AN ABSTRACT OF THE DISSERTATION OF

Jarallah Alqahtani for the degree of Doctor of Philosophy in Computer Science presented on November 30, 2021.

Title: Source-Routed Multicast Schemes for Large-Scale Cloud Data Center Networks

Abstract approved: ____

Bechir Hamdaoui

Data centers (DCs) have been witnessing unprecedented growth in size, number and complexity in recent years. They consist of tens of thousands of servers interconnected by fast network switches, hosting and enabling numerous applications with various traffic characteristics and requirements. As a result, DC networks have been presented with several unique challenges, pertaining to the scaling and allocation of network resources during the forwarding and moving of data across the different DC servers. Traffic routing in general and multicast routing in particular are important functions in DC networks, especially that modern cloud DCs tend to exhibit one-to-many communication traffic patterns. Unfortunately, recent multicast routing approaches that adopt IP multicast 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. In this thesis, we propose a set of new, complementary schemes that overcome these challenges. More specifically, firstly, we study existing DC network topologies, and propose Circulant Fat-Tree topology, an improvement over the traditional Fat-Tree topology with better properties to suit nowadays DC networks. Then, we review and classify recent studies that investigate and measure the traffic behavior of operational DC networks. We focus on the way they collect the traffic as well as on the key findings made in these studies.

Secondly, we propose Bert, a source-initiated multicast routing scheme for DCs. 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 state-of-the-art schemes.

Thirdly, we propose, Ernie, a scalable and load-balanced multicast source routing scheme. Ernie introduces a novel method for scaling out the number of supported multicast groups. In particular, it appropriately constructs and organizes multicast header information inside packets in a manner that allows core/root switches to only forward down the needed information. Ernie also introduces an effective multicast traffic load balancing technique across downstream links. Specifically, it prudently assigns multicast groups to core switches to ensure the evenness of load distribution across the downstream links.

Source-Routed Multicast Schemes for Large-Scale Cloud Data Center Networks

by

Jarallah Alqahtani

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented November 30, 2021
Commencement June 2022

Doctor of Philosophy dissertation of Jarallah Alqahtani presented on November 30, 2021.

APPROVED:

Major Professor, representing Computer Science

Head of the School of Electrical Engineering and Computer Science

Dean of the Graduate School

Jarallah Alqahtani, Author

ACKNOWLEDGEMENTS

First and foremost, I would like to show my gratefulness and thankfulness to “Allah” for giving me the strength to fulfill my goals and has been always beside me during my easy and hard times. Next, I would like to sincerely express my special appreciation and gratitude to my outstanding advisor, Prof. Bechir Hamdaoui, for his continuous guidance, encouragement, trust, patience, and continuous follow-up. Through his experience and guidance, I understood the real meaning behind research and the importance of publishing ideas as research papers. It is indispensable to say that his valuable feedback and fruitful discussions were behind all the works in this dissertation. I would like to thank my committee members, Prof. Bella Bose, Prof. Ben Lee, Prof. Lizhong Chen, and Prof. Maggie Niess, for accepting to serve as members of the committee of my defense and for their insightful comments and encouragement. I also wish to thank my professors and colleagues here at OSU, especially Dr. Bassem Khalfi, for their tremendous support and fruitful research discussions.

A special thanks to Najran University, Saudi Arabia, for the full paid scholarship.

The thanks cannot express my gratitude to my parents and my siblings for their unconditional love and support.

Last, but definitely most, I am very thankful to my wife Najwa for her extremely patient, love, forgiving, and constant support. To my precious kids, Ali, Sarah, and Dalal, who have made all this hard work worth the time and effort.

CONTRIBUTION OF AUTHORS

Sultan Alanazi helped with the idea and the writing of manuscript 2. Dr. Hassan Sinky helped with the writing and discussion in manuscript 3. Manuscript 4 in this dissertation was written with valuable feedback from Dr. Rami Langar.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF TABLES

LIST OF ALGORITHMS

Chapter 1: Introduction

Data Centers (DCs) are the nerve center of modern enterprises in today's digital world. DCs consist of tens of thousands of servers and switches, all connected with high-speed communication links [101, 45]. Their use has been growing tremendously in recent years, which has further been intensified by emerging cloud-based services [14, 63, 39, 15] such as data storage, web services, and social media applications and edge cloud offloading paradigms [71, 121, 70, 38, 12, 72] that have significantly increased not only the size and number of DCs but also the data traffic that these DCs have to carry.

1.1 DC Network Topologies

DC network topologies play a significant role in the performances that DCs can achieve in terms of scalability [28, 27, 18, 29], reliability [104, 35], energy consumption [49, 46, 21, 44, 120, 50, 48], latency [120, 94], and traffic load balancing [23, 120]. As a result, in the past few years, there has been a tremendous research focus on developing new DC topology designs with the aim of improving the DC network performance in terms of cost, power consumption, and traffic and resource management [18, 47, 66, 22, 67, 105, 108, 64, 120, 59, 22, 117, 42]. Generally speaking, as shown in Fig. 1.1, these proposed topologies can be categorized based on device function type, device technique type, or architecture type. In the device function taxonomy, DC network topologies can be viewed as either switch-centric (e.g. Fat-Tree, Portland [105], and VL2 [64]), or server-centric (e.g. BCube [66] and DCell [67]). In switch-centric topologies, the forwarding

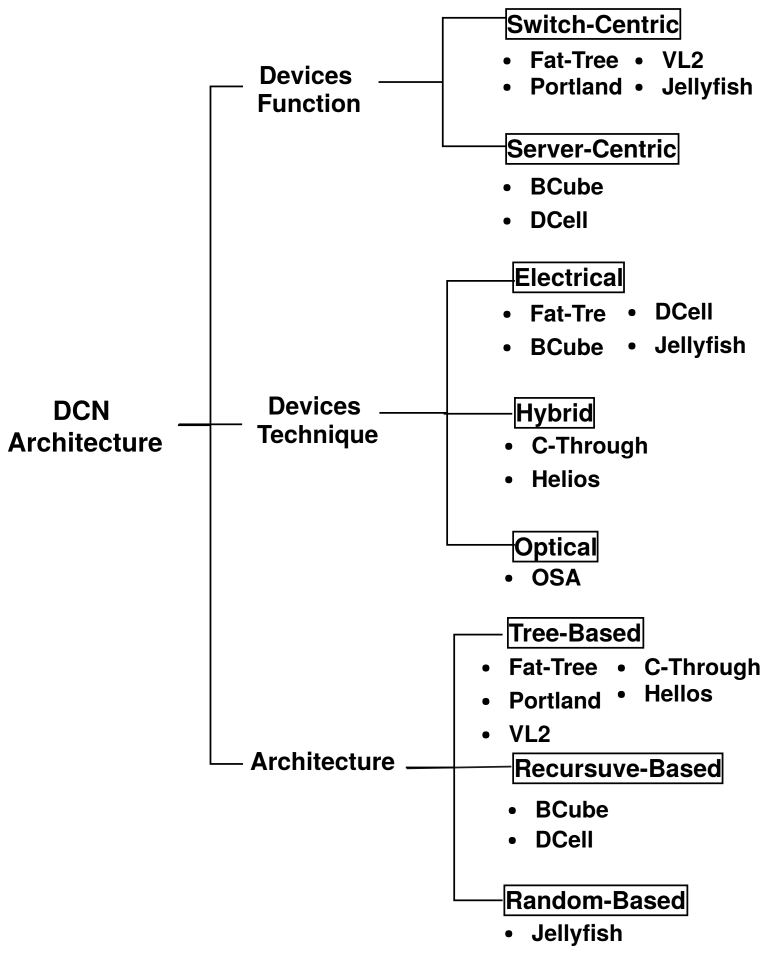


Figure 1.1: Taxonomy of existing DC network topologies.

function is enabled and implemented by switches only, whereas, in the server-centric topologies, servers are also enabled with such a forwarding capability and do play a role in data routing decisions. DC network topologies can also be categorized according to devices technique; that is, optical (e.g. OSA (Optical Switching Architecture) [37]), hybrid (e.g. C-Through [120] and Helios [59]), or electrical (e.g. Fat-Tree, BCube, DCell, and Jellyfish [117]). DC network topologies can also be categorized based their architectures and can be viewed as tree-based architecture (e.g. Fat-Tree, Portland [105], and VL2 [64]), recursive-based architecture (e.g. BCube [66] and DCell [67]), or random-based architecture (e.g. Jellyfish).

1.1.1 Fat-Tree Topologies

Fat-Tree as a Dc network topology was first introduced by Al-Fares et. al [18] as an improvement to the tree topology. The main characteristic of Fat-Tree topology is that the number of links connecting a switch to its lower-layer switches is equal to the number of links connecting the switch to its parent switch. This feature helps in alleviating traffic congestion at the root level by considering multiple core switches as opposed to only one.

As shown in Fig. 1.2, there are three types of Fat-Tree switches, and depending on which layer the switch is placed in, a switch is either called a core, aggregation, or edge/access switch. A Fat-tree topology has k pods, with each pod having $k/2$ edge and $k/2$ aggregation switches. Also, each pod has a link to each core switch via aggregation switches. Each switch in the edge layer is connected to $k/2$ servers. For illustration, we consider $k = 4$ pods in Fig. 1.2 with 4 switches in each pod, two of which are aggregation switches and two are edge switches. Likewise, each aggregation switch has two ports connecting it to the edge switches and to the core switches. In general, a

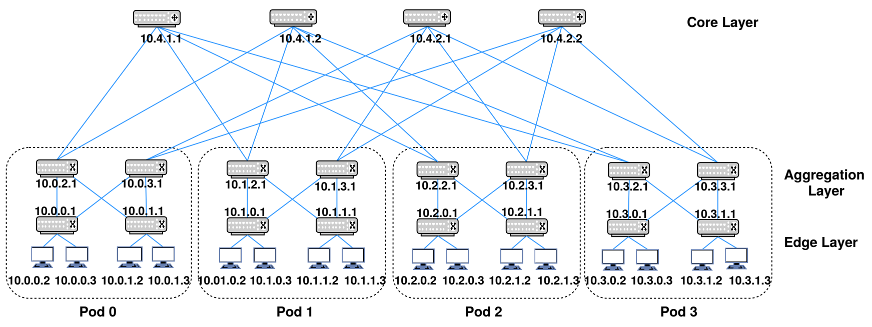


Figure 1.2: Fat-Tree topology with $k = 4$.

Fat-Tree with k pod has $k^3/4$ servers.

1.2 Cloud DC Networks

Cutting edge cloud DC networks consist of massive numbers of servers enabling multi-tenant occupancy, network virtualization and programmable switches [118, 33, 119]. Next, we discuss state-of-the-art components a with modern cloud DCs solutions.

1.2.1 Multi-Tenant DCs

In Multi-tenant DCs (like Microsoft Azure [8], Amazon Web Services (AWS) [1], and Google Cloud Platform [6]), a fraction of computing resources (e.g., CPU, memory, storage, and network) are rented to customers/tenants (e.g., commercial or government organization, or an individual) by means of virtualization technology [51, 88, 17]. These multi-tenant DCs need to guarantee resource isolation and sharing of bandwidth among different tenants [53, 14]. 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 [123, 16]. For example, Netflix, the world's leading online video streaming service provider, uses AWS for nearly all its computing and storage needs [9].

1.2.2 Virtualization in DCs

In multi-tenant DCs, computing and network resources are virtualized [13, 15, 43] typically done by using software or firmware called a hypervisor [11]. The hypervisor allows one host computer to support multiple guest VMs through virtual resource sharing, such as memory and processing. A virtual switch in the hypervisor, called the vswitch [110], 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 [99]) to guarantee resource isolation and fair share of network resources among tenants.

1.2.3 Programmable Switches

Emerging programmable switch ASICs (e.g., Barefoot Tofino [7]) 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 [34]. P4, a language for Programming Protocol Independent Packet Processors, is a recent innovation providing an abstract model suitable for programming the network data plane.

1.3 DC Multicast Routing

Today's cloud DCs host hundreds of thousands of tenants [2, 45], with each tenant possibly running hundreds of workloads supported through thousands of virtual machines (VMs) running on different servers [8, 6, 1]. Furthermore, cloud DCs support a variety of applications that result in distinct network traffic behaviors. These applications are ranging from latency-sensitive services like Web search, to throughput-sensitive tasks like database update. One pattern of traffic in cloud DCs is that most of these applications commonly disseminate data from a single source to a group of receivers for service deployment, data replication, software upgrade, etc. These types of communication pattern naturally suggest the use of multicast because it reduces network traffic and improves application throughput.

1.3.1 Limitations of Current DC Multicast

- **IP Multicast.** Traditional IP multicast can greatly conserve network bandwidth and reduce server overhead of many DC applications. 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 [4]). Furthermore, today's DC networks operate under a single administrative domain and no longer require decentralized protocols like PIM [60] and IGMP [75]. In terms of load balancing, IP multicast-based protocols like PIM are principally designed for arbitrary network topologies and do not utilize topological structure properties of modern DC networks to improve load balancing.

For instance, such protocols usually choose a random single core switch on a Fat Tree as the rendezvous point (RP) to build the multicast tree. With multiple groups using the same core switch simultaneously, traffic bursts and network congestion are likely to occur.

- **SDN-based Multicast.** SDN-based approaches [93, 40], on the other hand, have strived to handle scalability need of IP multicast, 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. Moreover, even though, SDN-based Multicast can achieve better load balancing, the cost of this achievement is quite high. For instance, SDN-based multicast solutions [93, 89, 40, 69, 107] suffer from high computation complexity when maintaining real-time congestion of all network links to balance the multicast traffic.

- **Source Routing Multicast.** In multicast source-routing, the entire path information (multicast tree) is encoded into the header of each packet, and switches can read this information to make forwarding decisions. Under our investigation in this dissertation, source routing technique appears to meet the requirements of today's multi-tenant cloud data centers. Source routing mechanisms minimize the size of routing tables, and reduce the overload in the control plane when compared with other approaches. This dissertation is not the first to suggest multicast source routing techniques for the DC [114, 83, 80, 95, 90, 82].

  Although these approaches have been shown to scale well with the number of multicast groups, it gives rise to other major challenges. For example, some proposals [82, 90] use Bloom filters to encode link identifiers inside packets. 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 [91]). Other source routing approaches like [80, 95] use division (modulo) operation 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 scheme, named Elmo [114], addresses both control and data planes scalability limitations by exploiting DC topology symmetry as well as hardware switch reconfigurability. Although, Elmo is shown to scale well with millions of multicast groups, it still present some major issues. For instance, Elmo 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). Furthermore, source-routed multicast schemes either neglected the multicast traffic load balancing or relied on underlying multipathing protocols (e.g., ECMP[76]). In fact, these load balancing protocols are mainly designed for unicast traffic and are not suitable for multicast.

1.4 Dissertation Contributions

This dissertation proposes three innovations that improve performances and address issues that arise when scaling cloud DC networks. Specifically, it makes the following contributions:

- Proposes Circulant Fat-Tree topology, an improvement over the traditional Fat-Tree topology with better properties to suit nowadays data center networks. We show that our proposed topology alleviates traffic congestion at the core switches, improves network latency, and increases robustness against switch failures when compared to the traditional Fat-Tree topology.

- Proposes Bert, a source-initiated multicast routing for DCs. 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 state-of-the-art schemes.
- Proposes Ernie, a scalable and load-balanced multicast source routing for cloud DCs. Ernie is a novel method for scaling out the number of supported multicast groups. In particular, it appropriately constructs and organizes multicast header information inside packets in a manner that allows core/root switches to only forward down the needed information. Ernie also introduces an effective multicast technique that load-balances traffic in the downstream link path. Specifically, it prudently assigns multicast groups to core switches to ensure the evenness of load distribution across the downstream links. 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

Rethinking Fat-Tree Topology Design for Cloud Data Centers

Jarallah Alqahtani, Bechir Hamdaoui

In Proceedings of IEEE International Global Communications Conference, Dec. 2018

Chapter 2: Manuscript 1: Rethinking Fat-Tree Topology Design for Cloud Data Centers

Abstract:

Data center network (DCN) topologies have recently been the focus of many researchers due to their vital role in achieving high DCN performances in terms of scalability, power consumption, throughput, and traffic load balancing. This chapter presents a comprehensive comparison between two most commonly used DCN topologies, Fat-Tree and BCube, with a focus on structure, addressing and routing, and proposes a new DCN topology that is better suited for nowadays data center networks. We show that our proposed topology, termed Circulant Fat-Tree, alleviates traffic congestion at the core switches, improves network latency, and increases robustness against switch and server failures when compared to traditional Fat-Tree DCN topologies.

2.1 Introduction

Data centers nowadays consist of tens of thousands of servers and switches, all connected with high-speed communication links. The network topology design choice of these data centers is vital to ensuring scalability [18] and to improving data throughput [84] and power consumption [111]. As shown in Fig. 2.1, traditionally, data center network (DCN) topologies are built hierarchically, and are composed of core, aggregation and access (aka edge) layers [74]. Recent years have witnessed an exponential growth in DCN traffic, with

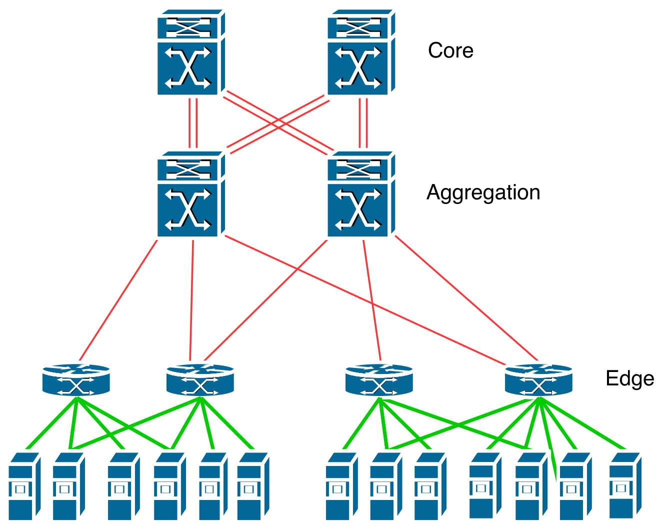


Figure 2.1: Hierarchical Structure of DCNs.

a global data traffic projected to reach 20.6 ZB by 2021, about a three-fold increase from 2016 [102]. In addition, measurement-based studies [31, 32] show that the average traffic loads on core-layer switches are much higher than those on aggregation- and access-layer switches. With the booming of social media, DCNs have been growing even more rapidly, both in size and in numbers, making them more susceptible to device and link failures, which in turn results in frequent data routing failures and flow disruption.

In this chapter, we study existing DCN topologies, and propose Circulant Fat-Treetopology, an improvement over the traditional Fat-Tree topology to better suit nowadays data center networks. Our proposed Circulant Fat-Treeoutperforms traditional Fat-Tree by:

1. Alleviating traffic congestion at the core switches by balancing traffic loads across the different network switches.

2. Improving network latency by reducing the average path lengths between commu-

nicating servers.

3. Increasing robustness against network failures by providing more possible paths between servers.

The reminder of this chapter is organized as follows. In Section 2.2, we provide a taxonomy of various typologies that have been proposed in the literature, and detail the two commonly used DCN topologies, Fat-Tree and BCube. We introduce and present our proposed Circulant Fat-Treetopology in Section 2.3 and evaluate its performance in Section 2.4. We conclude the chapter in Section 2.5.

2.2 DCN Topologies

We compare the two commonly used network topologies, Fat-Tree and BCube, by highlighting their pros and cons vis-a-vis of their structure, data addressing, and routing capability.

2.2.1 Fat-Tree

Fat-Tree as a DCN topology was first introduced by Al-Fares et. al [18] as an improvement to the tree topology. The main characteristic of Fat-Tree topology is that the number of links connecting a switch to its lower-layer switches is equal to the number of links connecting the switch to its parent switch. This feature helps in alleviating traffic congestion at the root level by considering multiple core switches as opposed to only one.

