic incurred in the upstream (from leaf switches to core switches) and downstream (from core switches to leaf switches) paths varies as k increases for a multicast group scenario with 5000 members and 1500-byte payload. Observe that while the upstream traffic always increases with k, the downstream traffic keeps decreasing as k increases (though the decrease rate is higher for smaller k). However, the overall combined traffic decreases at first, then starts to increase again, with an optimal overall traffic amount being achieved when k is about 7.

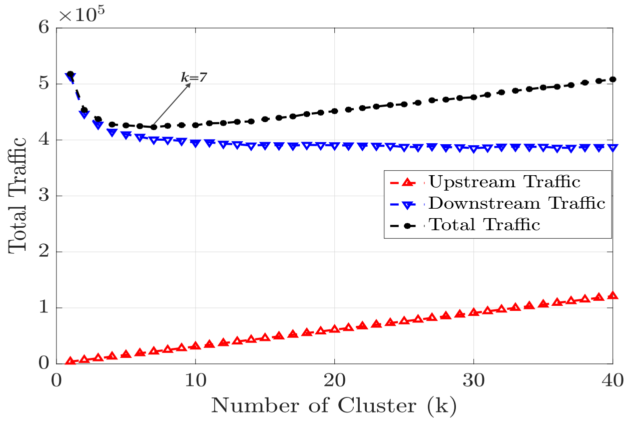


FIGURE 3. Multicast group size is 5000 members. Packet payload $B = 1500$ Bytes. The member placement strategy is leaf-based, described in Section V. The total (payload + header) traffic is minimized when $k = 7$.

For the general scenario, let us consider a multicast group whose members are already placed across the DC servers, and let us denote the forwarding header size by H, the packet payload size by B, and the number of hops between leaf and core switches by h. We want to mention here that H depends on the size of the multicast group, as well as on where the group members are placed in the DC servers, and, hence, it is constant for this considered multicast group. Also, the parameter h is DC-topology specific; for instance, $h = 1$ in 2-tier fat-tree topologies and $h = 2$ in 3-tier fat-tree topologies. The overall (upstream and downstream) traffic incurred by Bert can be expressed as

$$h(H + kB) + \sum_{i=1}^{h} r_i(\frac{H}{k} + B) \qquad (1)$$

where r_i, $1 \leq i \leq h$, is the total number of destination switches in the downstream path at hop i. For example, referring to Figure 1a for illustration, we have $h = 2$, $r_1 = 3$, and $r_2 = 4$. In Eq. (1), the terms $h(H+kB)$ and $\sum_{i=1}^{h} r_i(\frac{H}{k}+B)$ represent the total traffic in the upstream and downstream paths, respectively.

Note that, like header size H, the parameter r_i depends on the multicast group size and on the placement of the group members across the DC servers, but not on the number k of clusters chosen by the multicast group. After simple calculation, the optimal value of k that minimizes the total traffic can be expressed as $(\frac{x}{h}\sum_{i=1}^{h} r_i)^{1/2}$ where $x = H/B$ is the faction of the header size to the payload size. Since this optimal value may not be an integer, for practical and evaluation purposes, we set the optimal k to the closest integer. We want to mention here that, as will be explained in Section IV-E, group members are clustered based on the pods as opposed to the leaf switches to prevent redundant packet transmissions. Therefore, we also restrict k to be lesser than the number of pods.

E. MULTICAST GROUP CLUSTERING

Bert aims to reduce the control message traffic by reducing the traffic overhead that Elmo incurs in the downstream paths, as well as the size of the multicast packet header. As illustrated in the motivating example given in the previous section, Bert achieves this goal by clustering the set of group members into k clusters using Algorithm 1. The optimal k from Section IV-D is calculated for each multicast group independently. Before presenting the clustering approach of Bert, we introduce the following notations/parameters of the studied three-tier DC: throughout, let us denote the number of pods by n, the number of ports per-leaf switch by l, the number of leaf switches per pod by m. Note that although in traditional fat-tree DC, $m = n/2$ and $l = n/2$, for the sake of keeping our technique applicable to any tree-based DC topologies, we use the general parameter notation. Also, let $L_{g,i}^{j}$ be the l-bit binary vector, corresponding to the jth leaf switch belonging to the ith pod, where $1 \leq i \leq n$ and $1 \leq j \leq m$, with each bit corresponding to one port of the leaf switch and taking 1 when the port is serving a member of the multicast group g and 0 otherwise. For each multicast group g and each pod i, let $L_{g,i}$ be the concatenation of the m l-bit vectors of the m leaf switches belonging to pod i. That is, $L_{g,i} = L_{g,i}^{1}||L_{g,i}^{2}||\ldots||L_{g,i}^{m}$; here, $L_{g,i}$ is a binary vector of size $l \times m$.

Back to Bert's clustering method, we begin by mentioning that in Bert, we choose to cluster group members based on the pods as opposed to the leaf switches. That is, for each multicast group g, Bert clusters the set of n vectors, $L_{g,i}$ with $1 \leq i \leq n$, as opposed to the set $n \times m$ of vectors, $L_{g,i}^{j}$ with $1 \leq i \leq n$ and $1 \leq j \leq m$. This choice is supported shortly via an example. Bert uses K-Means clustering algorithm with the Hamming distance as the distance metric, where the Hamming distance between two binary vectors is simply the number of bit positions in which they differ. For each multicast group g, K-Means algorithm takes as an input the set of n vectors, $L_{g,i}$ with $1 \leq i \leq n$, and the number of clusters, k, and outputs k clusters, with each cluster specifying a subset of the pods that need to belong to the same cluster. Once clustering is done, the p-rules of each cluster are created by the hypervisor, which makes one copy of the multicast packet (data + header/p-rules) for each cluster. For example, in Figure 1a, when the hypervisor of host H_1 receives the multicast packet, it creates another copy of this packet, and adds the $R1$ rules to the first packet and the $R2$ rules to the second packet.

F. POD-BASED VERSUS LEAF-BASED CLUSTERING

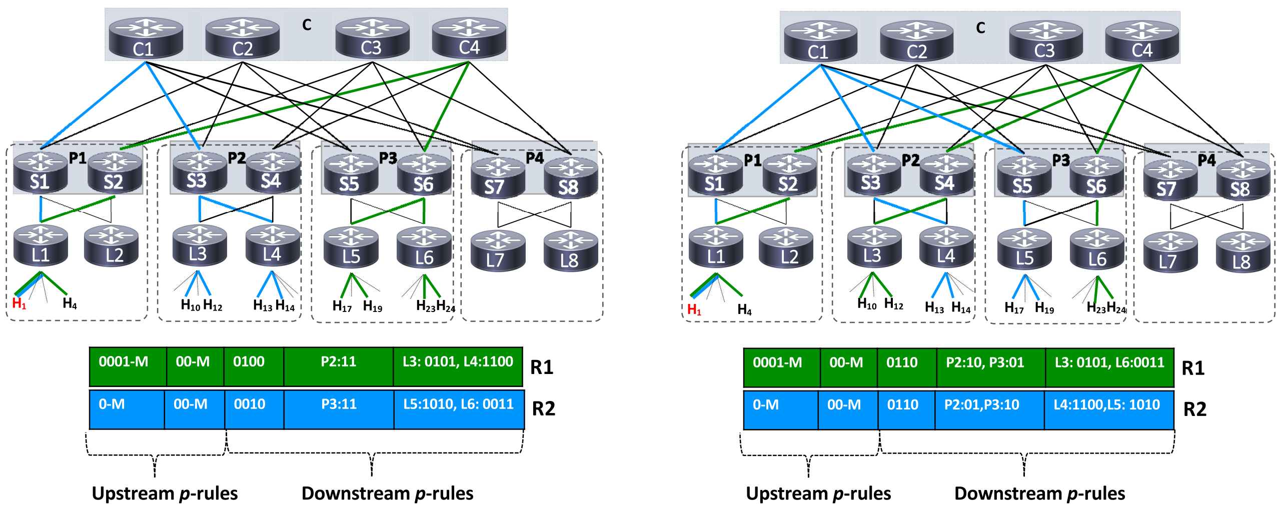
In Section IV-E we mentioned that Bert adopts pod-based clustering rather than leaf-based. The reason for that is as follows: if we cluster the downstream pods based on the p-rules for the downstream leaf switches regardless of which pod they belong to, extra packets transmissions will occur at the core and spine switches in the downstream path. For example, in Figure 4b, when clustering is based on leaf switches only and when using the Hamming distance similarity, L_4 and L_5 will be clustered in the same cluster (i.e. $R2$), and L_3 and L_6 will be clustered in the other/second cluster (i.e. $R1$). In this case, because $L3$ and $L4$ are in the same pod (pod 2) but they are in different clusters, the packet will be sent twice at both core and spine downstream layers. The same thing happens with L_5 and L_6. To avoid this, Bert adopts a clustering choice that is locality-aware of leaf switches (see Figure 4a).

Algorithm 1 Clustering of Multicast Group

```
Input: Multicast Group $G$
Output: $k$ clusters $G_1, G_2, .., G_k$
 1: calculate $TR_{Bert}$
 2: int($k$) = arg min $TR_{Bert}(k)$
 3: if $k > n$ then
      %$n$ is the number of pods.
 4:   $k = n$
 5: end if
 6: if $k > 1$ then
 7:   initialize $M$
 8:   for $i \in \{1, \ldots, n\}$ do
 9:     for $j \in \{1, \ldots, m\}$ do
10:       for $l \in \{1, \ldots, l\}$ do
11:         if $l \in G$ then
12:           $M(i, j * l) = 1$
13:         else
14:           $M(i, j * l) = 0$
15:         end if
16:       end for
17:     end for
18:   end for
19:   k-means($M$, $k$, hamming distance)
20:   return $k$ clusters, with each cluster specifying a subset
      of the pods that need to belong to the same cluster.
21: else
22:   exit   %no need for clustering.
23: end if
```

G. KEY FEATURES OF BERT

Bert strikes to balance between two conflicting objectives: maintaining low network overhead while not increasing CPU overhead substantially. Here CPU overhead is captured in


C
C1
C2
C3
C4
P1
P2
P3
P4
S1
S2
S3
S4
S5
S6
S7
S8
L1
L2
L3
L4
L5
L6
L7
L8
H1
H4
H10 H12
H13 H14
H17 H19
H23H24
0001-M | 00-M | 0100 | P2:11 | L3: 0101, L4:1100 R1
0-M | 00-M | 0010 | P3:11 | L5:1010, L6: 0011 R2
Upstream p-rules
Downstream p-rules
0001-M | 00-M | 0110 | P2:10, P3:01 | L3: 0101, L6:0011 R1
0-M | 00-M | 0110 | P2:01,P3:10 | L4:1100,L5: 1010 R2

(a) Locality-aware clustering (b) Locality-oblivious clustering

FIGURE 4. Clustering choice example of a three-tier multicast Clos tree topology with four pods. In this topology, there are 4 hosts under each leaf switch (ToR). H_1 is the multicast source and H_4, H_{10}, H_{12}, H_{13}, H_{14}, H_{17},H_{19},H_{23}, and H_{24} are the destinations of the multicast group.