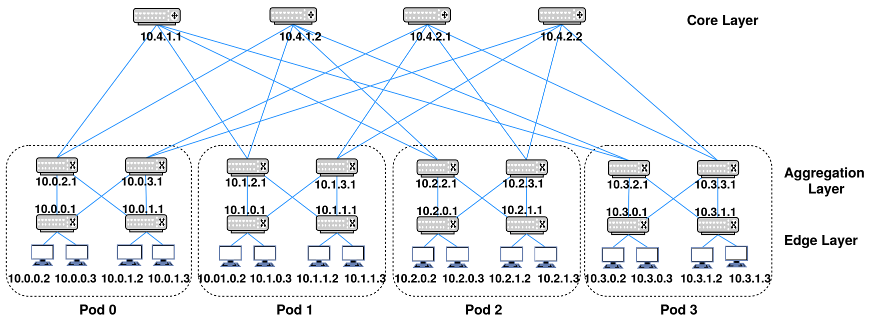


Figure 2.2: Fat-Tree topology with $k = 4$.

2.2.1.1 Structure

As shown in Fig. 2.2, there are three types of Fat-Tree switches, and depending on which layer the switch is placed in, a switch is either called a core, aggregation, or edge/access switch. A Fat-tree topology has k pods, with each pod having $k/2$ edge and $k/2$ aggregation switches. Also, each pod has a link to each core switch via aggregation switches. Each switch in the edge layer is connected to $k/2$ servers. For illustration, we consider $k = 4$ pods in Fig. 2.2 with 4 switches in each pod, two of which are aggregation switches and two are edge switches. Likewise, each aggregation switch has two ports connecting it to the edge switches and to the core switches. In general, a Fat-Tree with k pod has $k^3/4$ servers.

2.2.1.2 Addressing

Fat-Tree topology has special IP addressing. Switches in the edge and aggregation layers are assigned IP addresses of the form 10.pod.$switch$.1, where $pod = 0, 1, \ldots, k-1$ denotes the pod number, and $switch = 0, 1, \ldots, k-1$ denotes the position of the switch in the pod, starting from left to right and bottom to top. Core switches are assigned addresses of the form $10.k.j.i$, where $j, i = 1, 2, ..., k/2$ denote the switch's coordinates, starting from top-left. Finally, servers are assigned addresses of the form $10.pod.switch.ID$, where $ID = 2, 3, ..., k/2 + 1$ denotes the position of the server in the subnet, starting from left to right. Fig. 2.2 shows how addresses are assigned to the Fat-Tree when $k = 4$.

2.2.1.3 Routing

Routing is performed into two steps, first based on primary prefix and then based on secondary suffix lookup. For each incoming packet, destination address prefix is checked in primary table first, and if longest prefix is matched, which means packets are being routed down to the servers, then the packet is forwarded to the specified port. Otherwise, the secondary level table is checked and the port entry with the longest suffix match is used to forward the packet, which means packets are being routed up towards core switches. At the core switches, packets are simply directed to the pod in which the destination is located.

2.2.2 BCube

BCube is a server-centric topology introduced mainly to support *modular data centers* [66]. One of its other main purposes is also to support all the traffic patterns (one-to-one, one-to-several, one-to-all, and all-to-all).

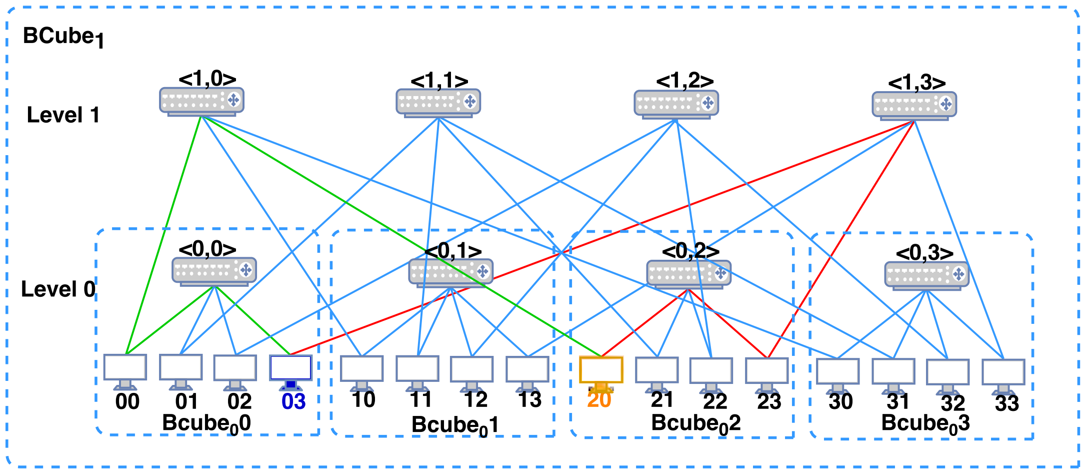


Figure 2.3: Construction of $BCube_1$ from $BCube_0$ with $n = 4$. Source is (03) and destination is (20). Using hamming distance yields two parallel paths: (03) $\rightarrow$ (00) $\rightarrow$ (20) or (03) $\rightarrow$ (23) $\rightarrow$ (20).

2.2.2.1 Structure

BCube is a recursive structure, with $BCube_0$ is constructed by n servers that are connected to an n-port switch, $BCube_1$ is constructed from n $BCube_0$ that are connected to n switches, and so on. In general, $BCube_k$ is constructed from n $BCube_{k-1}$ that are connected to n^k n-port switches. Each server in $BCube_k$ has $k+1$ ports, numbered from level-0 to level-k. Thus, $BCube_k$ has n^{k+1} servers and n^k switches. Fig. 2.3 shows the BCube structure with $n = 4$ and $k = 1$.

2.2.2.2 Addressing

Servers in $BCube_k$ are assigned addresses $a_k a_{k-1} a_0$ with $a_i \in \{0, 1, ..., n-1\}$. Switches, on the other hand, are assigned addresses in the form $< l, s_{k-1} s_{k-2} s_0 >$ where $l =$

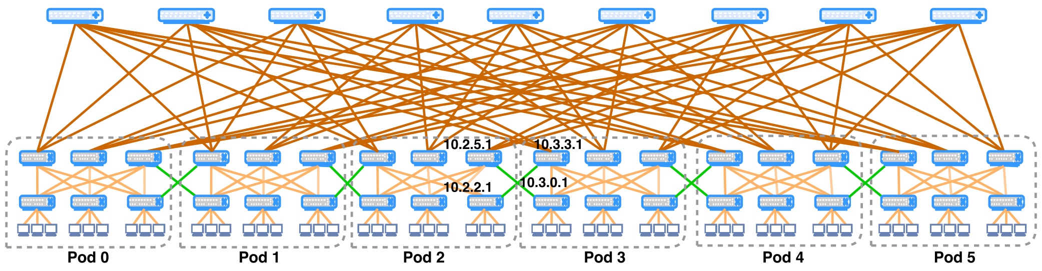


Figure 2.4: The proposed Circulant Fat-Tree with $k = 6$.

$\{0, 1, ..., k\}$ is the level of the switch. The i^{th} server in the j^{th} $BCube_0$ connects to the j^{th} port of the i^{th} level-1 switch. For example, server 12 is connected to switch $< 0, 1 >$ in level 0 and to switch $< 1, 2 >$ in level 1. Fig. 2.3 shows how addresses are assigned to BCube when $n = 4$ and $k = 1$.

2.2.2.3 Routing

BCube uses the Hamming distance technique, $h(A, B)$, to denote the distance between two servers, A and B, which corresponds to the number of different digits within their address arrays. When a packet is forwarded from source to destination, one digit is changed in each step. Thus, the path length is at most $k + 1$. There are $k + 1$ edge-disjoint paths between any two servers in a $BCube_k$ topology. These paths can be utilized for achieving high bandwidth in order to improve one-to-one, one-to-many, and all-to-all traffic performances. Fig. 2.3 shows an example of routing in $BCube_1$ when n=4.

Table 2.1 presents a summary of the Fat-Tree and BCube features that we presented.

Table 2.1: Features of Fat-Tree and BCube topologies: N=number of servers, n=number of n-port switches, k= number of BCube levels

	Structure						**Routing**			
	Nbr of Servers	**Nbr of Switches**	**Nbr of Wires**	**Connecting Rule**	**Server Degree**	**Diameter**	**Decider**	**Type**	**Protocol**	**Node-Disjoint Paths**
Fat-Tree	$n^3/4$	$5N/n$	$3N$	A switch can connect to other switches. A server has only one direct link to another switch	1	6	Switches only	Two-level lookup table	Equal-cost multi-path routing (ECMP)	1
BCube	n^{k+1}	$(Nlog_n N)/n$	$N(k+1)$	Switch-to-switch connections are not allowed. A server has k+1 direct links to k+1 other switches.	$k+1$	$2(k+1)$	Servers and Switches	Hamming distance	BCube Source Routing (BSR)	$k+1$

2.3 The Proposed Topology Design: Circulant Fat-Tree

To overcome these aforementioned challenges, we propose an improved Fat-Tree topology, termed Circulant Fat-Tree in this chapter. Circulant Fat-Tree improves DCN performances by reducing data traffic traversing congested links, thereby reducing network latency. In addition, Circulant Fat-Tree improves the robustness of DCNs against node and link failures by increasing the number of possible paths among server pairs.

2.3.1 The Physical Structure

Similarly to Fat-Tree, the switches in Circulant Fat-Tree are also categorized into core, aggregation and edge switches. Edge and aggregation switches are arranged into k pods, where every edge switch connects $k/2$ servers. The total number of servers and switches are $k^3/4$ and $5 * k^2/4$ respectively. In Circulant Fat-Tree, every pod is connected to the adjacent pod through its aggregation and edge switches situated at the pod's border. Formally, if pod switches are addressed in the form $10.pod.switch.1$, then the connecting rule between adjacent pods are as follows. For every pod, $10.pod.k/2 - 1.1$ switch is connected to $10.pod + 1.k/2.1$ and $10.pod.k - 1.1$ is connected to $10.pod + 1.0.1$. Hence, the number of wires in Circulant Fat-Tree is increased by $2(k - 1)$ when compared to Fat-Tree. However, the increase in number of wires is negligible especially when the number of pods is large. For example, when $k = 24$ this increase is only 0.4%. Figure 2.4 shows an example of Circulant Fat-Tree where $k = 6$.

2.3.2 Key Features of Circulant Fat-Tree

Circulant Fat-Tree outperforms traditional Fat-Tree in each of the following performance metrics.

2.3.2.1 Alleviating Traffic Congestion at Core Switches

In Fat-Tree DCNs, the average link loads at the core switches are higher than those at aggregation and edge switches. For instance, any two servers belonging to two different pods can only communicate through core switches. However, unlike Fat Tree, in Circulant Fat-Tree, any two servers belonging to any two consecutive pods can communicate directly through aggregation switches without needing to go through core switches. For fair comparison, throughout, we only consider paths whose lengths (number of hops) do not exceed 6, since this is the maximum path length of Fat-Tree topologies.

2.3.2.2 Reducing Average Path Length (APL)

APL is a key performance metric for evaluating DCN topologies, as it captures end-to-end latency of traffic between pairs of servers. The proposed Circulant Fat-Tree reduces APL of $(k^3/4 - k/2)$ pairs of servers[1] among a total of $(k^4/16)$ pair of servers from 6 (as in the case of Fat-Tree) to only 4. When $k = 24$, this, for example, corresponds to reducing APL of about 16% of the total pairs from 6 hops to 4 only. More on this will be provided in the next section.

[1]We only consider pairs whose servers belong to different pods.

Table 2.2: APL Comparison

	APL: adjacent pods only		**APL: all pods**	
No. of Servers	Fat-Tree	`Circulant Fat-Tree`	Fat-Tree	`Circulant Fat-Tree`
128	6	5.125	5.688	5.496
1024	6	5.469	5.859	5.804
3456	6	5.681	5.911	5.884

2.3.2.3 Reducing Network Latency

Most DCN routing algorithms use the shortest path metric for making routing decisions. Since Circulant Fat-Tree reduces the APL among communicating servers, then the average network latency achieved under Circulant Fat-Tree is reduced when compared to Fat-Tree, especially for communicating servers belonging to adjacent pods; this is shown in Table 2.2.

2.3.2.4 Improving Robustness Against Node/Link Failure

Circulant Fat-Tree is more tolerant to switch and server failures than Fat-Tree topologies. Robustness to network failures is improved because Circulant Fat-Tree increases the number of possible paths between pairs of servers belonging to consecutive pods when compared to Fat-Tree.

2.4 Performance Evaluation

In this section, we evaluate and compare our proposed Circulant Fat-Tree topology to Fat-Tree topology in terms of the metrics presented in Section 2.3.2. To ensure a fair

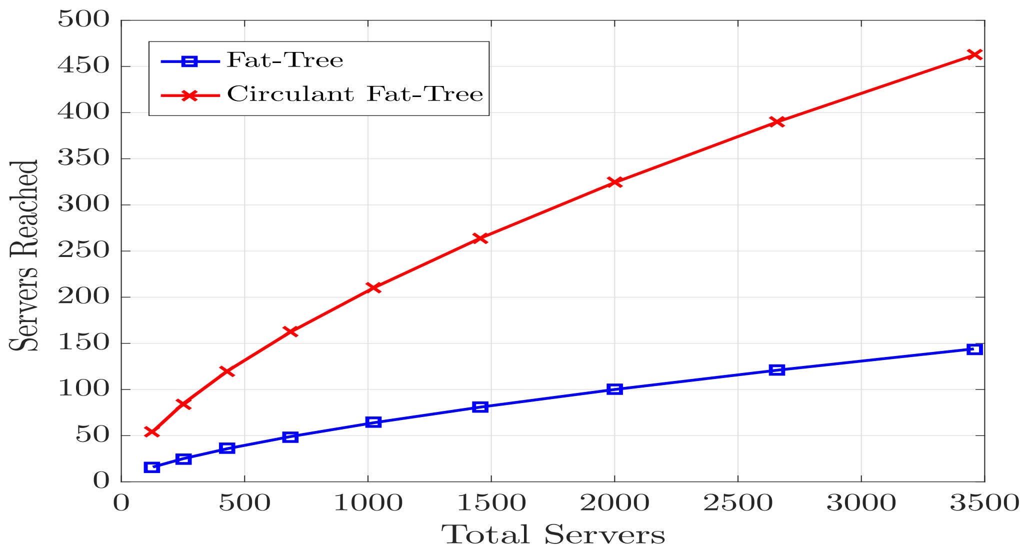


Figure 2.5: Number of servers that can reach other servers without going through core-level switches with path length of at most 6.

comparison, we only consider path lengths of at most 6 hops. Recall that Circulant Fat-Tree increases the number of servers that can be reached from another server via pod-level by at least 200% when compared to Fat-Tree. For example, when referring to Fig. 2.4, a server connected to switch 10.3.0.1 in pod 3 can communicates with 38 other servers without needing to go through core-level switches. On the other hand, this same server can only communicate with 8 servers when Fat-Tree topology is used for the same scenario. Fig. 2.5 shows the number of servers that can reach each other through pod-level switches instead of core-level switches for both topologies, Fat-Tree and Circulant Fat-Tree, with path length of at most 6 for different total numbers of servers.

Without loss of generality, to measure the network traffic, we suppose that each server sends one traffic unit (e.g., one packet) to every other server in the network. This scenario is called *all-to-all* traffic. We also assume that traffic is routed along shortest

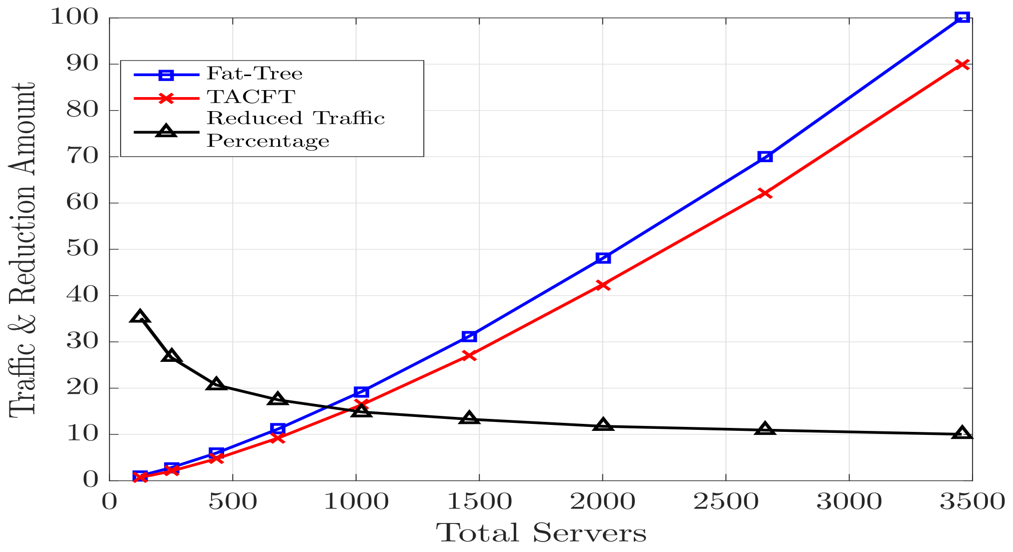


Figure 2.6: All-to-All traffic between two random pods.

paths, where here the number of hops is used to measure the path length. To ensure a fair comparison, we assume that the maximum shortest distance (in number of hops) among all the server pairs is at most 6 for both topologies. Fig. 2.6 shows the total amount of traffic that crosses core-level switches when all server pairs (each belonging to a different pod) are communicating. In this experiment, the number of servers is varied from 128 to 3456. Observe that Circulant Fat-Tree reduces the core-level traffic when compared to Fat-Tree, especially when the number of servers is low. This reduction varies from 35% to 10% when the number of servers goes from 128 to 3456.

Fig. 2.7 shows that Circulant Fat-Tree reduces APL between any two adjacent pods when compared to Fat-Tree. APL achieved under Circulant Fat-Tree varies from 5.125 to 5.680 as opposed to 6 in the case of Fat-Tree.

Fig. 2.8 shows the total amount of traffic that crosses core-level switches under all-to-all traffic scenario when considering server pairs in the entire network. The figure

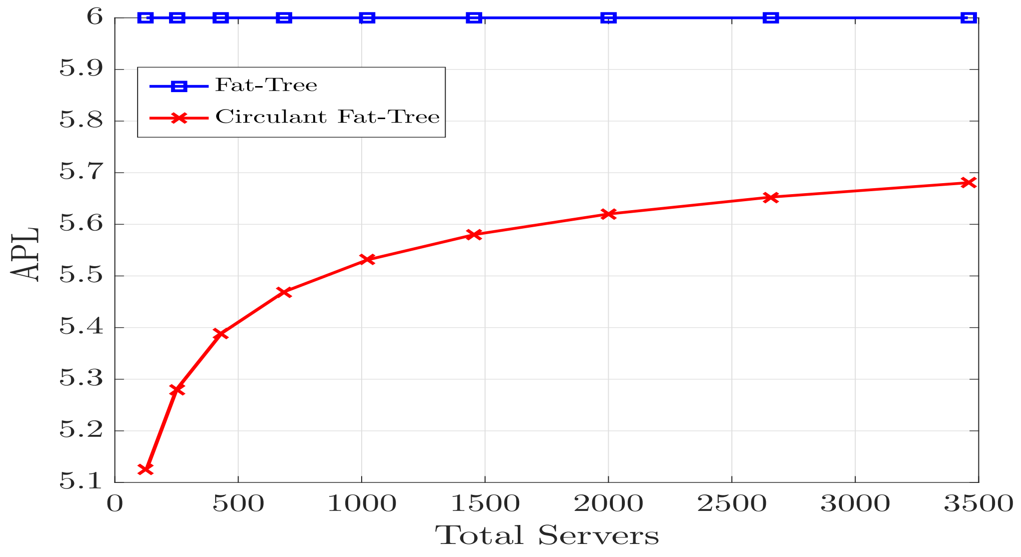


Figure 2.7: APL between any two consecutive pods.

shows that lesser traffic goes through core-level switches under Circulant Fat-Tree than under Fat-Tree, with a traffic reduction that varies from 34% to 10% when the number of servers varies from 128 to 3456. Therefore, Circulant Fat-Tree reduces the congestion at core-level switches by balancing traffic among all switches: core, aggregation and edge. This is one of key features of Circulant Fat-Tree.