terms of number of packet replications the source needs to make, which in some sense captures other aspects of CPU cost, like processing delay, CPU power consumption, etc. That is, Bert avoids substantial overhead caused by unicast and overlay mulicast [45] approaches where all packet replications occur at the host while reducing traffic cost and switch memory usage. Although Elmo avoids packet replications caused by Bert at the source, Bert surpasses Elmo in the following regards:

1) REDUCING PACKET HEADER SIZE

In multi-rooted Clos topologies, unlike traffic load in upstream paths which are equally distributed, downstream paths are much heavier and are always the main bottleneck of the network. This is because, in these types of topologies, the upstream routing is fully adaptive, while the downstream routing is deterministic. Moreover, the multicast workload may make this worse because multicast packets are replicated at the downstream paths in order to reach each group member. In Elmo, by adding the *p*-rules to the packet, a data packet may have several hundreds of bytes of forwarding rules for each packet. In Bert, the average header size for the downstream packet is inversely proportional to the number of clusters k, i.e., $\frac{1}{k}$, of that of Elmo's packet, as explained in the previous subsection.

2) REDUCING NUMBER OF EXTRA TRANSMISSIONS

One of the key designs of Elmo that reduces header sizes is to map multiple switches in downstream paths to a single *p*-rule, as a bitwise OR of their individual *p*-rule. Unfortunately, this strategy introduces huge amount of unwanted redundant packets in downstream switches. These redundant packets waste network bandwidth and induce network switches processing overhead. Bert on the other hand, significantly reduces the header size without incurring any redundant transmissions in downstream paths.

3) ELIMINATE SWITCH MEMORY USAGE

In Elmo, when a multicast group is large and cannot be encoded entirely in the packet header (i.e. header size $\geq$ 512 bytes), network switches are used to store forwarding rules. Because header sizes in Bert are significantly smaller than in Elmo, Bert does not use switch memory at all, which in turn conserves switch memory resources.

V. PERFORMANCE EVALUATION

Multicast routing in large DC networks should be simple to implement, scalable, robust, use minimal network overhead and consume minimal memory resources. Scalability can be evaluated not only in terms of the network overhead cost in the presence of a large number of groups but also by the number of participants per group and by groups whose participants change often over time. For an accurate and fair comparison with Elmo, we mimic the numerical experimental setup of [12] and [10] where full-sized cloud DC topologies are simulated, rather than implementing Bert on a small testbed. Running packet-level network simulations is not feasible at such a large scale (e.g. we simulate tens of thousands of groups each with hundreds or thousand members).

A. EXPERIMENT SETUP

In this section, we conduct a series of experiments to evaluate and compare Bert and Elmo in terms of multicast scalability on full-sized cloud DC topologies. Multi-tenant environments are simulated while adjusting parameters such as the number of tenants, number of VMs per tenant, VM placement strategies, and number of multicast groups and their sizes. We also consider P4 switches in the DC network so that network

TABLE 1. Simulation environment.

Component	Description
Topology	Symmetric 3-tierd fat-tree
Network size	48 pods
Leaf switches per pod	24 switches
Spine switches per pod	24 switches
Hosts per pod	24 hosts
Total hosts	27,648 hosts
Tenants	3000 tenants
Maximum VMs per host	20 VMs
VMs per tenant	`exprnd`(min=10, μ=178, max=5000)
Maximum header size	512 bytes
Payload size	1500 bytes

operators can specify how a physical switch processes and steers packets. Table 1 depicts the components used for our simulation environment.

1) **DC Topology**: we use a symmetric 3-tierd fat-tree DC topology consisting of 48 pods each with 24 leaf switches where each leaf switch is connected to 24 hosts serving 27,648 hosts in total. The 3-tierd fat-tree topology is the most widely used DC topology network.
2) **Tenants and their VMs**: our DC network is populated by 3000 tenants where the number of VMs per tenant is exponentially distributed between 10 and 5,000. Each physical server can host at most 20 VMs and tenant VMs do not share the same host.
3) **VM placement**: we consider three placement strategies when mapping a tenant's VMs to a physical host: (i)
   a) *pod-based* where a pod is selected uniformly at random and tenant VMs are greedily placed in all the available pod hosts. If more VMs need to be placed, another pod is selected at random and the process is repeated. In this strategy, tenant VMs tend to be too close.
   b) *leaf-based* where a pod and leaf are selected uniformly at random and tenant VMs are placed based on leaf host availability. If more VMs need to be placed, a pod and leaf are selected at random and the process is repeated. Here, tenant VMs tend to be placed close to each other but not too close as in the pod-based strategy.
   c) *random* where a pod, leaf, and host are selected uniformly at random to host tenant VMs. All placement strategies are repeated until there are no tenant VMs to be placed.
4) **Multicast groups**: the number of multicast groups assigned to each tenant is proportional to the tenant's size. Each tenant's group sizes are uniformly distributed between the minimum group size (i.e. 5) and the entire tenant size. Note that varying the number of groups and group sizes is a suitable strategy to measure scalabilty.

In addition, our setup combines p-rules only when bitmaps are the same, thus, preventing extra packet transmissions in downstream switches. This, in turn, conserves overall network resources.

B. PERFORMANCE ANALYSIS

We evaluate Bert and Elmo scalability behavior from two perspectives: 1) A single multicast group with different member sizes (e.g. 100-5000 members), and 2) different number of multicast groups (e.g. 10K-100K multicast groups) each with different member sizes. For each perspective, we analyze Bert's clusters, header sizes, traffic overhead, switch memory costs, and source packet replications using the aforementioned VM placement strategies.