Now when considering all servers communicating to all servers regardless of their pod, APL achieved under Circulant Fat-Tree is also smaller than that achieved under Fat-Tree. This is illustrated in Fig. 2.9.

In Circulant Fat-Tree, the multiple redundant paths that are introduced between any two adjacent pods result in alleviating core-level traffic congestion, reducing network latency, and improving robustness to switch and server failures. We only consider paths of length ≤ 6 to ensure fairness with the Fat-Tree. Fig. 2.10 shows Circulant Fat-Tree results in more possible routing paths between any two adjacent pods than Fat-Tree. For

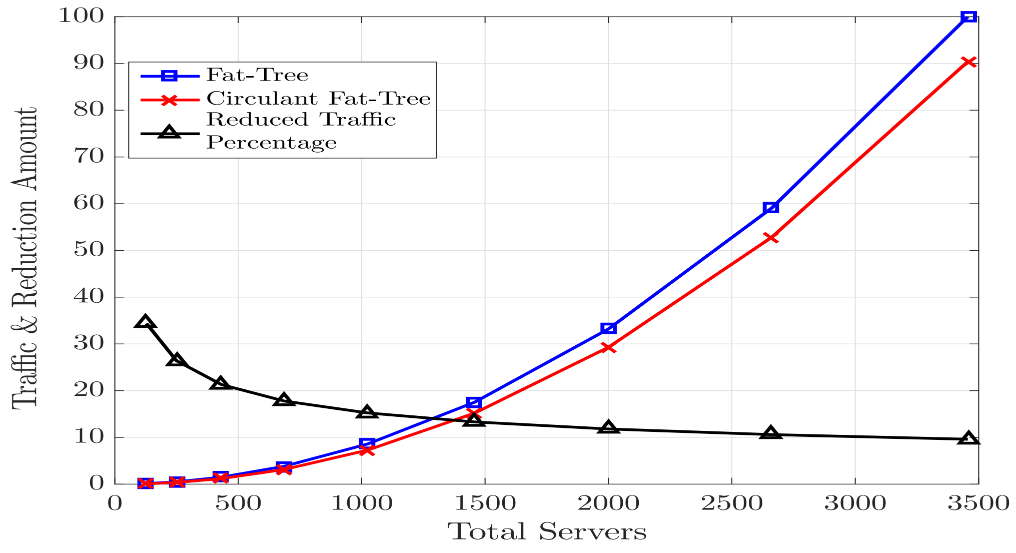


Figure 2.8: All-to-All traffic in entire network.

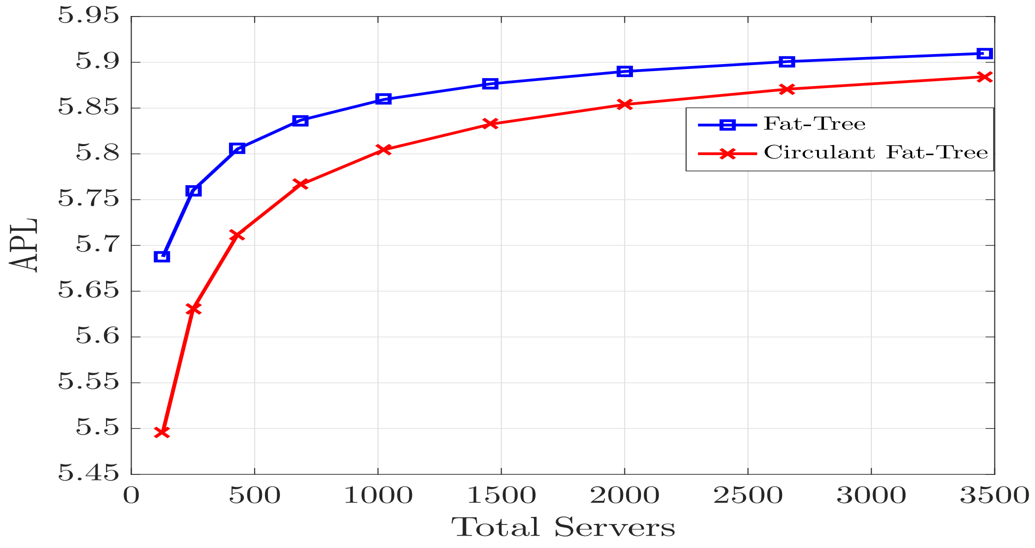


Figure 2.9: APL for the entire network.

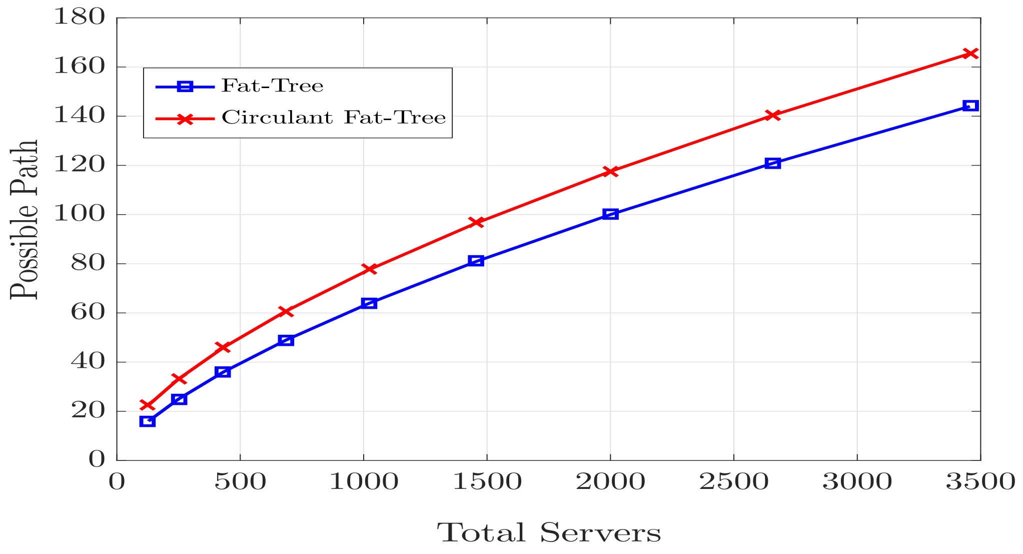


Figure 2.10: Average number of possible paths between any two servers in two consecutive pods.

example, in Circulant Fat-Tree, when the total number of servers is 1024, any two servers in two consecutive pods have 78 possible routing paths with length ≤ 6. Fat-Tree, on the other hand, has only 64 possible routing paths with the same length.

2.5 Conclusion

We propose the Circulant Fat-Tree topology, an improvement over the traditional Fat-Tree topology to better suit nowadays data center networks. The proposed Circulant Fat-Tree topology alleviates traffic congestions at core-level switches by providing a better balance of traffic loads across the different network switches, reduces latency of data communicated across servers by reducing the average path lengths among communicating servers, and augments robustness against network failures by increasing the number of possible paths between server pairs.

Traffic Behavior in Cloud Data Centers: A Survey

Jarallah Alqahtani, Sultan Alanazi, Bechir Hamdaoui

in Proceedings of International Wireless Communications and Mobile Computing Conference (IWCMC), June 2020

Chapter 3: Manuscript 2: Traffic Behavior in Cloud Data Centers: A Survey

Abstract

Data centers (DCs) nowadays house tens of thousands of servers and switches, interconnected by high-speed communication links. With the rapid growth of cloud DCs, in both size and number, tremendous efforts have been undertaken to efficiently design the network and manage the traffic within these DCs. However, little effort has been made toward measuring, understanding and chattelizing how the network-level traffic of these DCs behave. In this chapter, we aim to present a systematic taxonomy and survey of these DC studies. Specifically, our survey first decomposes DC network traffic behavior into two main stages, namely (1) data collection methodologies and (2) research findings, and then classifies and discusses the recent research studies in each stage. Finally, the survey highlights few research challenges related to DC network traffic that require further research investigation.

3.1 Introduction

During the last decade, the intense usage of cloud-based services such as data storage, web services, and social media applications have significantly increased the data traffic within data center networks (DCNs). For instance, recent studies [103] project that global data traffic will reach 20.6 ZB by 2021, about a three-fold increase from 2016.

Moreover, 73.3% of this traffic is anticipated to stay within the data center (DC). As a result, significant research attention has recently been devoted to DCNs, ranging from designing new architectures and topologies to coming up with new performance metrics and developing methodologies for assessing and improving them. Despite these research efforts, there are limited studies that investigate and measure the traffic behavior of DCNs through real-world workloads. To preserve the Quality of Service (QoS) that DCNs offer their clients, a deeper measurement and analysis of the traffic behavior of the DCNs are crucial. For example, high packet drops and poor achievable throughputs caused by traffic congestions can lead to poor DCN performances (e.g., incur delay of search queries, increase failure rates of email services, and increase interruption rate of instant messaging).

These existing studies examine traces from real-world workloads in deployed networks. The studied data centers involve cloud, enterprise, and university domains, which differ not only in size and hosted application types, but also in traffic behaviorial aspects, including flow characteristics, presence and locality of burstiness, and traffic patterns. The traffic characteristics of these studies are highly correlated with DCN applications, which are responsible for introducing data into the network, with applications ranging from latency-sensitive ones such as Web search applications to throughput-sensitive ones such MapReduce applications.

In this chapter, we present and discuss these studies in detail. To the best of our knowledge, this chapter provides the first taxonomy and detailed comparison of DCN traffic behavior studies. As shown in Fig 3.1, traffic characteristics of real DC workload studies conducted in literature can be decomposed into two main stages: 1) data collection, and 2) findings. Based on these two stages, we further break them down and classify them based on the features as presented Fig 3.1. The main contribution of this survey is the characterization of real workload DCN traffic studies by highlighting their similarities and differences. As a result, we believe this survey is a valuable resource for both researchers and practitioners seeking to understand DCN traffic behavior.

The remainder of this chapter is organized as follows. Section 3.2 discusses and classifies existing studies based on the data collection method. In Section 3.3, we present the research findings of these studies, and classify the surveyed studies. We highlight some challenges and unsolved issues in DCN traffic behavior in Section 3.4, and conclude the chapter in Section 3.5.

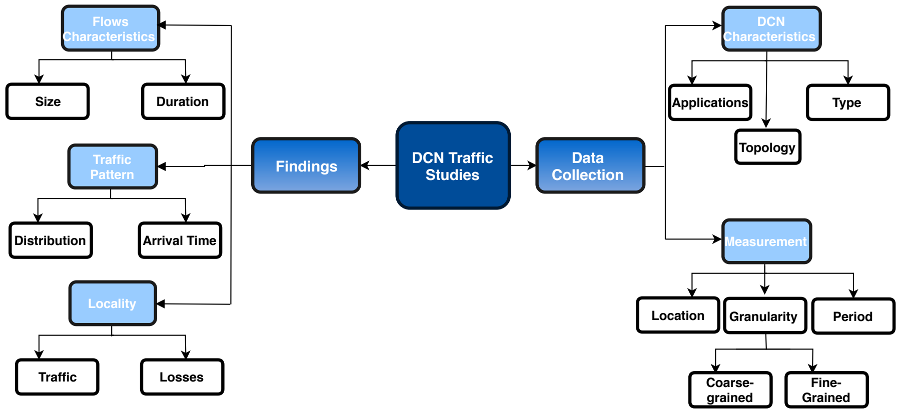


Figure 3.1: DCN Traffic Behavior Taxonomy.

3.2 Data Collection

In this section, we review and classify network behavior studies conducted on different data center networks based on their method of data collection. Furthermore, in Table 3.1, we compare in detail the methodology of collecting data for the surveyed studies in terms of deployed applications, DCN architecture, measurement tools and granularity. Data collection plays a significant role in data center network traffic studies. A good study must present an accurate measurement and useful data that can be used in different type of data centers. The data used in current data center network studies vary based on DC characteristics and traffic measurement.

3.2.1 DC Characteristics

Three main parameters can affect DCN traffic behavior. These parameters, which we discuss next, are DC type, DCN topology, and DC applications.

3.2.1.1 DC type

We classify studies under DC type into cloud and private data centers. Cloud DCs serve a variety of users and provide support for a broad range of internet services such as search, Email, video streaming, and real-time Messaging. Furthermore, cloud DCs have other systems hosted internally such as data mining, database, and storage to support the offered internet services. Some of these DCs are built with certain topology and oversubscription ratio to support specific applications, while other DCs are built for general purpose applications. We want to mention that the majority of traffic behavior studies are conducted on cloud DCs, namely Microsoft [86], Facebook [113], and Google [116], with, to the best of our knowledge, about 50% of the total studies being conducted on Microsoft DCs as shown in Fig 3.2.

On the other hand, private DCs serve a set of specific users and include universities and enterprises. University DCs serve students, faculty members, and administrative staff, and support a wide range of services such as student web portal, Email systems, administrative sites, and distributed file systems. Enterprise DCs, on the other hand, support a wide range of customized enterprise-specific applications along with the main applications like Web and Email services. There is only one study that focuses on private enterprise DC, IBM [101]. In addition, there are two DCN studies that focus on both cloud and private DCs, covering 19 DCs ([32]) and 10 DCS ([31]).

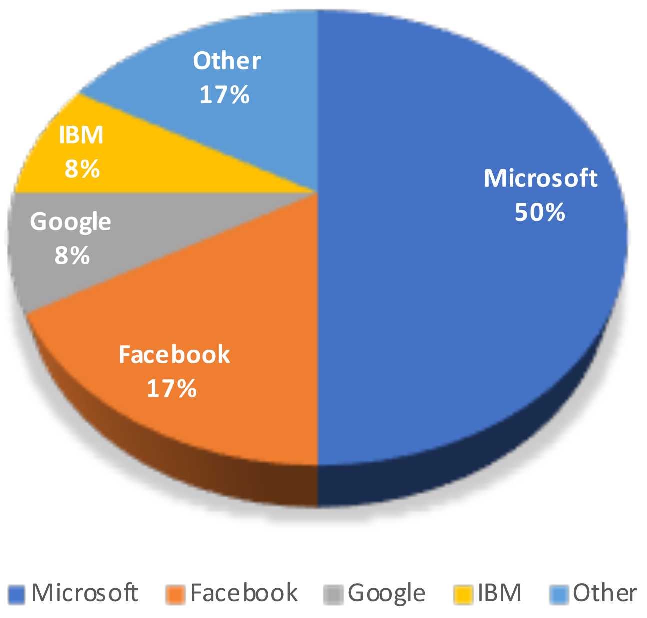


Figure 3.2: Percentage of studies conducted on different operational data centers .

3.2.1.2 DCN topology

The surveyed papers can be classified into two DC topology types: 3-tier and 2-tier. Fig 3.3 shows the architecture of these two types. The 3 tiers of the 3-tier DC topology are the edge tier, which consists of the Top-of-Rack (ToR) switches that connect the servers to the DC's network fabric; the aggregation tier, which consists of devices that interconnect the ToR switches in the edge layer; and the core tier, which consists of devices that connect the DC to the WAN. Examples of studies conducted on 3-tier DCs include [86, 113, 124, 79].

In smaller DCs, the core tier and the aggregation tier are collapsed into one tier, resulting in a 2-tier DC topology. In current network behavior studies, 3-tier topology dominates cloud DC architectures while 2-tier is most common in private DCs [32, 31].

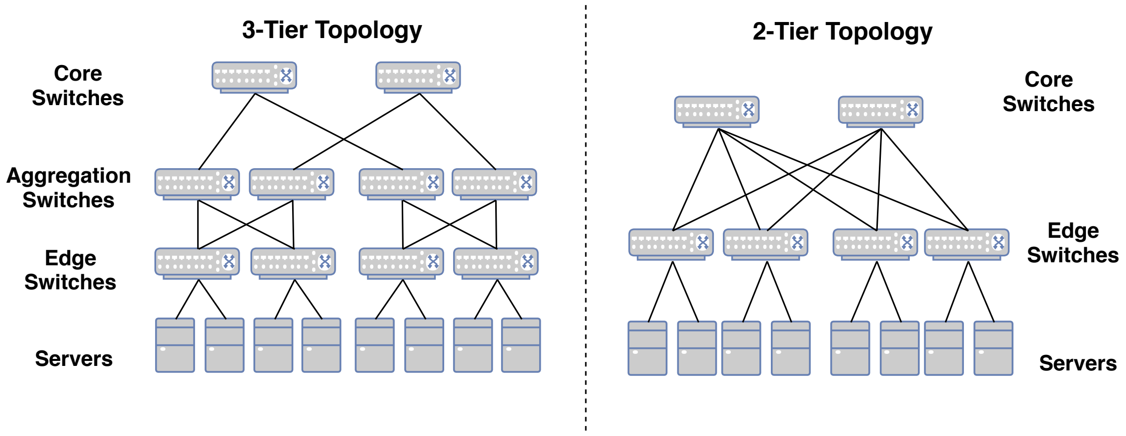


Figure 3.3: 2-tier vs 3-tier Architecture.

3.2.1.3 DC applications

DCs support a variety of applications that result in distinct network traffic behaviors. We classify existing works based on DCN application into data mining, web services, and caches. MapReduce and Hadoop are the most popular applications of data mining. MapReduce [52] is a programming model designed to handle data parallel applications by dividing the job into a set of independent tasks. It consists of two main functions: Map and Reduce. The Map function as shown in Fig 3.4 generates a set of intermediate key/value pairs from large amount of row data. Then, the intermediate data get shuffled. The Reduce function as shown in Fig 3.4 aggregates similar intermediate values and output the result. This model is widely used for system design. MapReduce is one key application that derives most of the traffic in the studied DC networks [32, 31, 86, 64, 85, 25]. Hadoop [115] is another application used for offline analysis and data mining. It is an efficient implementation of MapReduce, which provides both reliability and data transfer. Facebook researchers [113, 124], for instance, collect traffic generated by Hadoop servers in their DCN traffic studies.

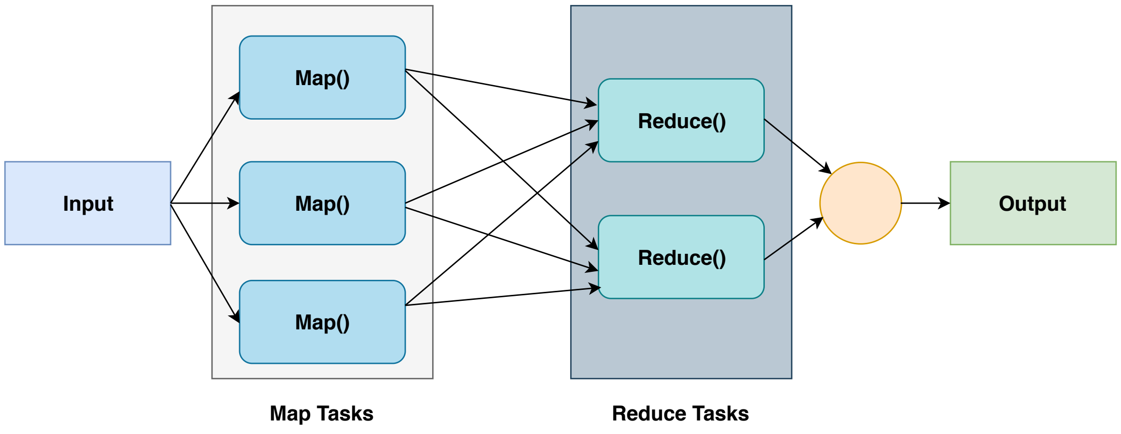


Figure 3.4: MapReduce.

Web service applications, including web requests, Email, and video streaming, are also commonly used in DCs. The majority of traffic analysis papers capture and use traffic generated by web service applications [32, 31, 113, 124, 54, 25, 79]. Cache serves as an in-memory cache of data used by the web servers. Some of these servers are leaders, which handle cache coherency, and some are followers, which serve most read requests. Such services constitute the main applications that generate traffic in Facebook DCs [113, 124].

3.2.2 Measurement

Measurement can be decomposed into granularity and period. We classify studies based on their measurement granularity into coarse and fine-grained measurements. Coarse-grained is per-minute measurement and can be sampled using measurement tools such as Simple Network Managment Protocol (SNMP) [36] and Fbflow [113]. The SNMP data provide link-level statistics that can be used to study coarse-grained characteristics such as link utilization, packet drops, congestion, loss rates, etc. These statistics are collected by default in many DCs to detect major problems in the network. For example, the work in [32] presents an end-to-end traffic analysis by collecting tens of gigabytes of SNMP logs at 19 DCs over ten days to study coarse-grained characteristics of network traffic. Furthermore, authors of [31] collected SNMP link information in 10 DCs of three different organizations including cloud, university, and enterprise DCs for a period of ten days. SNMP traces are used in the studies to show the variation of link properties based on time and location in network-level.