1) OPTIMAL NUMBER OF CLUSTERS

The number of Bert clusters depends on both the placement strategy and the group size. In this analysis, the optimal number k of clusters is based on our calculations from Section IV-D. In pod-based and leaf-based placement strategies, where group members are close to one another, we observe from the results shown in Figure 5 that the optimal number of clusters is small particularly when group sizes are small (i.e. less than 1000 members). However, this number slightly increases as group sizes increase. For example, when group sizes are around 500 members, optimal k is 1 for both pod-based and leaf-based placement strategies. Moreover, when group sizes are around 3500, optimal k is 4 and 5 in pod-based and leaf-based placement strategies, respectively. Conversely, the random placement strategy requires larger header sizes and is more optimal when the number of clusters is large. For example, Bert generates 5 to 44 clusters as group sizes increase. We verify our results by calculating and showing in Figure 6 the average optimal number of clusters achieved by averaging over multiple different groups. When group members are placed very close to one another (i.e. pod-based), the average of optimal k tends to be small (i.e. 1). With a random placement strategy, where header size usually is large, optimal k is around 14.

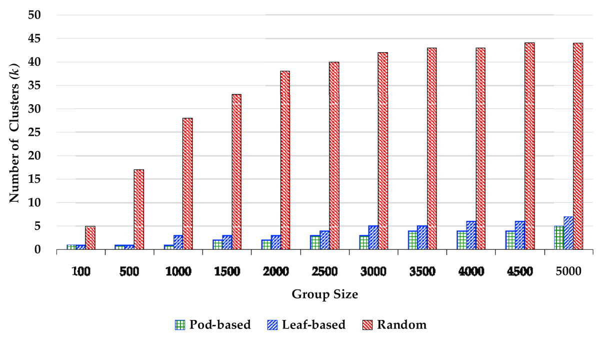


FIGURE 5. Number of clusters as a function of group size.

2) HEADER SIZE

Figure 7 shows the header size as a function of group size using a single multicast group. For each placement strategy, we calculate the header size for one multicast group

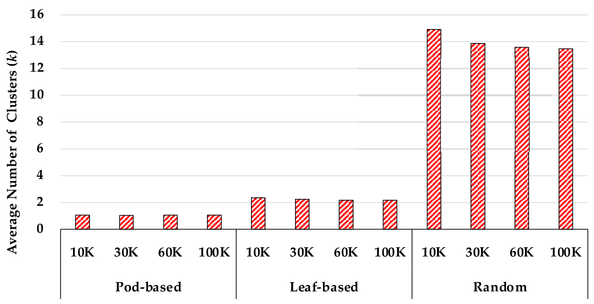


FIGURE 6. Average number of clusters as a function of number of multicast groups.

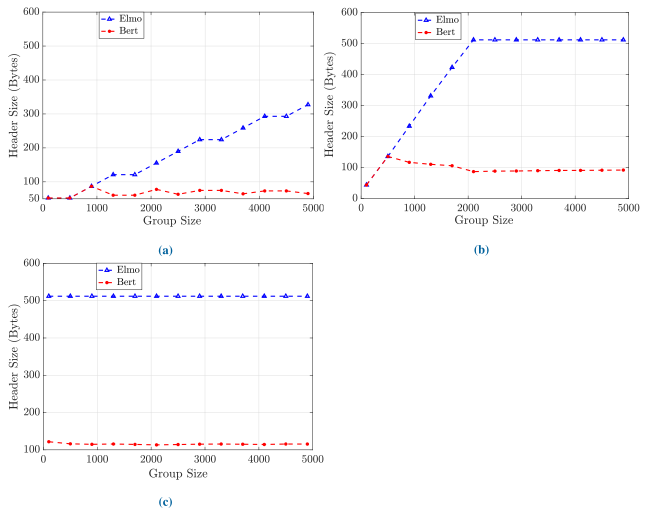


FIGURE 7. Header size as a function of group size: (a) pod-based, (b) leaf-based and (c) random.

as the group size is varied between 100 and 5000. Elmo shows to be sensitive to both the placement strategy used and the size of the multicast group. In each placement strategy, Elmo's header size increases as the size of the group increases whereas Bert adapts and keeps the forwarding header at the smallest possible size by clustering the multicast groups. Here, Bert obtains the optimal number of clusters using the process described in Section IV-D.

We notice as group size increases, more switches need to be encoded in the downstream path resulting in larger header sizes (i.e. 1024 switches require 10 bits to identify). In Figure 7a, with the pod-based placement strategy, Elmo requires 100 to 300 bytes to handle group sizes of 1000 to 5000 members while the average header size of Bert's clusters is roughly 75 bytes. In Figure 7b, with the leaf-based placement strategy, Elmo requires a maximum header size of 512 bytes when the number of members reaches 2000 whereas the average header size of Bert's clusters do not exceed 135 bytes for all group sizes. Finally with the random placement strategy, Elmo is forced to use the maximum header size for all group sizes where Bert only needs 100 bytes as shown in Figure 7c.

We observe that if group members (i.e. VMs) are placed too close to one another as in the pod-based strategy, we encode fewer downstream switches in the packet header and thus, header sizes are small. However, if group members are placed at random, more downstream switches are encoded into the header which, in turn, results in larger header sizes. In addition, as multicast group sizes increase, more switch information needs to be encoded. Bert's ability to calculate the optimal k and cluster multicast groups into k clusters allows for smaller header sizes to be used. That is, header sizes are much smaller compared to Elmo when $k > 1$.

Figure 8 shows the average header size as the number of multicast groups is varied from 10K to 100K groups. Bert significantly reduces the header size by 55% and 73% for the leaf-based and random placement strategies respectively. In addition, Bert shows a 10% improvement in header size in the pod-based placement strategy. Note that Elmo's average header size is comparable to Bert only in the pod-based placement strategy particularly when group sizes are small (as previously shown in Figure 7a).

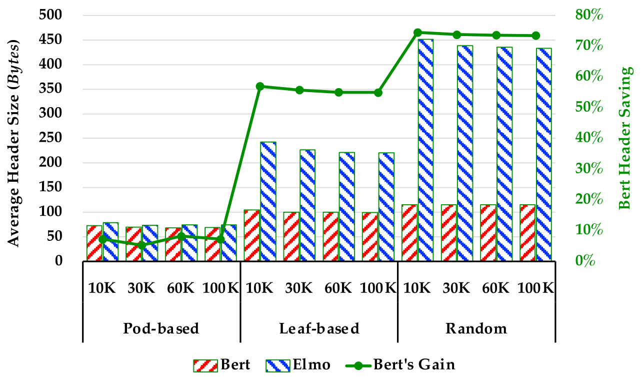


FIGURE 8. Average header size as a function of the number of multicast groups.

3) TRAFFIC COST

In this experiment, we evaluate the overall network overhead incurred by Elmo and Bert (normalized to Elmo). In Figure 9, we show the normalized traffic cost in terms of network overhead incurred in the upstream and downstream paths as described in Section IV. The figure evaluates pod-based (Figure 9a), leaf-based (Figure 9b), and random (Figure 9c) placement strategies using a multicast group with 5000 members and a packet payload of 1500 bytes. Observe that Bert contributes negligible network overhead in the upstream paths (less than 0.1% of total traffic), however, more than makes up for it in the downstream paths with a total improvement of 11% for pod-based placement, 18% for leaf-based placement, and 14% for random placement compared to Elmo. Bert does incur overhead in the upstream path when clusters or subgroups are greater than one, however, reduces network traffic costs in the downstream paths resulting in overall improvements in network performance.

Figure 10 illustrates the network traffic cost as a function of group size using a single multicast group with different

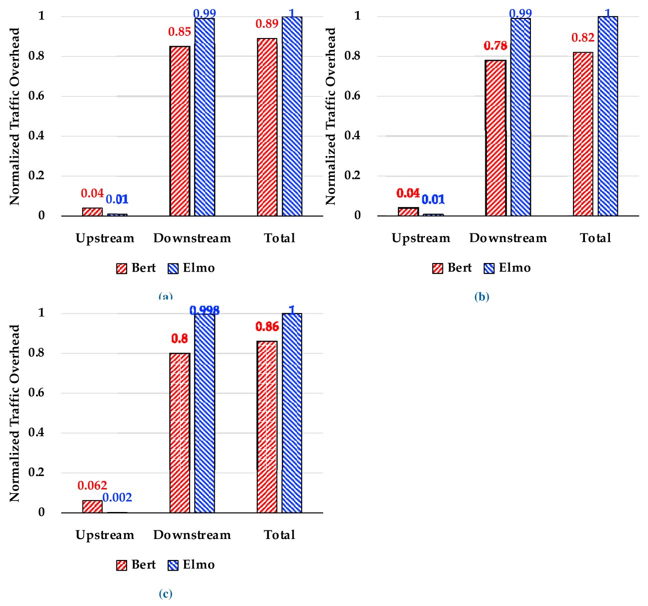


FIGURE 9. Upstream vs downstream traffic cost (d=5000): (a) pod-based, (b) leaf-based and (c) random.

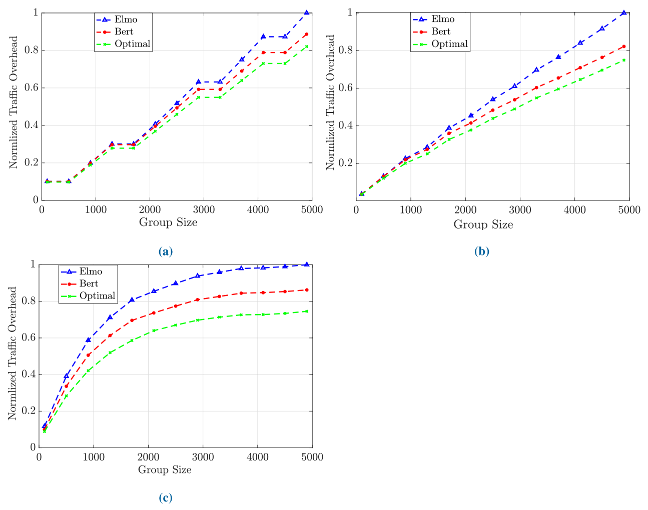


FIGURE 10. Traffic cost as a function of group size: (a) pod-based, (b) leaf-based and (c) random.

VM placement strategies. As shown in Figure 10a (pod-based placement) and Figure 10b (leaf-based placement), when multicast group sizes are small (i.e. few hundreds), Elmo and Bert both incur additional traffic overhead compared to the optimal (zero header overhead). However, as multicast group sizes increase, Elmo contributes more traffic overhead (25%) than Bert (8%) compared to the optimal as shown in Figure 10b when group sizes are greater than 4000 in the leaf-based placement strategy. Similar behavior is exhibited in Figure 10c for all group sizes with the random placement strategy where Elmo and Bert contribute 26% and 13% more traffic compared to the optimal. Thus, by minimizing header size, Bert is able to minimize total traffic overhead.

In addition, in Figure 11 we measure the average traffic cost as a function of the number of multicast groups. Bert shows improvements in traffic cost for both leaf-based and random placement strategies. However, when most multicast members are within the same pod, Elmo and Bert perform similarly as shown in Figure 11a.