Fbflow, on the other hand, is a monitoring tool that constantly samples packet headers across the whole network. It has two main components: Agents, which parse the headers of nflog samples extracting information including source and destination IP addresses, port numbers, and protocol type, and Taggers, which collect additional information such as the rack and cluster of sampled machines. Fbflow is deployed across the entire network of Facebook DCs [113], that connects hundreds of thousands of 10-Gbps nodes, to monitor data traffic among their major services.

Although these coarse-grained measurements are suitable for network monitoring and management, their sampling rate limits the study of some traffic properties including burstiness, and interarrival times that have an effect on traffic engineering approaches.

On the other hand, fine-grained is per-second measurements, such as port mirroring, packet sampling, and others. Port mirroring records packet-header traces over microsecond intervals. Facebook researchers [113] turn on port mirroring on top of rack switches to capture the full, in-going and out-going traffic for selected servers.

Sampling packet directly using tcpdump is another way of providing fine-grained measurements such as packet inter-arrival times. Packet sampling can give a better understanding of traffic patterns. In [32] a part of their study is conducted on a small number of edge switches to obtain fine-grained traffic information. The data used in the study contains packet traces collected during a roughly two weeks period. Packet traces are collected from selected switches in private enterprise and university DCs that span 12 hours of multiple days [31]. Packet traces allow authors to study different types of applications running in different DCs and the amount of traffic each application contributes to the network traffic. Furthermore, it facilitates examining the sending pattern at both packet and flow levels.

Researchers have proposed hardware modifications to provide more scalable and accurate measurements. However, these are not deployed widely enough to perform large-scale production measurements. Authors in [86] instrument servers to gather socket-level logs with minimal performance overhead. Over two months period, nearly a petabyte of measurements has been collected and analyzed from a 1500 server operational data center to report traffic patterns in terms of server communications and traffic characteristics in terms of flow statistics. This paper chooses a server-centric approach using Event Tracing for Windows (ETW) [5] to collect traffic events. The argument behind server-centric preference is that per-server monitoring incurs minimal overhead compared to network device monitoring. Authors in [124] focus mainly on fine-grained traffic behaviors on rack switches using a high-precision microburst measurement in data center networks.

This paper modifies CPUs in modern switches to pool periodically local counters. Three sets of counters are the main focus in the paper which are Byte count, Packet size, and Peak buffer utilization. The measurement span 10 racks for each application type over one day period.

Obtaining finer grained data requires instrumentation of all switches and links in the cluster network; however, it is infeasible in today's DCs due to their large scale sizes.

Table 3.1: Data Collection Comparison Table

Paper	**Type of DCN**	**# of DCs**	**Topology**	**Applications**	**Duration**	**Measurement Granularity**	**Measurement Tool**	**Year of Study**
[32]	• Cloud • Enterprise	19	2 & 3 tiers	• Web Services • MapReduce	10 days	• Coarse-grained • Limited Fine-grained	• SNMP • Packet traces	2009
[86]	• Cloud (Microsoft)	1	3 tiers	• MapReduce	2 months	• Coarse-grained	• SNMP • Event Tracing for Windows (ETW)	2009
[31]	• 5 Cloud • 2 Enterprise • 3 University	10	2 & 3 tiers	• Web Services • MapReduce	10 days	• Coarse-grained • Limited Fine-grained	• SNMP • Packet traces	2010
[113]	• Cloud (Facebook)	1	3 tiers	• Web request • cache • Hadoop	One Day	• Coarse-grained • Limited Fine-grained	• FbFlow • Port Mirroring	2015
[124]	• Cloud (Facebook)	1	3 tiers	• Web request • cache • Hadoop	One day	• Limited Fine-grained	• CPUs in switches	2017
[64]	• Cloud (Microsoft)	1	3 tiers	• MapReduce		• Coarse-grained	• SNMP	2009
[25]	• Cloud (Microsoft)	1	3 tiers	• Web Services	One month			2011
[116]	• Cloud (Google)	1	2 & 3 tiers	• Web Services • MapReduce				2015
[54]	• Cloud (Microsoft)	1	3 tiers	• Web Services	5 Months	• Limited Fine-grained		2012
[85]	• Cloud (Microsoft)	1	3 tiers	• MapReduce	Few months	• Coarse-grained	• Event Tracing for Windows (ETW)	2009
[101]	• Enterprise (IBM)	1			10 days	• Coarse-grained		2010
[79]	Cloud (Microsoft)	1	3 tiers	• Web Services • MapReduce	Several hours	• Limited Fine-grained	• Packet Traces	2013

3.3 Findings

The existing studies of DCN traffic behavior can be categorized into three basic categories: flow characteristics, traffic pattern, and locality. For flow characteristics, studies show that both mice (small) and elephant (large) flows co-exist and are part of DC traffic. However, their existence varies based on DCN type and application. In addition to flow properties, traffic pattern is another key aspect of the DCN traffic behavior. Traffic patterns include flow arrival rates and distributions as well as traffic matrix. The traffic matrix represents how much traffic is exchanged between pairs of servers in the DC. Finally, locality of traffic, burstiness, and losses are getting much attention in the surveyed studies. Link utilization was measured to specify traffic amount in each layer. Packet drops also were quantitatively and spatially examined. We then explain these observations in detail and classify existing studies in Table 3.2 based on their findings.

3.3.0.1 Flow Characteristics

Flow characteristics including flow size and duration have been studied in some works, where DCN Traffic is frequently measured and characterized according to flows. A flow is a sequence of packets from a source to destination hosts. Flows can be large or small based on their sizes. Large flows are long-lived and throughput-sensitive while small flows are short-lived, latency sensitive, and highly bursty in nature. Many DC providers run a variety of applications that significantly influence the flow size. For example, flows that generated by web search applications are small in size (few KBs) while database update applications generate large flows (> 1MB).

Most of DCN traffic behavior studies conclude that majority of flows are small in size

($\leq$ 10KB) [31, 113, 64, 25]. These studies also exhibit lack of large flows and absence of super large flows. Moreover, most bytes are delivered by large flows [25, 64, 31]. Besides the flow size, flow duration is also observed in some of these studies. According to [86], most of flows duration are short. For example, about 80% of flows last for less than 10 seconds. However, about .1% of flows last longer than 200 seconds. In summary, there exist a mix of bandwidth-sensitive large flows and delay sensitive short flows in DC traffic.

3.3.0.2 Traffic Patterns

Understanding traffic patterns including flow arrival rates and distributions as well as traffic matrix are crucial for well designed DCNs. Traffic matrix analysis allows DCN operators to allocate the necessary resources (bandwidth, VM placement, etc.) to meet the required quality of end-users experience. Unfortunately, it is difficult to build a robust traffic prediction system for DCN. First, because a detailed real traffic information needs to be collected in fine time granularity. Second, DCN currently deploys a wide variety of applications and these applications vary from simple web applications to more complex business workflow applications.

However, several studies of real workload exploited some basic traffic characteristics to describe traffic pattern of cloud DCs. For example, [31, 32] shows that packet arrivals evidence an ON/OFF traffic behavior after packet traces of private enterprise and university DCs. Furthermore, this pattern fits best with log-normal distributions. Another study of Microsoft DCN traffic [64] concludes that total traffic matrix of cloud DCN is unpredictable and fluctuating. In spite of that, a study of Facebook DCN traffic pattern [113] concludes that traffic strongly exhibits continuous arrivals and do not

Table 3.2: Findings Comparison

	Findings	Paper
Fow characteristics	• Most of flows in data centers are small in size (a few KB) .	[31, 113, 64, 25]
	• Lack of large flows and absences for super large flows	[31, 113, 64, 86]
	• Most bytes are delivered by large flows	[25, 64, 31]
	• Most of traffic bursts in the DCN are “microburst”	[124]
Traffic Patterns	• Similarity in network activity patterns over time	[54]
	• ON/OFF behavior of packet arrivals	[32, 31]
	• Weak correlation between traffic rate and latency	[101]
	• Uneven distribution of traffic volumes from VMs	[101]
	• Traffic changes over different time scales	[113, 54, 31, 86]
	• Traffic pattern are unpredictable	[64]
	• Ratio of traffic volume between servers in data centers to traffic entering/leaving data centers 4:1	[64]
	• The inter-arrival time of majority of microbursts are short	[124]
	• Existence of Incast Problem	[25, 86]
Locality of Traffic and Losses	• Packet,drops are highest at the edge	[32, 31, 124]
	• Links utilization in the core level are the highest	[32, 31, 113]
	• Majority of cloud data centers traffic stays within the rack	[31, 86, 54]
	• Small fraction of discards is due to oversubscription of ToR uplinks towards the fabric	[116]
	• ToR switches layer are more bursty	[124]
	• Only a few ToRs are hot and most of their traffic goes to a few other ToRs	[85]
	• Traffic is not localized to any particular layer and it depends upon the service	[113]
	• Traffic locality is stable across time period	[113, 54]

evidence on/off pattern. In addition, it is observed that DCN traffic can be very bursty, and most of this bursty traffic is "microburst". For example, about 90 % of bursts last less than 200[μs]. Moreover, the inter-arrival time of these majority μ bursts are short. For example, inter-bursts last less than 100μ for about 40% of Cache and Web services.

3.3.0.3 Locality of Traffic and Losses

Another key aspect of traffic properties of DCN is traffic and loss localization. Determining the spatial locality of traffic and packet losses also advocates DCN performance efficiency. Although the lack of predictable traffic matrix can provide no locality of traffic, DCN traffic studies evidence some locality trends of traffic and losses. First, surprisingly, there is a weak correlation between link utilization and localization of packet drops. For example, [32, 31, 113] observe that link utilization in the core level is higher than other levels. However, packet drops are highest at the edge [32, 31, 124]. The reason behind this loss is that traffic burstiness is higher at the rack level. In [31, 86, 54], authors noted that most traffic in the cloud is restricted to within a rack. In contrast, a significant fraction of traffic of the enterprise and universities DCs leaves the rack [32, 31]. Study on Facebook DCN [113] found that traffic is not localized to any particular layer and depends on the offered service. For example, in Hadoop server's majority of traffic is intra-rack while traffic in Web servers are largely intra-cluster. In addition to heavy racks traffic observation, most of their traffic goes to a few other ToRs [85]. In other words, there are few ToRs that are likely prone to be bottleneck. To recap, locality of traffic and losses are determined by applications that run within the DCN and how these applications are deployed on the servers.

To sum up, DCN traffic characteristics, including flow size and distribution, locality and episode of burstness, are highly dependant on applications. In addition, most flows in DCs are small in size (a few KBs). However, large flows whose length exceeds a few hundreds MB rarely exist. Moreover, plenty of these studies conclude that the traffic patterns are highly rack local. The reason behind this behavior is that DCN operators place the applications in such a way that the inter-rack traffic is reduced. Finally, although traffic patterns originated from a rack exhibit an ON/OFF pattern traffic and fit best with log-normal distributions, the entire traffic matrix is random and hardly predictable.

3.4 Discussion

3.4.1 A Multi-Tenant Data Center (MTDC) traffic

An infrastructure as a service (IaaS) cloud providers, such as Amazon AWS [1], Microsoft Azure [8], and Google Compute Engine [6], offers computing and storage resources to tenants (customers) by hosting their computing instances on the cloud provider's physical machines. In a traditional single-tenant cloud DC, providers offer a dedicated cloud service to their tenants, where resources are not shared with other tenants. During the last decade, many organizations choose to move and obtain their operations, storage and computation to multi-tenant (MT) DCs. However, we observed that only one paper (IBM) briefly studied network traffic behavior on MTDC, but the rest considers single-tenant DCs. We believe MTDCs exhibit traffic behaviors that are different from those of single-tenant DCs.

3.4.2 Implications for Load Balancing

Majority of the surveyed studies use the Equal-Cost Multiple Path (ECMP) [77] approach to load balance flows among ToRs. However, only a couple studies briefly mentions the implication of load balancing mechanisms. We believe that traffic congestion and loses observed in these studies are mainly due to their deployed load balancing mechanisms. It is well known that ECMP performs poorly in DC networks due to hash collisions. ECMP can cause congestions when two long-running flows are assigned to the same path. Also, ECMP doesn't adapt well to asymmetry in the network topology. As a result, a rich body of work such as Hedera [20], Conga [24], FastPass [109], Clove [87], Presto [73], Luopan [122], have been proposed in literature to address the problem of load balancing in DCN. However, none of these schemes are used in today's DCs. Future DCN traffic studies should investigate the implication of different load balancing schemes on the traffic behavior of DCs.

3.4.3 High Resolution Measurement

Real-time applications such as Web services are delay sensitive and strictly require QoS to be maintained. As a result, measuring and monitoring of real-time traffic at high-speed DCNs is substantial. Moreover, it is important for evaluating new protocols and architectures. Current studies instrument a small fraction of network devices for short time scale (few hours) to obtain fine-grained measurement, which does not reflect the entire traffic behavior of high speed DCs.

3.4.4 Lack of Traces

Traffic traces represent a very rich and useful source of information not only for understanding the traffic characteristics, but also in evaluating newly proposed solution approaches for DCs. Furthermore, due to the lack of these real traces, the majority of the proposed solutions for efficient DCN including topology design and traffic management are evaluated based on synthetic data, which is not a perfect representative of large-scale production workloads. There is a clear need for measuring and collecting real traces of DC traffic and especially for making them publicly available for DC researchers.

3.5 Conclusion

Different studies for measuring and understanding operational data center network behavior have been conducted during last decade. We review and classify these works focusing on the way they collect the traffic as well as the key observations made from these studies. We have learned that it is difficult to conduct entire fin-grained measurements for large scale data centers that produce terabytes of traffic per second. We also learned that DCN traffic typically correlated with applications and exhibit irregular and volatile patterns.

Clustered Multicast Source Routing for Large-Scale Cloud Data Centers

Jarallah Alqahtani, Hassan Sinky, Bechir Hamdaoui

IEEE Access, vol. 9, pp. 12 693– 12 705, 2021.

Chapter 4: Manuscript 3: Bert: Clustered Multicast Source Routing for Large-Scale Cloud Data Centers

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 chapter, 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.

4.1 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 [2, 45], with each tenant possibly running hundreds of workloads supported through thousands of virtual machines (VMs) running on different servers [8, 6, 1]. These workloads often involve one-to-many communications among the different servers running VMs supporting the same workload/application [115, 52, 46]. 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 chapter, 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 [78, 93, 58, 114, 92, 81, 41]. For instance, Elmo [114], 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 traffic. Elmo does so by taking advantage of programmable switches [3] 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 chapter 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 chapter is organized as follows. In Section 4.2 we discuss related works. Related state-of-the-art works and limitations are described in Section 4.3. We present Bert, the proposed multicast routing scheme, in Section 4.4. We study and evaluate the performances of Bert compared to those obtained under Elmo in Section 4.5. Finally, we conclude the chapter and discuss future directions in Section 4.6.

4.2 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 [75] and PIM [60] 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 [55, 30, 56] 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 [28], focus on other multicast routing problems such as scalability and load balancing in DCs. These works can be classified as decentralized [58, 57], SDN-based [93, 100, 68], or source-routed approaches [82, 90, 112, 62].

4.2.1 Source Routed Multicast

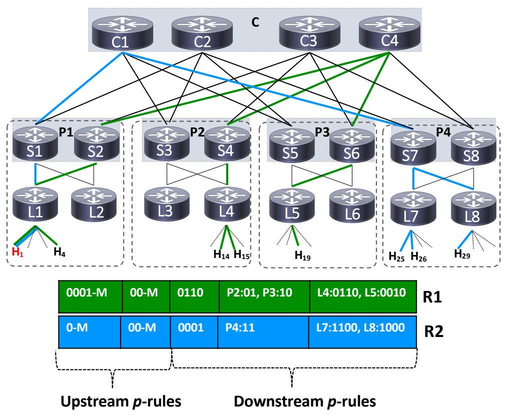
In [82, 90, 112, 62], bloom filters are used to encode forwarding state inside packets. These approaches import unnecessary traffic leakage (unnecessary multicast transmissions). Moreover, these approaches can't either support large group sizes or large network topology. On the other hand, Elmo [114] 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 4.4.

4.2.2 SDN-Based Multicast

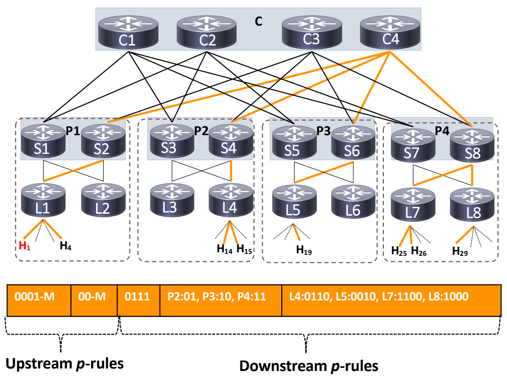
In [93], a centralized controller partitions the address space, and local address aggregation are implemented when the table space in switches is not enough. This approach suffers from exhausting network switch resources with a large number of flow-table entries, as well as a high number of switch entry updates. [106, 78, 89, 91] focus, on the other hand, on multicast tree reconstruction, but without addressing any scalability issues.

4.2.3 Decentralized Multicast

In [58] and [57], the authors present new address mapping schemes and discuss decentralized load balancing strategies, whereas [98] 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.


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
Upstream p-rules
Downstream p-rules

(a) Bert



(b) Elmo

Figure 4.1: 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.

4.3 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.

4.3.0.1 DC Topologies

Large-scale DCs typically are multi-rooted, tree-based topologies (e.g., fat-tree [19] and its variants [65, 96, 27]). 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 4.1. At the lowest layer, leaf (aka edge) switches are interconnected through spine (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.

4.3.0.2 Multi-Tenant DCs

In Multi-tenant DCs (like Microsoft Azure [8], Amazon Web Services (AWS) [1], and Google Cloud Platform [6]), 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 [9].

4.3.0.3 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 [11]. 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 [110], 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 [99]) to guarantee resource isolation and fair share of network resources among tenants.

4.3.0.4 Programmable Switches

Emerging programmable switch ASICs (e.g., Barefoot Tofino [7]) 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 [34]. P4, a language for Programming Protocol Independent Packet Processors, is a recent innovation providing an abstract model suitable for programming the network data plane.

4.4 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.

4.4.1 Elmo

Elmo [114] 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) [114], 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 4.1a, 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.

4.4.2 Motivating Example

To illustrate the limitations of Elmo and motivate the design of the proposed scheme, Bert, we present a detailed example in Figure 4.1. 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:

4.4.2.1 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 [76]. In Elmo, only one packet goes through upstream paths. Using Figure 4.1b 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.

4.4.2.2 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 4.1b 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 4.1a). 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 4.1b). 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 4.1b, 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.

4.4.3 Bert Architecture

Bert consists of three main components: a centralized controller, hypervisor switches, and network switches. Figure 4.2 illustrates our design architecture and is summarized below:

4.4.3.1 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.

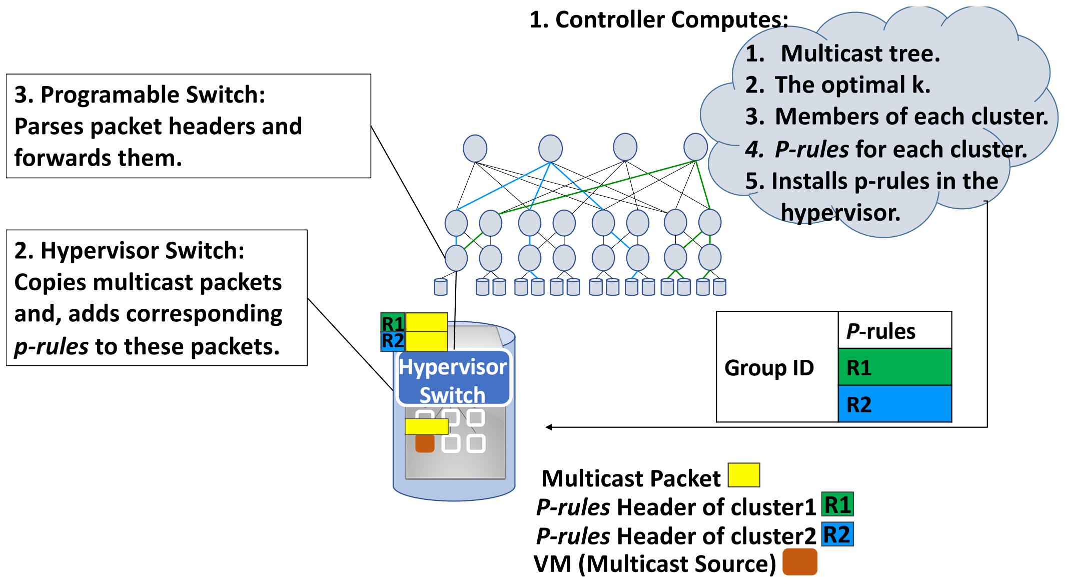

1. Controller Computes:
1. Multicast tree.
2. The optimal k.
3. Members of each cluster.
4. P-rules for each cluster.
5. Installs p-rules in the hypervisor.
3. Programable Switch: Parses packet headers and forwards them.
2. Hypervisor Switch: Copies multicast packets and, adds corresponding p-rules to 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 4.2: 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.

4.4.3.2 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.

4.4.3.3 Physical Switches (Network Switches)

As in Elmo, we assume DCs are running P4 programmable switches [34], which allow for parsing up to 512 bytes of the packet's header size [114]. 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.

4.4.4 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 4.3 how the amount of traffic incurred in the upstream (from leaf switches to core switches)

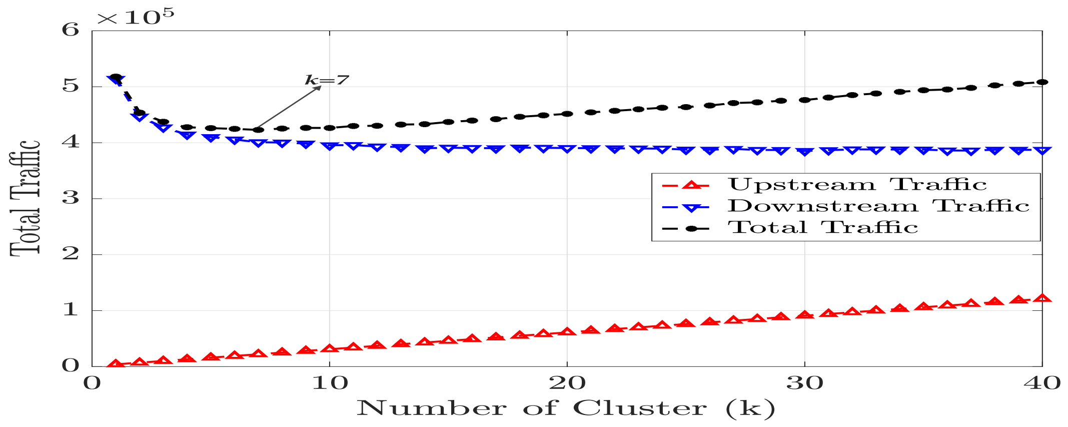


Figure 4.3: Multicast group size is 5000 members. Packet payload $B = 1500$ Bytes. The member placement strategy is leaf-based, described in Section 4.5. The total (payload + header) traffic is minimized when $k = 7$.

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.

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

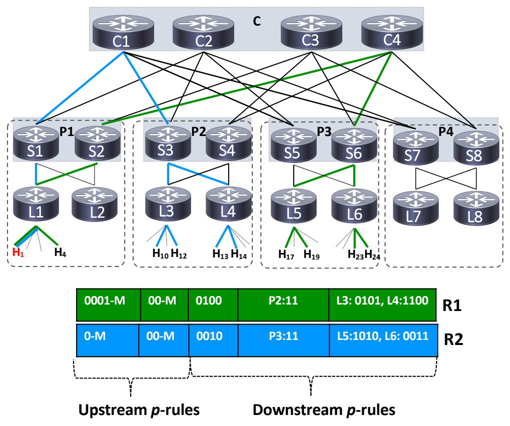
$$h(H + kB) + \sum_{i=1}^{h} r_i(\frac{H}{k} + B) \tag{4.1}$$

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 4.1a for illustration, we have $h = 2$, $r_1 = 3$, and $r_2 = 4$. In Eq. (4.1), 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.

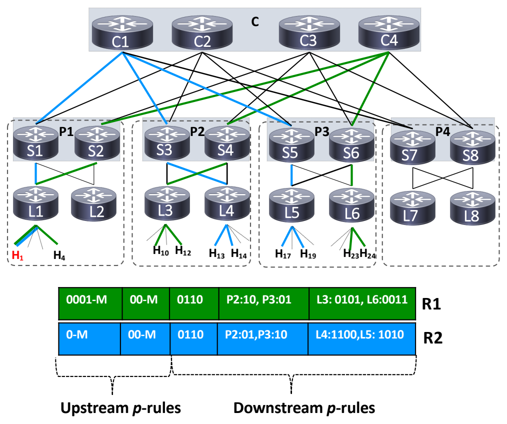
Note that, like header size H, the parameter r_i depends on the multicast group size and on the placement of the group members across the DC servers, but not on the number k of clusters chosen by the multicast group. After simple calculation, the optimal value of k that minimizes the total traffic can 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 4.4.5, 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.

4.4.5 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 4.4.4 is calculated for each multicast group independently. Before presenting



(a) Locality-aware clustering



(b) Locality-oblivious clustering

Figure 4.4: 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.

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 $L^j_{g,i}$ 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, $L_{g,i} = L^1_{g,i}||L^2_{g,i}||...||L^m_{g,i}$; 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, $L^{j}_{g,i}$ 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 4.1a, 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.

4.4.6 Pod-Based Versus Leaf-Based Clustering

In Section 4.4.5 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 4.4b, 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 4.4a).