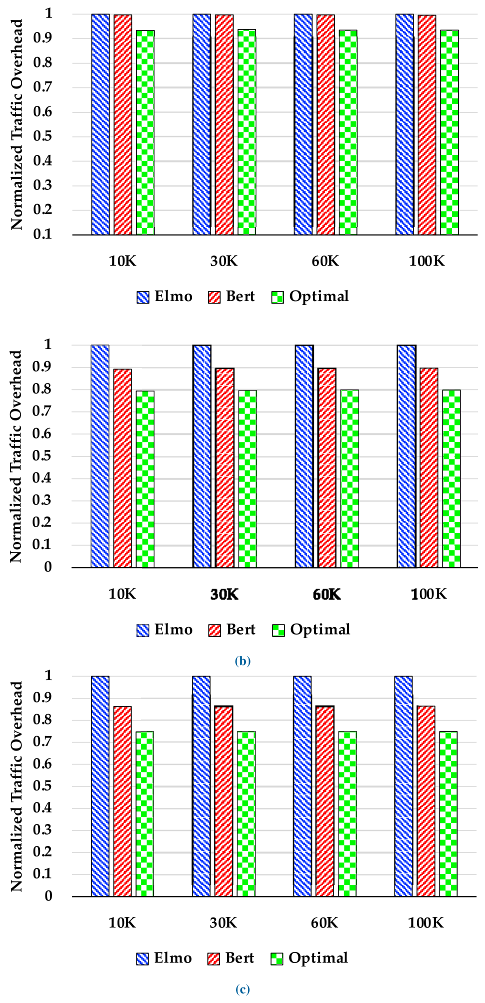


FIGURE 11. Traffic cost as a function of the number of groups: (a) pod-based, (b) leaf-based and (c) random.

In Figure 12 traffic cost is analyzed using 100K multicast groups and varying the packet payload sizes between 64 bytes (minimum transmission unit) and 1500 bytes. Here we emphasize the benefit of Bert compared to Elmo when different packet payload sizes are used. Bert provides substantial benefits when smaller payload sizes are used. For example, with 64-byte payload sizes, Bert reduces traffic overhead by 10%, 67% and 73% in pod-based, leaf-based, and random placement strategies, respectively. For large payload sizes (i.e. 1500 bytes), Bert reduces traffic overhead by 10% and 14% in leaf-based and random placement strategies, respectively. That is, when packet payload sizes are small Bert outperforms Elmo particularly in both leaf-based and random placement strategies.

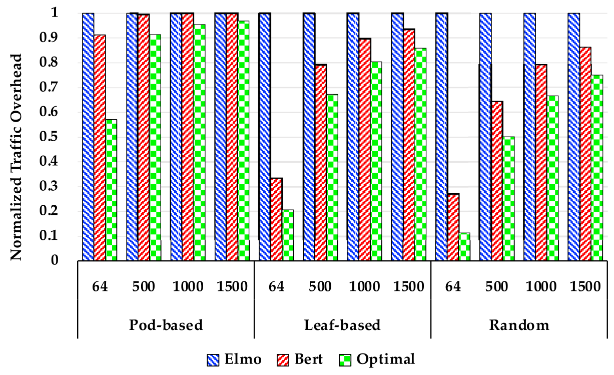


FIGURE 12. Traffic cost as a function of packet payload size in bytes.

4) SWITCH MEMORY COST

Programmable switches typically set restrictions on the packet header sizes they can parse (i.e., 512 bytes) [12]. In Elmo, when forwarding headers reach their maximum size and not all forwarding rules can be encoded into the header, switch memory is exploited to store the remaining rules. In this experiment, we calculate the total memory usage at the downstream switches for Bert and Elmo using a maximum header size of 512 bytes.

In Figures 13 and 14, we evaluate and show the total switch memory utilization by a single and multiple multicast groups, respectively. In both leaf-based and random placement strategies, Bert shows drastic improvements in switch memory utilization whereas Elmo must use switch memory to store its larger header sizes. Unlike Elmo, Bert does not use switch memory to store forwarding rules as depicted in Figures 13 and 14. Furthermore, in Elmo, when both the size and the number of multicast groups increase, the switch memory usage also increases. In the case of pod-placement strategy, since the header size of Elmo and Bert do not exceed 512 bytes for all sizes of the multicast group, there is no need to store rules and hence switch memory usage is zero for both Elmo and Bert (figure for this scenario is not included).

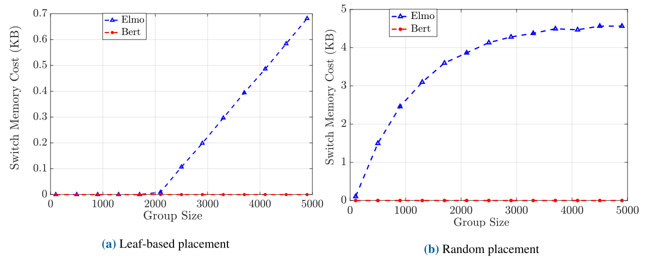


FIGURE 13. Switch memory use with one multicast group.