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
 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
```

4.4.7 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 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 [10] 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:

4.4.7.1 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.

4.4.7.2 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.

4.4.7.3 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.

4.5 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 [114] and [93] where full-sized cloud DC topologies are simulated, rather than implementing Bert on a small test-bed. 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).

4.5.1 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 operators can specify how a physical switch processes and steers packets. Table 4.1 depicts the components used for our simulation environment.

Table 4.1: 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

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) *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.

   (ii) *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.

   (iii) *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.

4.5.2 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.

4.5.2.1 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 4.4.4. In pod-based and leaf-based placement strategies, where group members are close to one another, we observe from the results shown in Figure 4.5 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

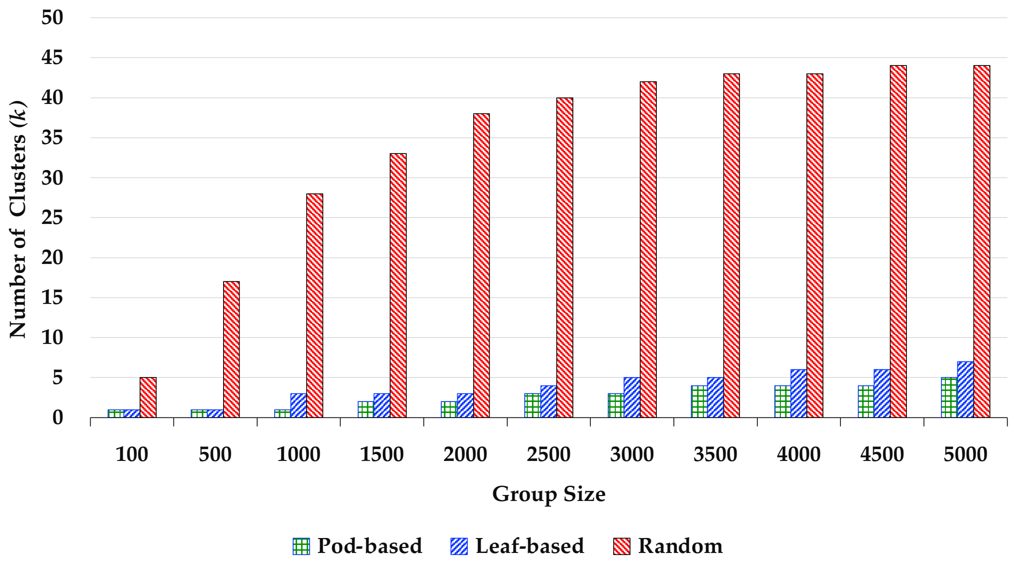


Figure 4.5: Number of clusters as a function of group size

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 4.6 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.

4.5.2.2 Header Size

Figure 4.7 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 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

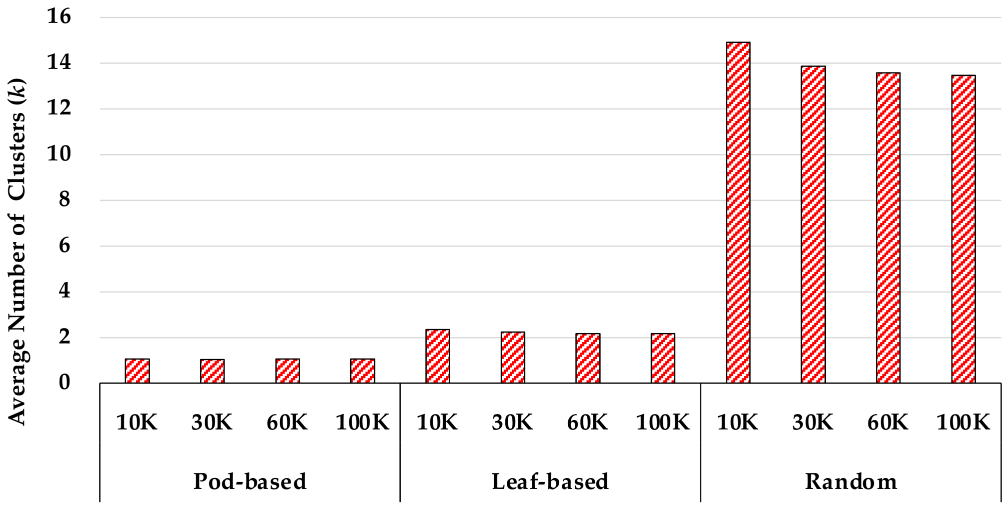


Figure 4.6: Average number of clusters as a function of number of multicast groups

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 4.4.4.

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 4.7a, 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 4.7b, 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 4.7c.

We observe that if group members (i.e. VMs) are placed too close to one another as

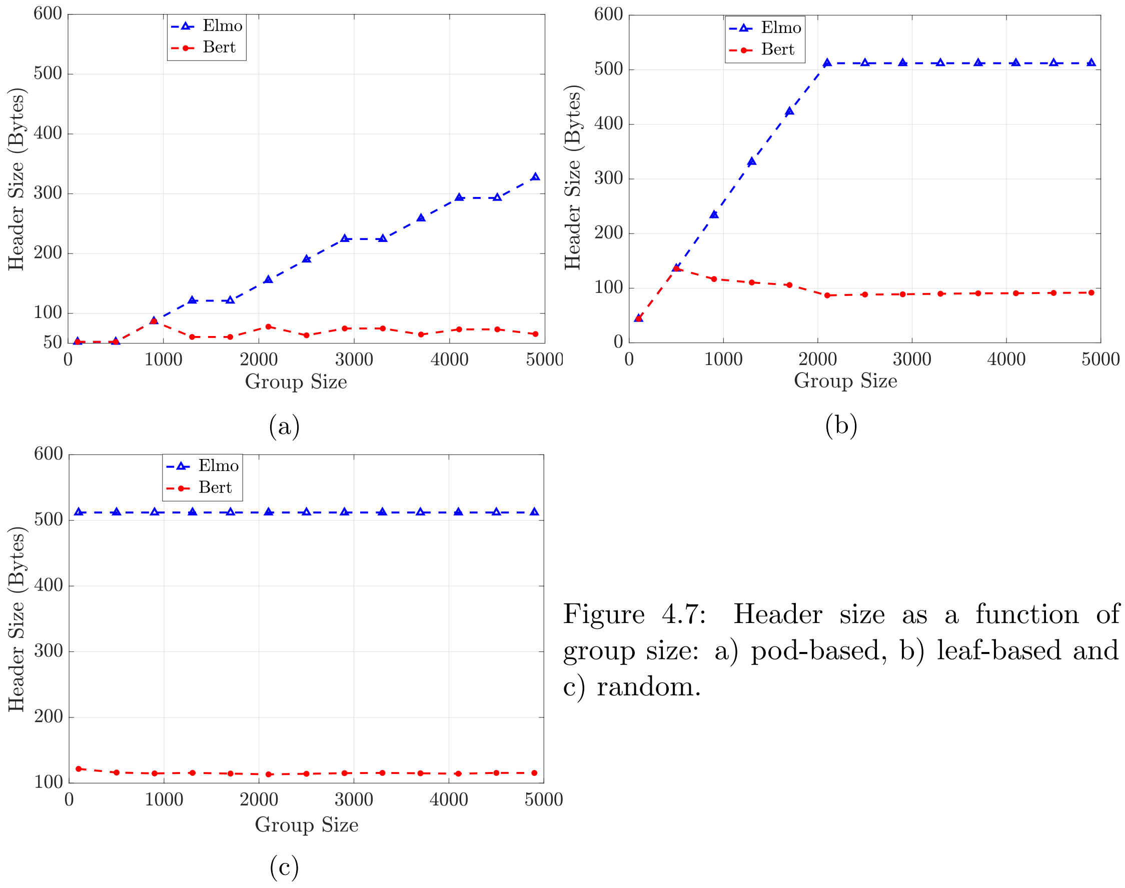


Figure 4.7: Header size as a function of group size: a) pod-based, b) leaf-based and c) random.

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 4.8 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

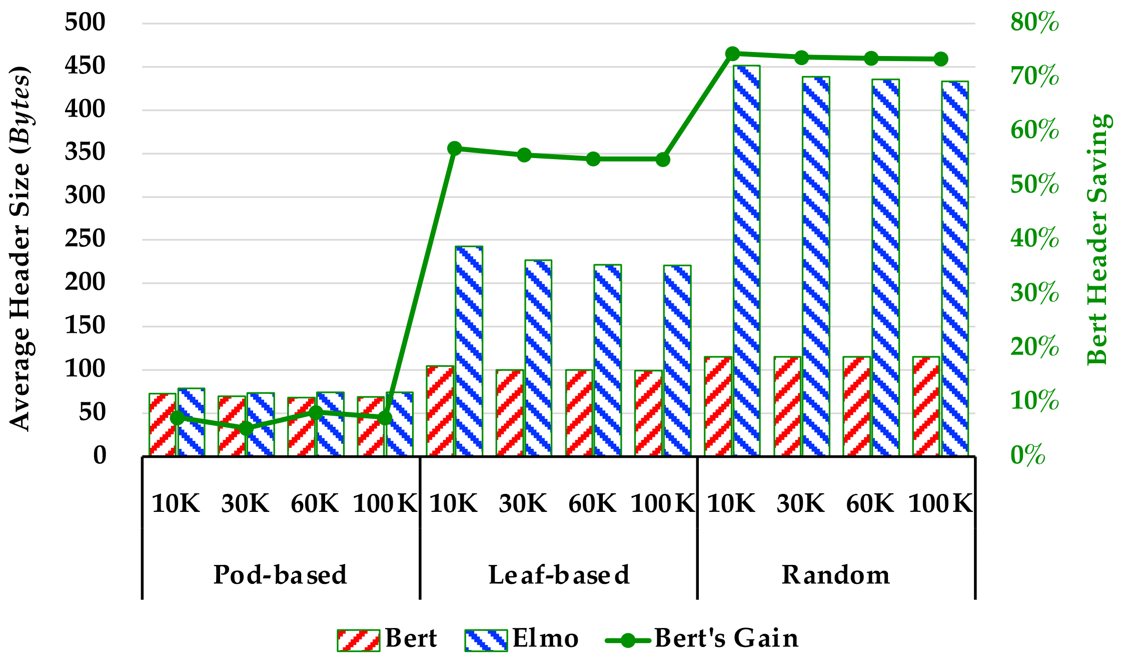
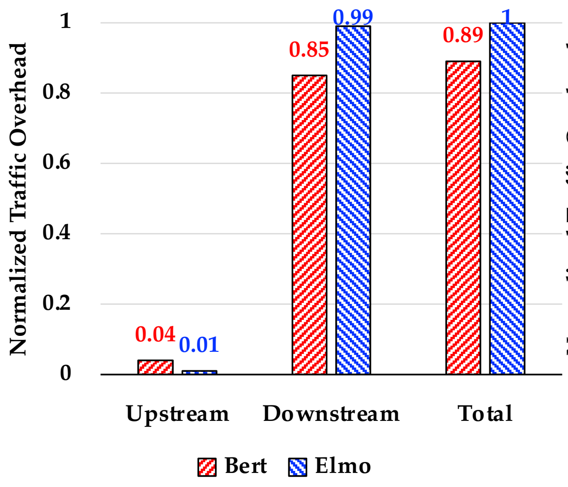


Figure 4.8: Average header size as a function of the number of multicast groups

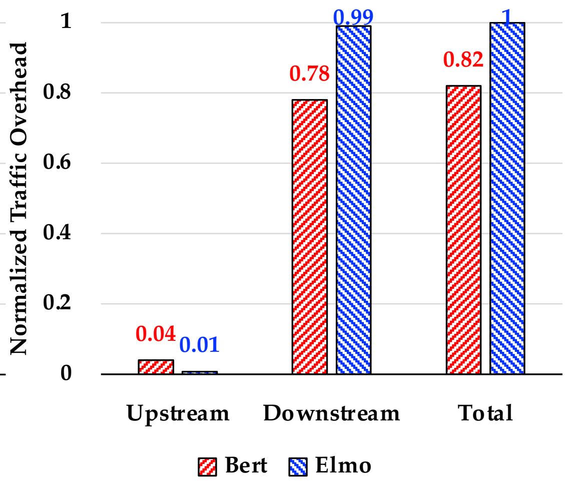
the leaf-based and random placement strategies respectively. In addition, Bert shows a 10% improvement in header size in the pod-based placement strategy. Note that Elmo's average header size is comparable to Bert only in the pod-based placement strategy particularly when group sizes are small (as previously shown in Figure 4.7a).

4.5.2.3 Traffic Cost

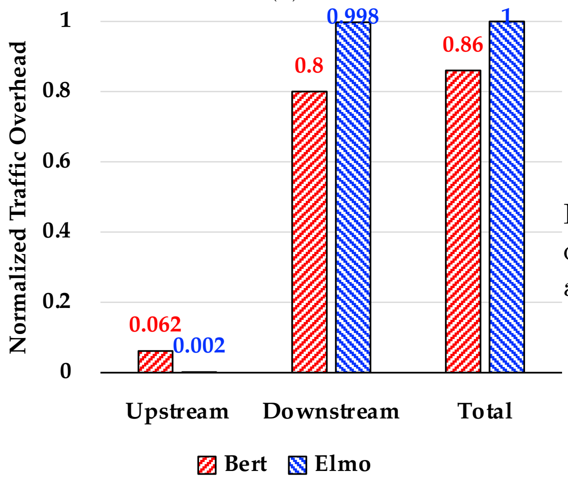
In this experiment, we evaluate the overall network overhead incurred by Elmo and Bert (normalized to Elmo). In Figure 4.9, we show the normalized traffic cost in terms of network overhead incurred in the upstream and downstream paths as described in Section 4.4. The figure evaluates pod-based (Figure 4.9a), leaf-based (Figure 4.9b), and random (Figure 4.9c) placement strategies using a multicast group with 5000 members



(a)

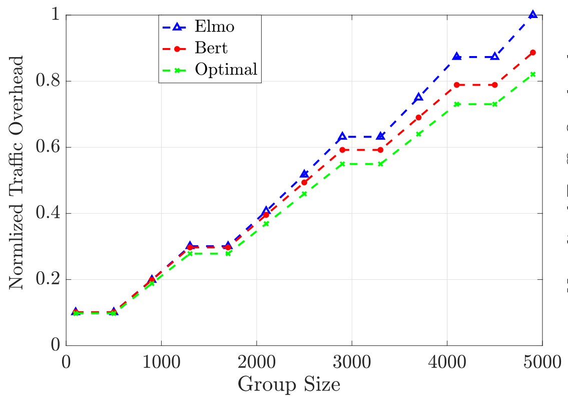


(b)

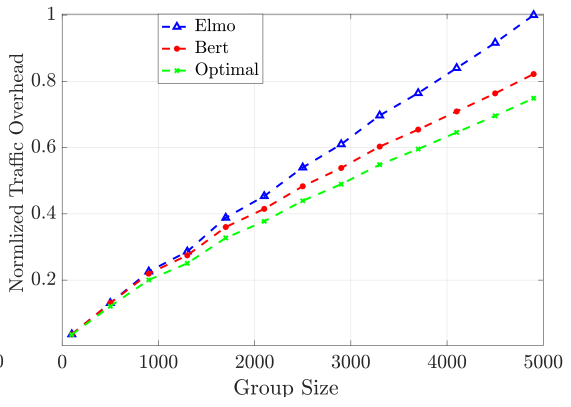


(c)

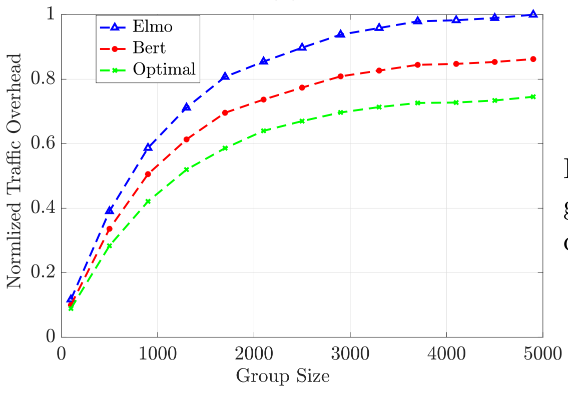
Figure 4.9: Upstream vs downstream traffic cost (d=5000): a) pod-based, b) leaf-based and c) random.



(a)



(b)

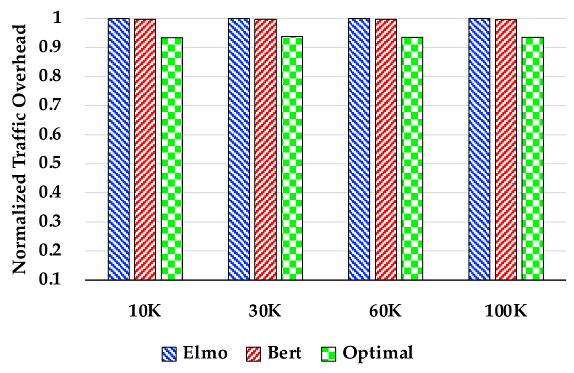


(c)

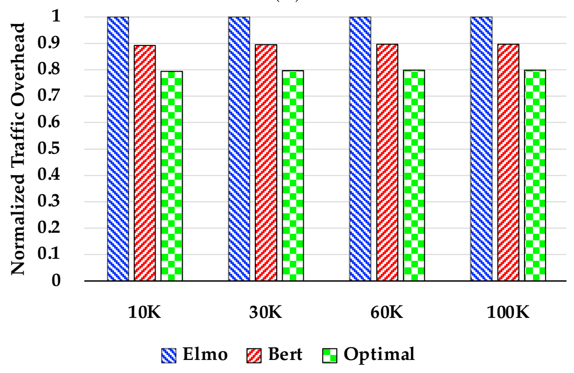
Figure 4.10: Traffic cost as a function of group size: a) pod-based, b) leaf-based and c) random.

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.

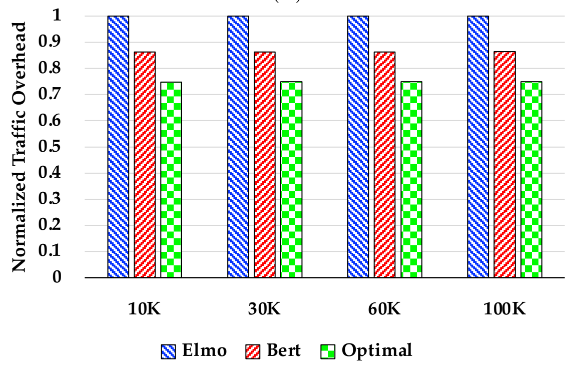
Figure 4.10 illustrates the network traffic cost as a function of group size using a single multicast group with different VM placement strategies. As shown in Figure 4.10a (pod-



(a)



(b)



(c)

Figure 4.11: Traffic cost as a function of the number of groups: a) pod-based, b) leaf-based and c) random.

based placement) and Figure 4.10b (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 4.10b when group sizes are greater than 4000 in the leaf-based placement strategy. Similar behavior is exhibited in Figure 4.10c 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 4.11 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 4.11a.

In Figure 4.12 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.

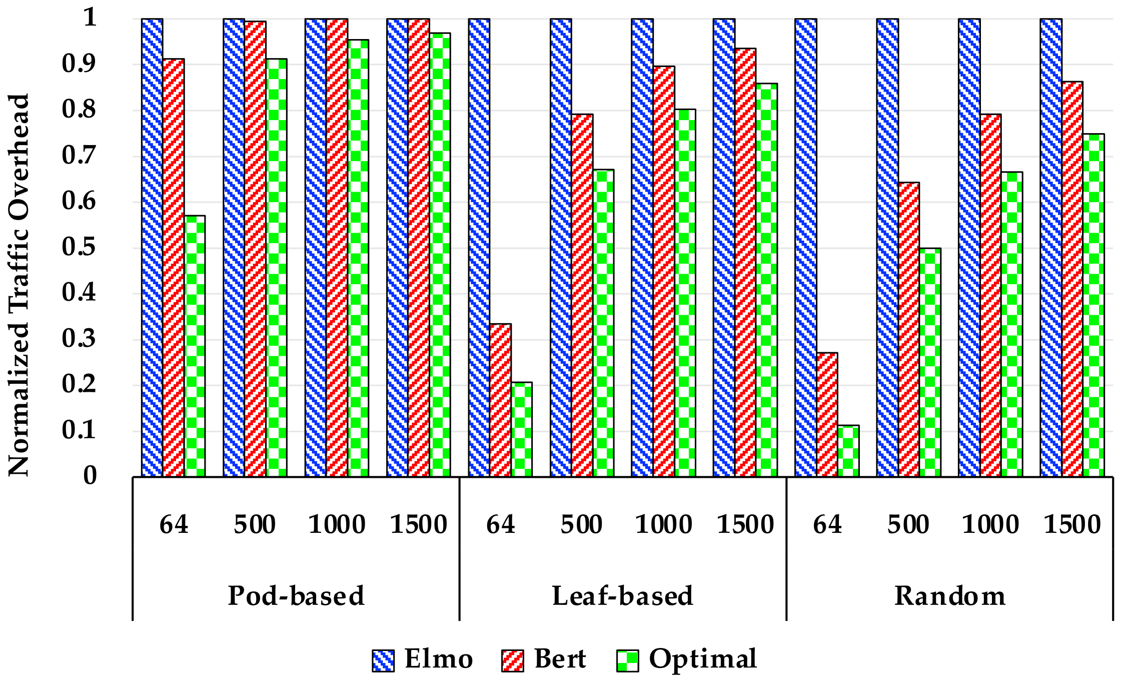


Figure 4.12: Traffic cost as a function of packet payload size in bytes.

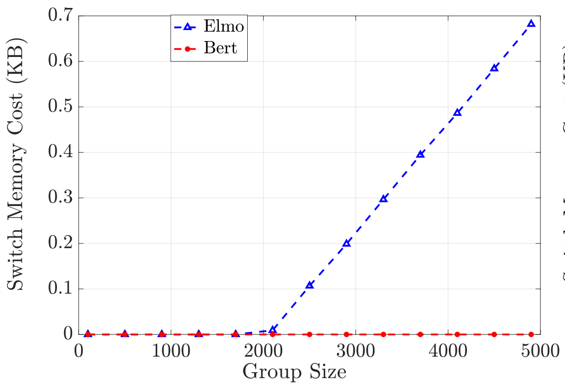
4.5.2.4 Switch Memory Cost

Programmable switches typically set restrictions on the packet header sizes they can parse (i.e., 512 bytes) [114]. 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.

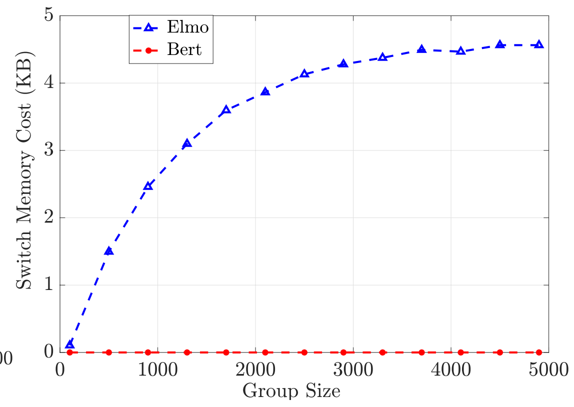
In Figures 4.13 and 4.14, we evaluate and show the total switch memory utilization by a single and multiple multicast groups, respectively. In both leaf-based and random placement strategies, Bert shows drastic improvements in switch memory utilization whereas Elmo must use switch memory to store its larger header sizes. Unlike Elmo, Bert does not use switch memory to store forwarding rules as depicted in Figures 4.13 and 4.14. 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).

4.5.2.5 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 each receiver, and than that of overlay-based multicast protocols [10], where all multicast packet replications occur at the source. Figure 4.15

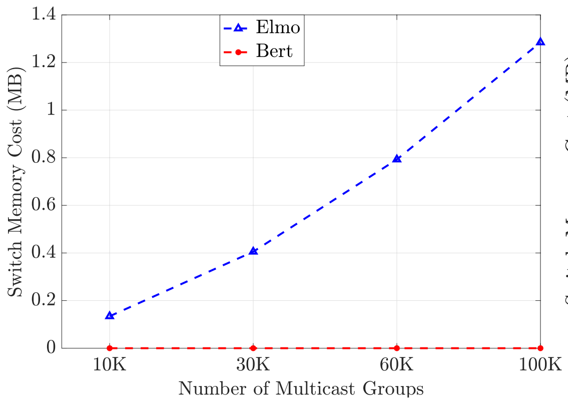


(a) Leaf-based placement

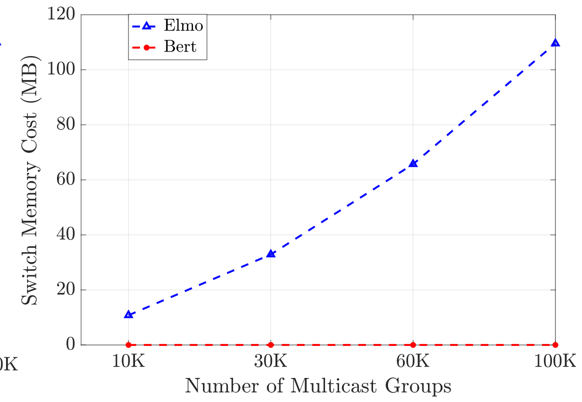


(b) Random placement

Figure 4.13: Switch memory use with one multicast group



(a) Leaf-based placement



(b) Random placement

Figure 4.14: Switch memory use with multiple multicast groups

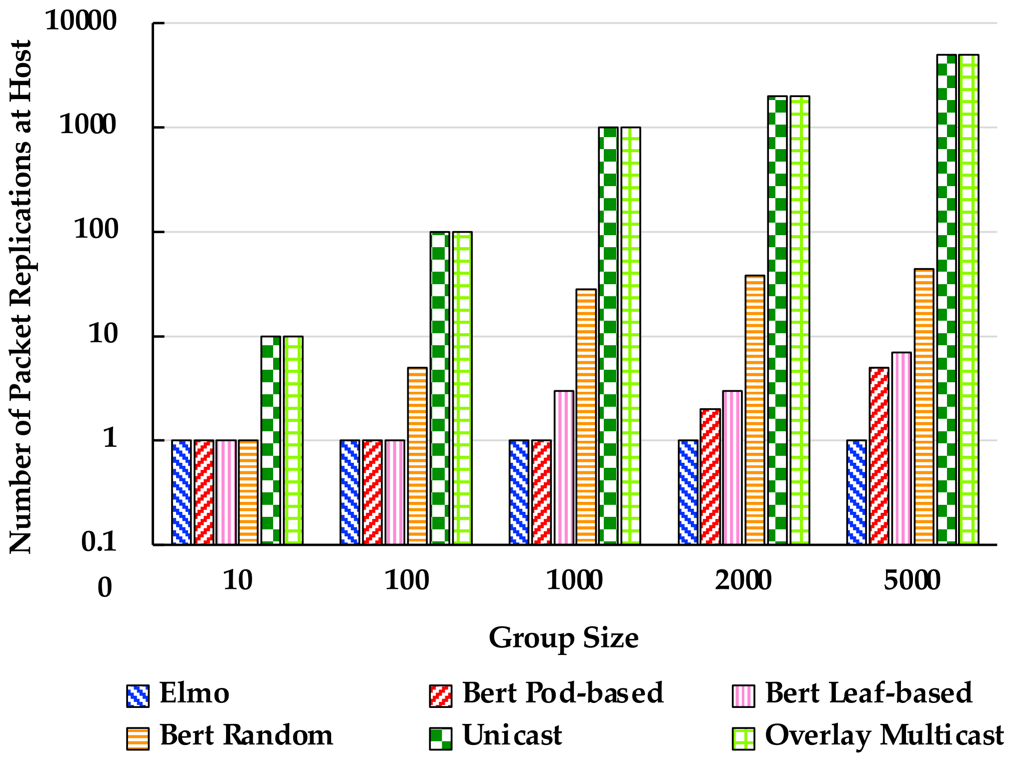


Figure 4.15: Number of packet replications as a function of group size

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.

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 [114]. 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.

4.6 Conclusion

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.

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

Jarallah Alqahtani, Bechir Hamdaoui, Rami Langar

IEEE Access, 2021.

Chapter 5: Manuscript 4: Ernie: Scalable Load-Balanced Multicast Source Routing for Cloud Data Centers

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 chapter 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.

5.1 Introduction

The past decade has witnessed a rapid boom of cloud computing services. Large cloud service providers, such as Amazon AWS [1], Microsoft Azure [8] and Google Cloud Platform [6], host hundreds of thousands of tenants, each of which possibly running hundreds of applications. Many of these applications [115, 52, 26] 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 [4]). Furthermore, today's data center (DC) networks operate under a single administrative domain and no longer require decentralized protocols like PIM [60] and IGMP [75]. 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 multipath property. For instance, such protocols usually choose a 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 [93, 40], 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 [89, 69]. 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 [114] and Bert [29], 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 [114] incurs extra overhead when the multicast group is large in size or dispersed across the network. It performs poorly under these scenarios by increasing network overhead (i.e., switch memory and packet size overheads). Although Bert [29] 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 [76]). 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.

First, Ernie proposes a novel method for scaling out the number of supported multicast groups. In particular, it appropriately constructs and organizes multicast header information inside packets in a manner that allows core/root switches to only forward down the needed information. Experiments using large scale DCs with 27,648 servers show that Ernie requires a downstream header size that is $10\times$ and $3\times$ smaller than that needed under Elmo and Bert, respectively. Second, Ernie introduces an effective multicast traffic load balancing technique across downstream links. Specifically, Ernie prudently assigns multicast groups to core switches to ensure the evenness of load distribution across the downstream links. Experiments also show that Ernie achieves about 25-65% better load balancing than other schemes at highly congested network layer.

The rest of this chapter is organized as follows. In Section 5.2, we discuss related works. Section 5.3 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 5.4. In Section 5.5, we study and evaluate the performances of Ernie and compare them with those achieved under existing schemes. Finally, we conclude the chapter in Section 5.6.

5.2 Related Works

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 on related works for DC multicast.

5.2.1 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 [93, 40] suffer from a high number of switch updates as well as limited switch group-table capacities. For example, in [93], 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 [82, 90, 112, 62], which 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 [91]). In [95, 80], 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 [114] and Bert [29], 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 [114] 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 [29] imposes bandwidth and end-host CPU overheads when the number of clusters (packet replications at the source) is large.

5.2.2 Load Balancing

Abundant research on DC traffic load balancing has mostly focused on unicast traffic [20, 24, 109, 87, 73, 122, 23]. 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 [89, 69, 107], 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 [40] 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 [58], 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 5.4.2.1, 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.

5.2.3 Ernie

To the best of our knowledge, none of the aforementioned schemes are able to efficiently achieve both load balancing and scalability of multicast in large scale DCs. In most of these schemes, when load balancing is attained, scalability is neglected and vise versa. To enable both scalable and load-balanced multicast, we propose Ernie, which aims to mend large DCs multicast scalability and load balancing seamlessly. Ernie complements these proposals and can be used to construct a better multicast routing system for large-scale DCs. 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 5.1, we present the drawbacks of prior solutions, thus motivating our design of Ernie.

Table 5.1: 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 [58]	Decentralized	High	Minimal	Small (5K)	No	Yes	No
iRP [89]	SDN	High	Minimal	Small (5K)	No	Yes	No
IP Multicast Scaling [93]	SDN	High / Medium with rule agg.	Minimal / Low with rule agg.	Medium(70K) / High (500K) with rule agg.	No	No	No
RDNA[95], COXcast[80]	Source-Routed	No	Medium	Huge (1M+)	Yes	No	No
LIPSIN [82], ESM [90]	Source-Routed	No	Medium	Huge (1M+)	Yes	No	No
Elmo [114]	Source-Routed	Medium / Low (with rule agg.)	Low / Medium (with rule agg.)	Huge (1M+)	No	No	No
Bert [29]	Source-Routed	No	Low	Huge (1M+)	No	No	Low
Ernie **(proposed scheme)**	Source-Routed	No	Low	Huge (1M+)	No	Yes	No

5.3 Background

5.3.1 DC Topologies

Large-scale production DCs typically are multi-rooted tree-based topologies (e.g., fat-tree [19] and its variants [65, 96, 27]). 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 core, from bottom to top, as shown in Fig 5.1. 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.

5.3.2 Multi-Tenant DCs

Multi-tenancy is one of the key features of cloud DCs. In Multi-tenant DCs (like Microsoft Azure [8], Amazon Web Services (AWS) [1], and Google Cloud Platform [6]), 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

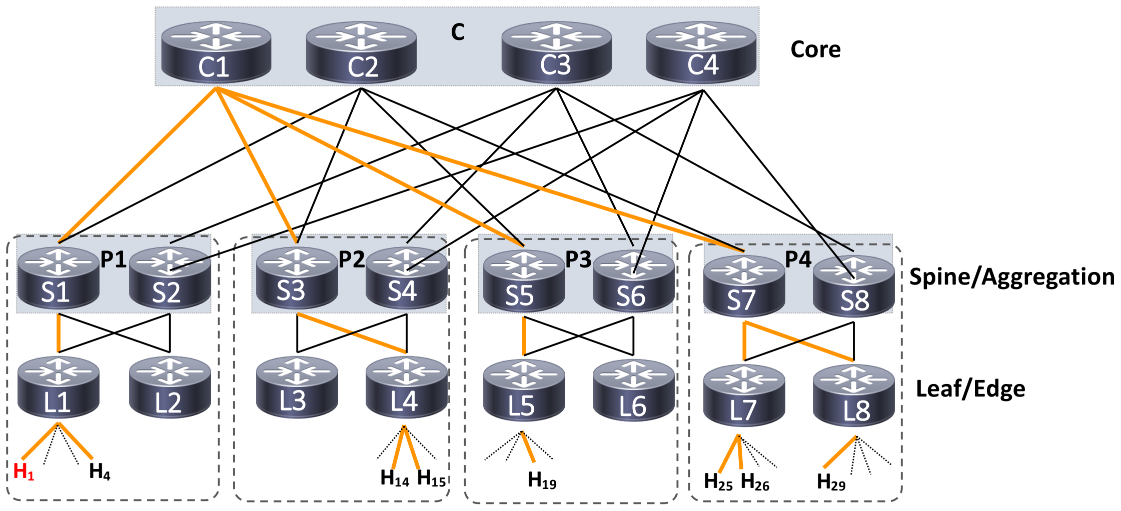


Figure 5.1: A three-tier multicast Clos tree topology with four pods. In this Example, there are 4 hosts under each leaf switch. H_1 is the multicast source, and H_4, H_{14}, H_{15}, H_{19}, H_{25}, H_{26}, and H_{29} are the destinations of the multicast group.

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 [9].

5.3.3 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 [11]. 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 [110], 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 [99]) to guarantee resource isolation and fair share of network resources among tenants.

5.3.4 Programmable Switches

Emerging programmable switch ASICs (e.g., Barefoot Tofino [7]) 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 [34]. P4, a language for Programming Protocol Independent Packet Processors, is a recent innovation providing an abstract model suitable for programming the network data plane.

5.3.5 State-of-the-Art Source-Routed Multicast Approaches

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 [114, 29] 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:

5.3.5.1 Elmo

Elmo [114] 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 [76]. Using Fig. 5.2a 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.

Although Elmo [114] has been shown to scale well with the number of multicast groups, it does not do so with multicast group sizes. It suffers from scalability issues in terms of incurred traffic overhead and increased switch memory usage when facing large group sizes, especially when group members are dispersed across the network.

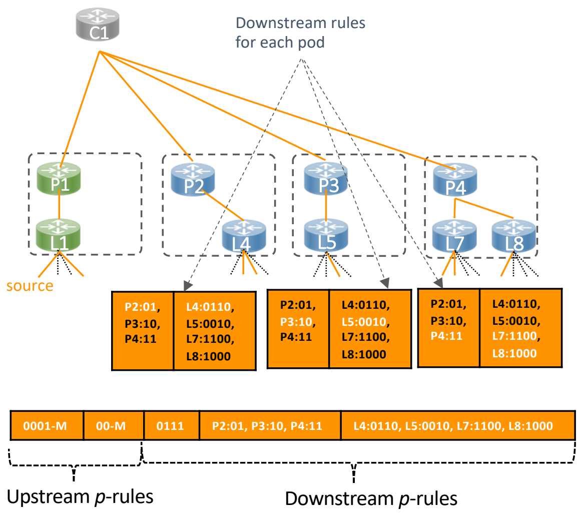
Additionally, in Elmo, acting towards heavy multicast traffic at downstream paths is not presented.

5.3.5.2 Bert

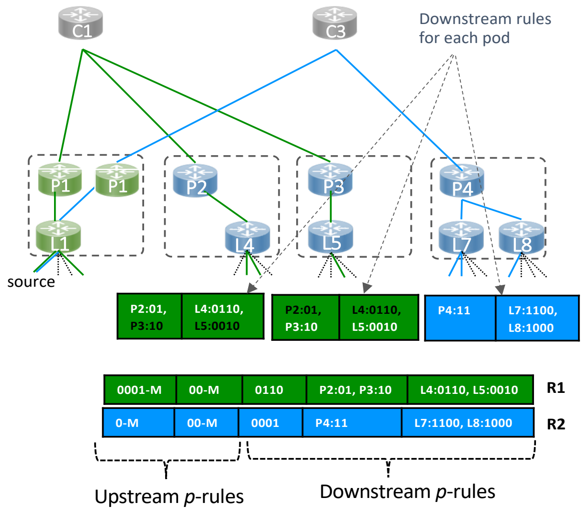
Bert [29] overcomes Elmo's aforementioned limitations by alleviating traffic congestion at downstream paths as it reduces both the packet header sizes and the number of extra packet transmissions, as well as eliminates switch memory utilization. It does so by adequately splitting multicast group members into multiple disjoint multicast clusters and encodes forwarding information for each cluster in the packet header. For example, as illustrated in Fig. 5.2b, Bert clusters the destination members into two clusters, $R1$ and $R2$. As for multicast tree encoding, Bert [29] follows Elmo [114].

Although Bert [29] overcomes some of Elmo's limitations, the clustering of a multicast group into multiple subgroups (or clusters) adopted by Bert induces packets replication at the source. For instance, when the group members are dispersed across leaf switches, the number of clusters (packet replications) can reach up to 48 for large group size (Sec. **??**). Such large numbers of packet replications at the source lead to an overall throughput reduction and a CPU utilization increase. Furthermore, similar to Elmo, multicast traffic congestion at downstream paths is neglected.

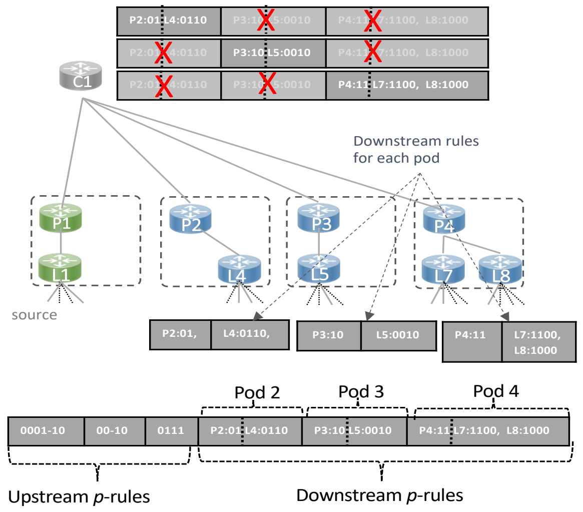
5.4 Ernie: The Proposed Multicast Scheme



(a) Elmo



(b) Bert



(c) Ernie

Figure 5.2: An example of multicast tree on a three-tier Clos topology with four pods. We use the same multicast tree in Fig.5.1. 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).

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.

5.4.1 Scalability

5.4.1.1 Motivation

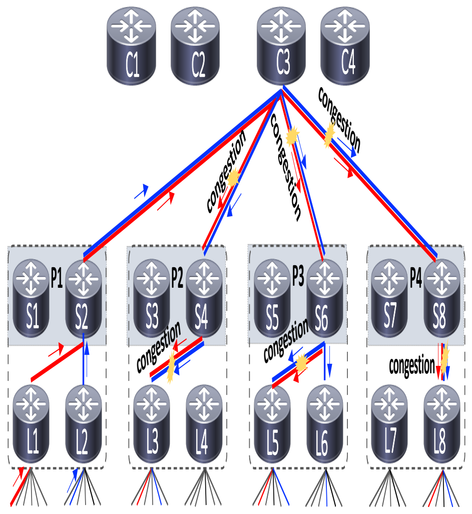
An efficient scalable multicast source routing design should aim to minimize (1) network overhead (i.e., switch memory and traffic overhead) and (2) CPU computation. These two aims can be obtained by reducing packet header size and by avoiding packet replication at the source.

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. 5.1, 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. 5.2a and 5.2b, 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 of other pods. By preventing this, we can locally (per pod) assign switch IDs. For example, in Figs. 5.2b and 5.2a, to identify switches, Bert and Elmo use three bits for each of the spine and leaf switches. Hence, in Bert (Fig. 5.2b), the average size of downstream *p*-rules received by each destination pod is 22 bits whereas with Elmo (Fig. 5.2a) is 43 bits. On the other hand, in Fig. 5.2c 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.

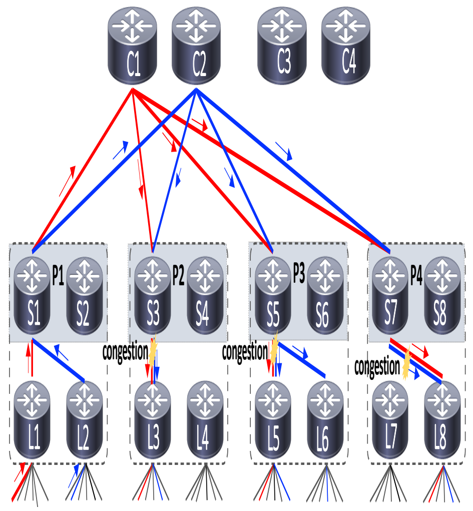
5.4.1.2 Design Choice

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 [29]. 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

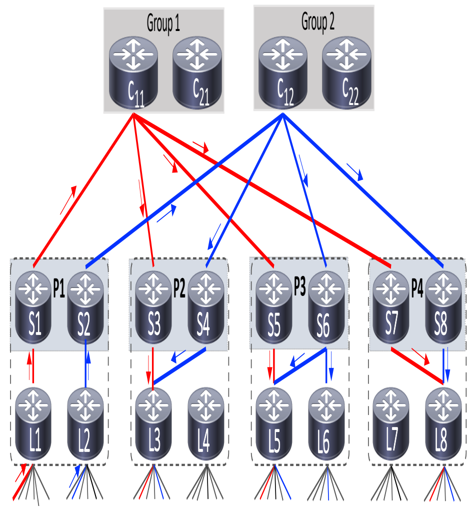
DC network topologies. For example, inter-pod traffic must pass through core switches where all inter-pod packets are sent to the core (root) switches, and then routed down to the host destinations. In other words, core switches are the intersection point between the upstream and downstream paths. Ernie 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.



(a) Random



(b) leaf layer obliviouse



(c) Ernie

Figure 5.3: 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 Ernie (a), leaf layer oblivious. e.g., RR and Miniforest (b), and Random (c). Unused links omitted for clarity

5.4.1.3 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. 5.2c, 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 [34] programmable switches ASICs (e.g., Barefoot Tofino [7]). When network switches receive a multicast packet, they parse, replicate (if needed), and forward the packet to the corresponding output ports. In Fig. 5.2c, 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. 5.2c, 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: ($L4:0110$).

5.4.2 Load Balancing

5.4.2.1 Motivation

We use the example given in Fig. 5.3 to illustrate the impact of the lack of even distribution of multicast traffic in multi-rooted Clos topologies DCNs. Suppose there are two 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 [61], which randomly chooses a single core switch as the rendezvous point (RP) or 2) ECMP [76], which balances traffic loads among multiple upstream paths through multipathing. For instance, as shown in Fig 5.3a, 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 [58]. 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.5.3b 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.

5.4.2.2 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 $j_1, j_2, ..., j_{k/2}$, each of which contains $k/2$ cores. Each core group connects to all pods through different spine switches. For example, in Fig.5.3c, 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.5.3c, 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 [89, 69], nor maintain the number of assigned multicast groups for each core switch [58].

5.4.3 Key Features of Ernie

5.4.3.1 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 compared to other approaches (e.g. Elmo [114] and Bert [29]), as shown in Section 5.4.1.1. 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.

5.4.3.2 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.

5.4.3.3 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.

5.4.3.4 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).

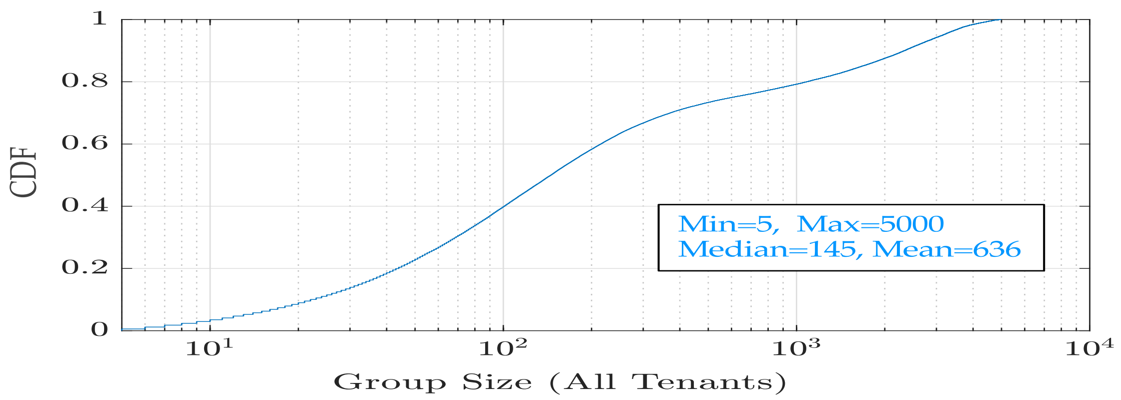
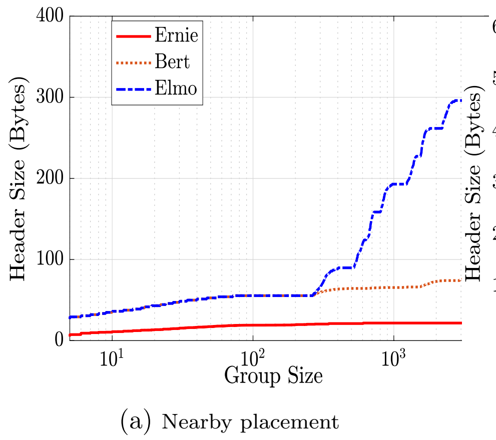


Figure 5.4: Group size follows random distribution for each tenant

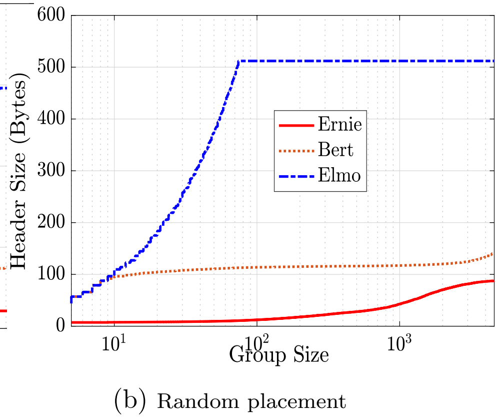
5.5 Performance Evaluation

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 [114, 29] 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.

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



(a) Nearby placement



(b) Random placement

Figure 5.5: Header size overhead in the downstream path as function of group size. Group size varying from 5-5000 members

shown in Fig. 5.4, where the average group size is 646.

5.5.1 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 5.3. We focus on four metrics/aspects (defined in Section 5.4.3): 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.

5.5.1.1 Header Sizes

We first evaluate the packet header overhead in the downstream path. Figs. 5.5a and 5.5b 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 5.4.1, 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 5.5.1.3.

Fig. 5.6 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. Nevertheless, as will be shown in

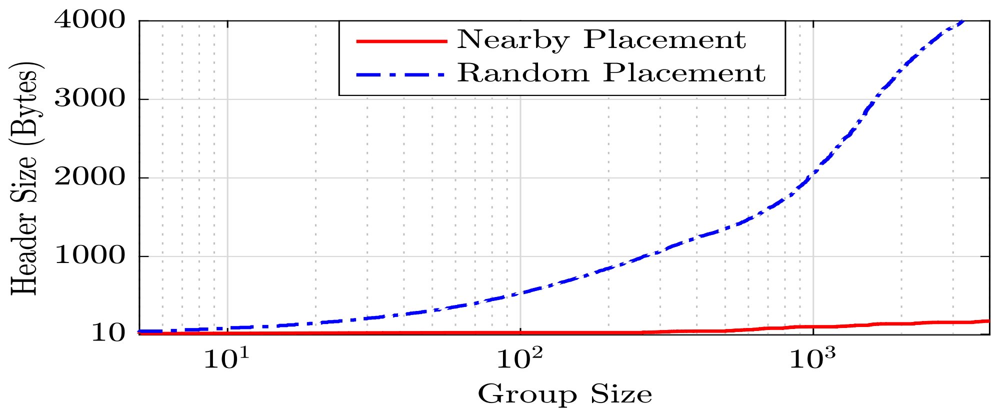
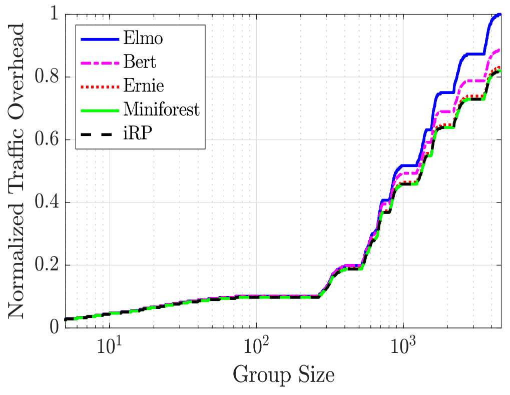


Figure 5.6: Ernie's upstream header size

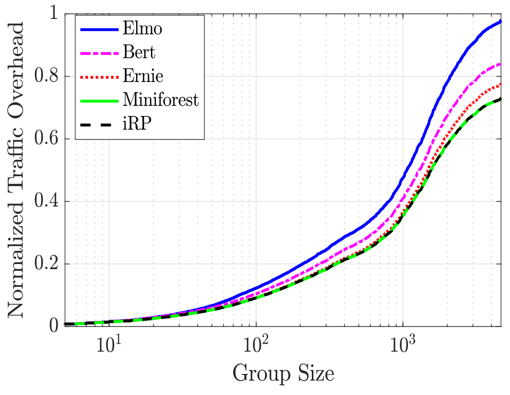
this section, the overall traffic, switch memory, and end-host CPU overheads achieved by Ernie are minimal.

5.5.1.2 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. 5.7a and 5.7b 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



(a) Nearby placement

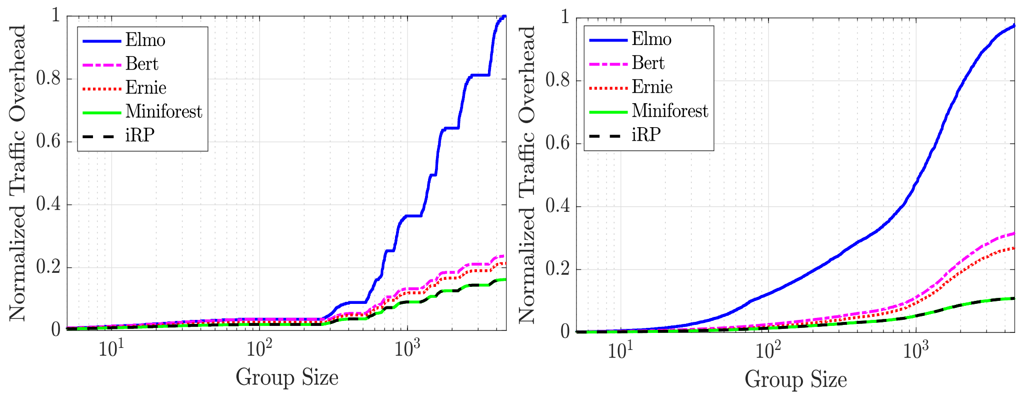


(b) Random placement

Figure 5.7: Traffic Overhead as function of group size for large packet payload of 1500 bytes. Group size varying from 5-5000 members

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 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. 5.8a and 5.8b 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.



(a) Nearby placement (b) Random placement

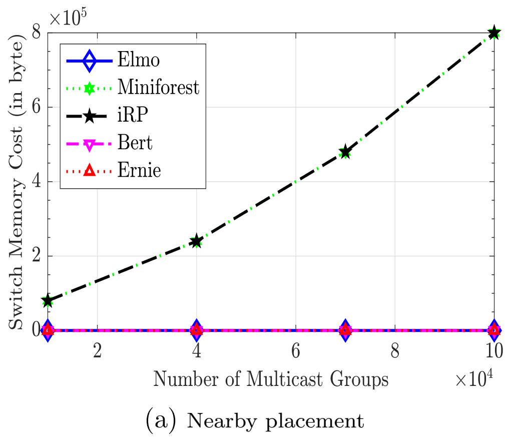
Figure 5.8: Traffic Overhead as function of group size for small packet payload sizes (64 bytes). Group size varying from 5-5000 members

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.

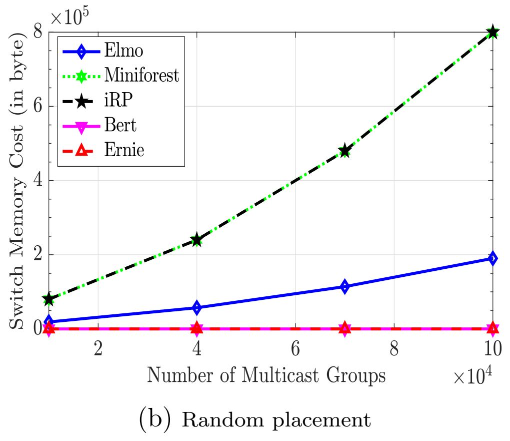
In summary, Ernie yields less traffic overhead when compared to the source-routed schemes, Elmo and Bert, while performing competitively when compared to the switch-based schemes (zero header overhead).

5.5.1.3 Switch Memory Cost

We next study the switch memory cost for all schemes in both placement strategies. First, in Fig. 5.9 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



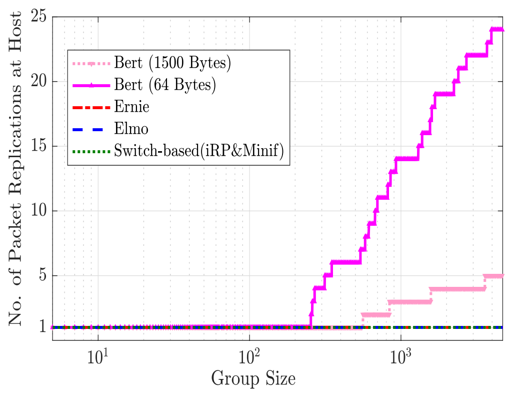
(a) Nearby placement



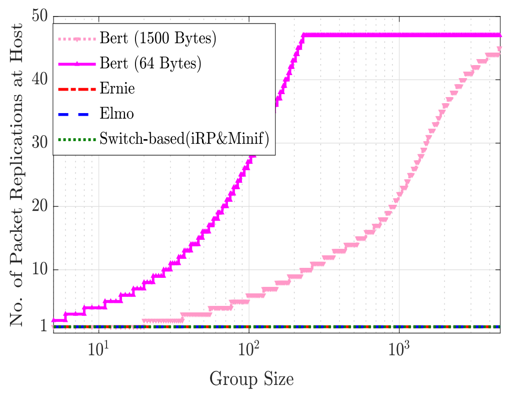
(b) Random placement

Figure 5.9: Switch Memory Costs

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 [4]. 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 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.



(a) Nearby placement



(b) Random placement

Figure 5.10: Number of packet replications at the host as a function of group size

5.5.1.4 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 5.10, 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. 5.10a, 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. 5.10b, for the Random placement with a small packet payload, the number of replications dramatically increases when the group size is increased.

5.5.2 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 [89]: 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) [58]: 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

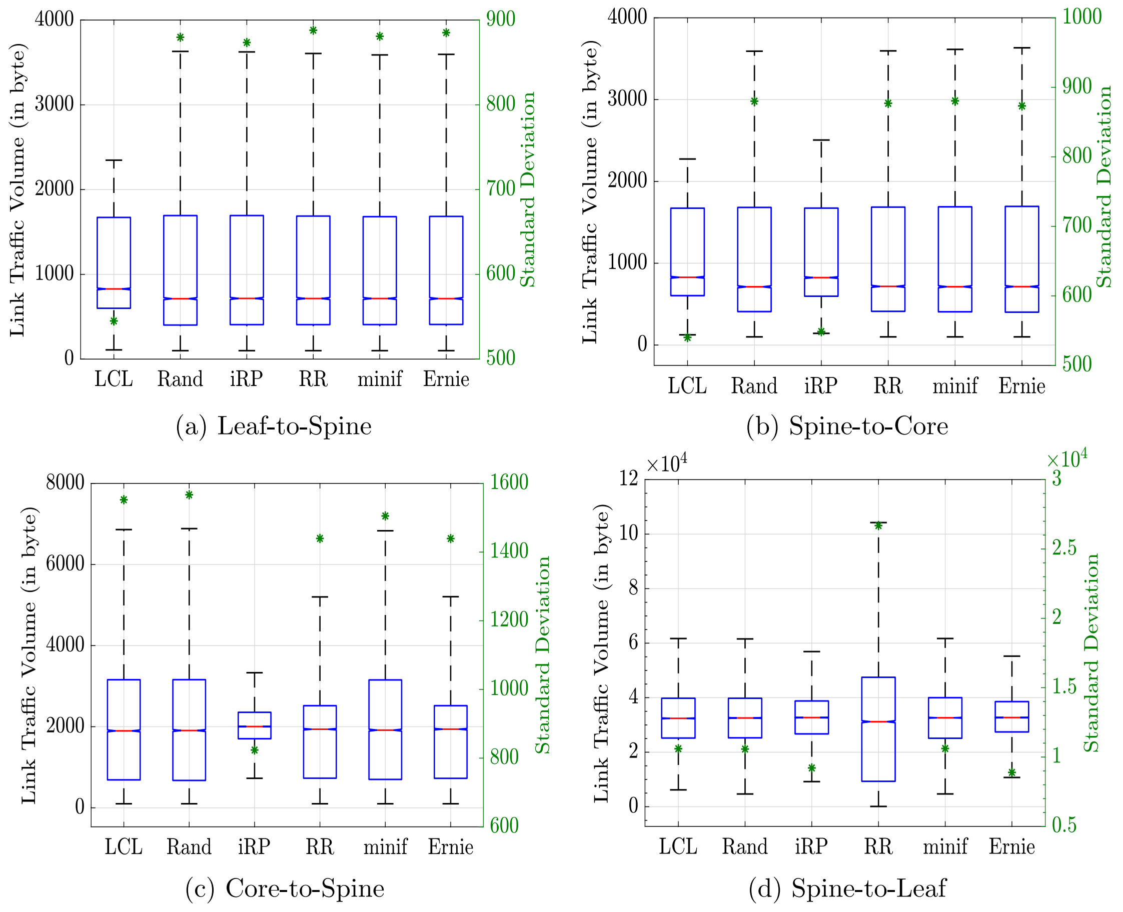
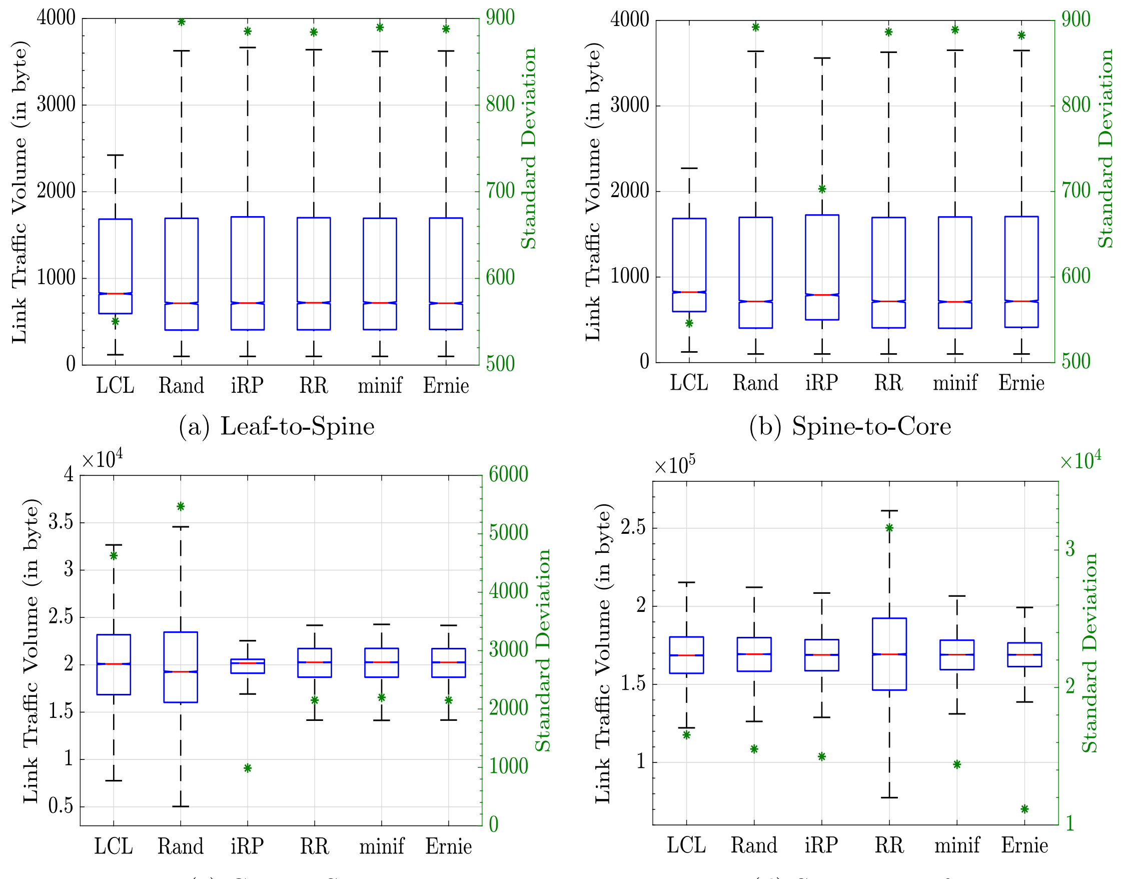


Figure 5.11: Load Balancing simulation for Nearby placement. In the boxplot, the central red mark is the median; whiskers represent the min and max value; boxes show the 25*th*, 50*th* (red line), and 75*th* percentiles. The right y-axis and the green star is for the Standard Deviation (SD).



(a) Leaf-to-Spine (b) Spine-to-Core

(c) Core-to-Spine (d) Spine-to-Leaf

Figure 5.12: 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 25th, 50th (red line), and 75th percentiles. The right y-axis and the green star (*) is for the Standard Deviation (SD).