5) SOURCE PACKET REPLICATIONS

Although Bert significantly reduces traffic overhead and switch memory usage, replicating packets at the source hypervisor might incur extra overhead. However, this overhead is much lesser than that of unicast-based multicast protocols, where separate end-to-end connections are needed for

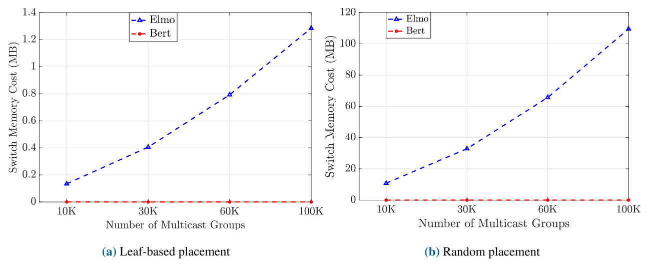


FIGURE 14. Switch memory use with multiple multicast groups.

each receiver, and than that of overlay-based multicast protocols [45], where all multicast packet replications occur at the source. Figure 15 depicts the number of packet replications incurred at the source for different group sizes. Regardless of the group size or placement strategy, it is shown that Elmo maintains minimal overhead since only one packet replication is done at the source. Contrarily, unicast and overlay multicast techniques replicate all packets at the source. For example, if there are 2000 multicast members, overlay multicast would need to replicate the packet 2000 times. This high number of replications can inflate CPU overhead (i.e. processing delay, CPU power consumption, etc.). In Bert, packets are replicated per cluster where the number of packet replications depends on the placement strategy used and group size. For instance, given a group size of 2000, the number packet replications in pod-based and leaf-based placement strategies is 2 and 3, respectively. For a group size of 100, only one packet replication is done at the source in both pod-based and leaf-based placement strategies.

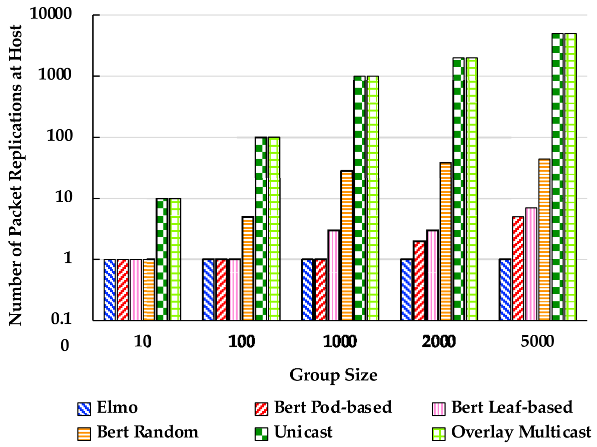


FIGURE 15. Number of packet replications as a function of group size.

For the random placement strategy, the number of packet replications ranges between 1 and 44 where group sizes range between 10 and 5000 members. In other words, random placement requires more clusters (more packet replications at the source) because forwarding rules (header sizes) are too large to encode even for small group sizes (e.g. $k = 5$ when group members are 100). This scenario is challenging for both Bert and Elmo. In fact, Elmo suffers when group members are dispersed across the network (i.e. random placement), even when combined rules and switch memory are used; Elmo's traffic overhead is shown to increase by 123% compared to overlay multicast in such scenarios [12]. Please note that, in Bert, the number of clusters (k) is a tunable design parameter, and can be set to a small number such as 2 or 3, which, in turn, can avoid extra overhead at the source. However, this can also limit the amount of improvement in network traffic and switch memory usage that Bert can achieve.

VI. CONCLUSION AND FUTURE DIRECTIONS

We proposed Bert, a scalable, source routed multicast scheme for cloud data centers. Bert builds on existing approaches to better suit state-of-the-art cloud data center networks. By wisely clustering a multicast group into subclustes, Bert alleviates traffic congestion at downstream paths (usually highly congested links) by reducing both the packet header sizes and the number of extra packet transmissions as well as eliminating switch memory utilization. Experiments show that Bert can reduce traffic overheads between 73–14% compared to Elmo for 64-byte and 1500-byte packets. In brief, unicast/overlay-based approaches incur excessive CPU overhead, but minimal network overhead; Elmo reduces CPU overhead substantially but at the price of increasing network overhead; Bert offers a better balance between the two.

Current advancements in DC networks unlock a variety of applications and open challenges. 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 [46]. Very few analyze multicast communication patterns which frequently arise in cloud DCs. Thus, we believe there is still a need for studies to analyze multicast behavior in detail such as the number of multicast groups and their sizes in real-world data centers. These studies would not only help in understanding communication patterns in general but also in evaluating newly proposed multicast approaches for DCs. In addition, many works address load balancing and resource scheduling issues [47]–[49] in DC networks pertaining to unicast communication and do not consider multicast traffic [50]–[56]. As a result, there is a clear need for load balancing protocols that handle both unicast and multicast traffic. In addition to scalability and load balancing, a reliable multicast protocol should easily 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. Evaluating Bert on a small-scale testbed for proof-of-concept while considering multicast load balancing and group membership dynamics are left for future work.

REFERENCES

[1] *Amazon Cloud Has 1 Million Users*. Accessed: Feb. 19, [Online]. Available: https://arstechnica.com/information-technology/2016/04/amazon-cloud-has%-1-million-users-and-is-near-10-billion-in-annual-sales

[2] 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.

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

[4] *Google Compute Engine*. Accessed: Feb. 10, 2020 [Online]. Available: https://cloud.google.com/compute/

[5] *Amazon Aws Kernel Description*. Accessed: Feb. 10, 2020. [Online]. Available: https://aws.amazon.com/

[6] 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.

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

[8] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, "Toward energy-efficient cloud computing: Prediction, consolidation, and overcommitment," *IEEE Netw.*, vol. 29, no. 2, pp. 56–61, Mar. 2015.

[9] A. Iyer, P. Kumar, and V. Mann, "Avalanche: Data center multicast using software defined networking," in *Proc. 6th Int. Conf. Commun. Syst. Netw. (COMSNETS)*, Jan. 2014, pp. 1–8.

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

[11] 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.

[12] 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.*, 2019, pp. 458–471.

[13] D. Li, M. Xu, M.-C. Zhao, C. Guo, Y. Zhang, and M.-Y. Wu, "RDCM: Reliable data center multicast," in *Proc. IEEE INFOCOM*, Apr. 2011, pp. 56–60.

[14] 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.

[15] W. Cui and C. Qian, "Dual-structure data center multicast using software defined networking," 2014, *arXiv:1403.8065*. [Online]. Available: http://arxiv.org/abs/1403.8065

[16] *Barefoot Tofino: World's Fastest P4-Programmable Ethernet Switch Asics*. Accessed: Feb. 10, 2020. [Online]. Available: https://barefootnetworks.com/products/brief-tofino/

[17] 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*, Standard RFC 4604 (Proposed Standard), Internet Engineering Task Force, 2006.

[18] 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.

[19] J. Duan and Y. Yang, "Placement and performance analysis of virtual multicast networks in fat-tree data center networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 10, pp. 3013–3028, Oct. 2016.

[20] S. Ayoubi, C. Assi, Y. Chen, T. Khalifa, and K. B. Shaban, "Restoration methods for cloud multicast virtual networks," *J. Netw. Comput. Appl.*, vol. 78, pp. 180–190, Jan. 2017.

[21] J. Duan and Y. Yang, "MCL: A cost-efficient nonblocking multicast interconnection network," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 9, pp. 2046–2058, Sep. 2018.

[22] J. Alqahtani and B. Hamdaoui, "Bert: Scalable source routed multicast for cloud data centers," in *Proc. Int. Wireless Commun. Mobile Comput. (IWCMC)*, Jun. 2020, pp. 1752–1757.

[23] F. Fan, B. Hu, and K. L. Yeung, "Distributed and dynamic multicast scheduling in fat-tree data center networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2016, pp. 1–6.

[24] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[25] Z. Guo, Y. Xu, M. Cello, J. Zhang, Z. Wang, M. Liu, and H. J. Chao, "JumpFlow: Reducing flow table usage in software-defined networks," *Comput. Netw.*, vol. 92, pp. 300–315, Dec. 2015.

[26] P. Jokela, A. Zahemszky, C. Esteve 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.

[27] 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.

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

[29] 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.

[30] R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "PortLand: A scalable fault-tolerant layer 2 data center network fabric," in *Proc. ACM SIGCOMM Conf. Data Commun. - SIGCOMM*, 2009, pp. 39–50.

[31] 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.

[32] 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.

[33] S. Luo, H. Yu, K. Li, and H. Xing, "Efficient file dissemination in data center networks with priority-based adaptive multicast," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 6, pp. 1161–1175, Jun. 2020.

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

[35] 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. SIGCOMM*, 2009, pp. 51–62.

[36] 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.

[37] 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.

[38] *What is a Hypervisor*. Accessed: Oct. 25, 2020. [Online]. Available: https://www.vmware.com/topics/glossary/content/hypervisor

[39] *Netflix on Aws*. Accessed: Oct. 25, 2020. [Online]. Available: https://aws.amazon.com/solutions/case-studies/netflix/

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

[41] 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 RFC 7348, 2014, pp. 1–22.

[42] *Intel Programmable Ethernet Switch Products*, Accessed: Oct. 25, 2020. [Online]. Available: https://www.intel.com/content/www/us/en/products/network-io/programmabl%e-ethernet-switch.html

[43] 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.

[44] C. Hopps, "Analysis of an equal-cost multi-path algorithm," RFC Editor, Redmond, WA, USA, Tech. Rep. RFC2992, 2000. [Online]. Available: https://www.rfc-editor.org/rfc/rfc2992.txt

[45] *Overlay Multicast in Amazon Virtual Private Cloud*. Accessed: Dec. 13, 2020. [Online]. Available: https://aws.amazon.com/articles/overlay-multicast-in-amazon-virtual-pri%vate-cloud/

[46] 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.

[47] M. Dabbagh, B. Hamdaoui, and A. Rayes, "Peak power shaving for reduced electricity costs in cloud data centers: Opportunities and challenges," *IEEE Netw.*, vol. 34, no. 3, pp. 148–153, May 2020.

[48] P. Delgado, F. Dinu, A.-M. Kermarrec, and W. Zwaenepoel, "Hawk: Hybrid datacenter scheduling," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2015, pp. 499–510.

[49] 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.

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

[54] 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, Sep. 2015.

[55] P. Wang, G. Trimponias, H. Xu, 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.

[56] 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.

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 the academic Scholarship from Najran University, in 2012, to support this Ph.D. research work. His research interests include data center networking and cloud computing.

HASSAN H. SINKY received the M.S. and Ph.D. degrees from Oregon State University, USA. Since 2017, he has been an Assistant Professor with the College of Computer and Information Systems, Umm Al-Qura University, Saudi Arabia. He specializes in large urban wireless communication networks, content-delivery and content-centric networks, quality of service and quality of experience methods, cross-layer assisted multi-path TCP, and seamless handoffs in wireless mobile scenarios.

BECHIR HAMDAOUI (Senior Member, IEEE) received the M.S. degree in ECE, the M.S. degree 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 computer networking, network security, and wireless communication, with a current focus on edge cloud computing, network analytics, autonomous systems, dynamic spectrum management, 5G systems, and datacenters. 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/symposia, including the 2017 INFOCOM Demo/Posters Program, the 2016 IEEE GLOBECOM Mobile and Wireless Networks Symposium, and many others. He also serves as the Chair for the IEEE Communications Society's Wireless Communication Technical Committee (WTC). He served as a Distinguished Lecturer for the IEEE Communication Society in 2016 and 2017. 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.