the median; whiskers represent the min and max value; boxes show the $25th$, $50th$ (red line), and $75th$ 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. 5.11 and 5.12, 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. 5.11c) and spine-to-leaf (Fig. 5.11d) is about $3\times$ and $50\times$ more than spine-to-core (Fig. 5.11b) and leaf-to-spine (Fig. 5.11a), respectively. For the Random placement, traffic of core-to-spine (Fig. 5.12c) and spine-to-leaf (Fig. 5.12d) is about $28\times$ and $240\times$ more than spine-to-core (Fig. 5.12b) and leaf-to-spine (Fig. 5.12a), respectively.

In spine-to-leaf layer (Figs. 5.12d and 5.11d), 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.5.12d) 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 5.4.2.2, 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. 5.11d), 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. 5.11d and 5.12d) in both placement strategies. The reason is as shown in Section 5.4.2.1 Fig. 5.3b; 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. 5.12c and 5.11c), 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. 5.11a, 5.11b, 5.12a and 5.12b), Ernie, Rand, Miniforest, and RR perform closely to each other in both placement strategies. In these schemes, unlike LCL, the upstream path is predetermined; hence, they cannot avoid congested links. 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. 5.11b and 5.12b) traffic, and again this is due to the monitoring of enter-pod traffic.

5.5.3 Discussion: Pros and Cons

We discuss here some challenging scenarios that can limit the performance of Ernie. First, we have seen in Section 5.5.1.1 (Fig. 5.6) 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. 5.5b), resulting in higher traffic overhead (4× greater than Ernie (Fig. 5.7)). 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. 5.9b). 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. 5.10).

Second, Ernie performs worse than some of the other schemes (Figs. 5.11and 5.12) in terms of load balancing at the upstream layers. The reason is, as discussed in Section **??**, 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.

5.6 Conclusion

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.

Chapter 6: Conclusion

The IT enterprises are experiencing a paradigm shift by moving towards cloud computing, which consist of tens of thousands of servers interconnected by fast network, and a huge amount of computing and storage resources are provided. This dissertation explores and addresses issues arise in large cloud DCs. This chapter provides a summary of the thesis contribution and sheds light on directions of future work.

6.1 Summary of Contributions

First, we propose Circulant Fat-Tree topology, an improvement over the traditional Fat-Tree topology to better suit nowadays data center networks. Circulant Fat-Tree alleviates traffic congestion, reduces the average path lengths between communicating servers, and provides more possible paths between servers.

Second, 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 subclusters, 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.

Third, we propose Ernie, a load-balanced multicast source routing scheme suitable for large-scale cloud DCs. When compared to existing mulitcast 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.

6.2 Future Directions

In the following, we list some possible future research directions:

- **Handling Group Membership Dynamics.** 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.
- **Multicast Traffic Traces.** 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 [26]. 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.
- **VNFs/Middle-boxes Placement Considering Multicast Traffic.** Network Function Virtualization (NFV) is a promising technology that transforms from dedicated hardware implementations to software instances running in a virtualized

environment. Network operators often require the traffic to be steered through a sequence of multiple Virtual Network Functions VNFs (e.g. firewall or load balancer) which are also called middleboxes. These VNFs are placed with a predefined order, and these are also known as the Service Function Chaining (SFC). Network operators can leverage Software Defined Networking (SDN) to flexibly and agilely place VNFs. The placement of VMs that carry VNFs is crucial to the performance of offered services. In other words, an inefficient placement of VNFs can induce high network latency, traffic, and cost. Consequently, a lot of research efforts have been devoted to optimizing the placement of VNFs. However, most of these placement studies are limited to one-to-one unicast communication and cannot be extended to multicast traffic.

- **Inter-DC Multicast.** Increasing demand of online services on many spheres (e.g., health, social networking, streaming, business) leads to explosive growth of DC networks in both size and numbers. For example, Microsoft has 160+ physical DCs. Many cloud services require replicating massive/bulk data ranging from terabytes to petabytes from one DC to multiple DCs. For example, for availability, many cloud services typically require data or content (e.g., search indices, video files, and backups) to be dynamically copied from the DC hosting the data to many destination DCs that rely on such replica to run services. To provide end-users with guaranteed services, these data transfers are usually required to be completed within designated deadlines. To this end, several attempts in the last few years has been proposed to meet transfer deadlines or to reduce transmission costs [97]. First, there is a need to study the inter-DC traffic characteristics which should explain e.g. packet lose, resource utilizations, data sizes. Moreover, machine learning has become one of the hottest topics in both academia and industry. We believe machine learning can be applied to address the complex massive inter-DC transfer problems. For example, tailored machine learning algorithms can intelligently predict the popular data that need to be replicated/multicasted, and then keeping a copy of popular objects in multiple DCs that close to users.

Bibliography

[1] Amazon aws kernel description. https://aws.amazon.com/. Accessed: 2021-9-10.

[2] Amazon cloud has 1 million users. https://arstechnica.com/information-technology/2016/04/amazon-cloud-has-1-million-users-and-is-near-10-billion-in-annual-sales. Accessed: 2020-02-19.

[3] Barefoot tofino: World's fastest p4-programmable ethernet switch asics. https://barefootnetworks.com/products/brief-tofino/. Accessed: 2020-02-10.

[4] Cisco nexus 5000 series nx-os multicast routing command reference. https://www.cisco.com/c/en/us/td/docs/switches/datacenter/nexus5000/sw/command/reference/multicast/n5k-mcast-cr/n5k-igmpsnp_cmds_h.html. Accessed: 2021-9-29.

[5] Event tracing for windows (etw). https://docs.microsoft.com/en-us/windows/desktop/etw/event-tracing-portal. Accessed: 2019-01-22.

[6] Google compute engine. https://cloud.google.com/compute/. Accessed: 2021-9-10.

[7] Intel programmable ethernet switch products. `https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch.html`. Accessed: 2021-4-20.

[8] Microsoft azure. `https://azure.microsoft.com/`. Accessed: 2020-02-10.

[9] Netflix on aws. `https://aws.amazon.com/solutions/case-studies/netflix/`. Accessed: 2021-10-11.

[10] Overlay multicast in amazon virtual private cloud. `https://aws.amazon.com/articles/overlay-multicast-in-amazon-virtual-private-cloud/`. Accessed: 2021-10-11.

[11] What is a hypervisor? `https://www.vmware.com/topics/glossary/content/hypervisor`. Accessed: 2021-10-11.

[12] Sherif Abdelwahab and Bechir Hamdaoui. Flocking virtual machines in quest for responsive IoT cloud services. In *2017 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2017.

[13] Sherif Abdelwahab, Bechir Hamdaoui, and Mohsen Guizani. Cloud-assisted remote sensor network virtualization for distributed consensus estimation. *arXiv preprint arXiv:1501.03547*, 2015.

[14] Sherif Abdelwahab, Bechir Hamdaoui, Mohsen Guizani, and Ammar Rayes. Enabling smart cloud services through remote sensing: An internet of everything enabler. *IEEE Internet of Things Journal*, 1(3):276–288, 2014.

[15] Sherif Abdelwahab, Bechir Hamdaoui, Mohsen Guizani, and Taieb Znati. Cloud

of things for sensing-as-a-service: Architecture, algorithms, and use case. *IEEE Internet of Things Journal*, 3(6):1099–1112, 2016.

[16] Sherif Abdelwahab, Bechir Hamdaoui, Mohsen Guizani, and Taieb Znati. Network function virtualization in 5g. *IEEE Communications Magazine*, 54(4):84–91, 2016.

[17] Sherif Abdelwahab, Sophia Zhang, Ashley Greenacre, Kai Ovesen, Kevin Bergman, and Bechir Hamdaoui. When clones flock near the fog. *IEEE Internet of Things Journal*, 5(3):1914–1923, 2018.

[18] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *ACM SIGCOMM Computer Communication Review*, volume 38, pages 63–74. ACM, 2008.

[19] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. *ACM SIGCOMM computer communication review*, 38(4):63–74, 2008.

[20] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. Hedera: dynamic flow scheduling for data center networks. In *Nsdi*, volume 10, pages 89–92, 2010.

[21] Sultan Alanazi, Mehiar Dabbagh, Bechir Hamdaoui, Mohsen Guizani, and Nizar Zorba. Reducing data center energy consumption through peak shaving and locked-in energy avoidance. *IEEE Transactions on Green Communications and Networking*, 1(4):551–562, 2017.

[22] Sultan Alanazi and Bechir Hamdaoui. Energy-aware resource management frame-

work for overbooked cloud data centers with sla assurance. In *2018 IEEE global communications conference (GLOBECOM)*, pages 1–6. IEEE, 2018.

[23] Sultan Alanazi and Bechir Hamdaoui. Caft: Congestion-aware fault-tolerant load balancing for three-tier clos data centers. In *2020 International Wireless Communications and Mobile Computing (IWCMC)*, pages 1746–1751. IEEE, 2020.

[24] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Francis Matus, Rong Pan, Navindra Yadav, George Varghese, et al. Conga: Distributed congestion-aware load balancing for datacenters. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 503–514. ACM, 2014.

[25] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). *ACM SIGCOMM computer communication review*, 41(4):63–74, 2011.

[26] Jarallah Alqahtani, Sultan Alanazi, and Bechir Hamdaoui. Traffic behavior in cloud data centers: A survey. In *2020 International Wireless Communications and Mobile Computing (IWCMC)*, pages 2106–2111. IEEE, 2020.

[27] Jarallah Alqahtani and Bechir Hamdaoui. Rethinking fat-tree topology design for cloud data centers. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2018.

[28] Jarallah Alqahtani and Bechir Hamdaoui. Bert: Scalable source routed multicast for cloud data centers. In *2020 International Wireless Communications and Mobile Computing (IWCMC)*, pages 1752–1757. IEEE, 2020.

[29] Jarallah Alqahtani, Hassan H Sinky, and Bechir Hamdaoui. Clustered multicast source routing for large-scale cloud data centers. *IEEE Access*, 9:12693–12705, 2021.

[30] Sara Ayoubi, Chadi Assi, Yiheng Chen, Tarek Khalifa, and Khaled Bashir Shaban. Restoration methods for cloud multicast virtual networks. *Journal of Network and Computer Applications*, 78:180–190, 2017.

[31] Theophilus Benson, Aditya Akella, and David A Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 267–280. ACM, 2010.

[32] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. Understanding data center traffic characteristics. In *Proceedings of the 1st ACM workshop on Research on enterprise networking*, pages 65–72. ACM, 2009.

[33] Kashif Bilal, Marc Manzano, Samee U Khan, Eusebi Calle, Keqin Li, and Albert Y Zomaya. On the characterization of the structural robustness of data center networks. *IEEE Transactions on Cloud Computing*, 1(1):1–1, 2013.

[34] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014.

[35] Milan Bradonjić, Iraj Saniee, and Indra Widjaja. Scaling of capacity and reliability in data center networks. *ACM SIGMETRICS Performance Evaluation Review*, 42(2):46–48, 2014.

[36] Jeffrey Case, Russ Mundy, David Partain, and Bob Stewart. Introduction and applicability statements for internet-standard management framework. Technical report, 2002.

[37] Kai Chen, Ankit Singla, Atul Singh, Kishore Ramachandran, Lei Xu, Yueping Zhang, Xitao Wen, and Yan Chen. Osa: An optical switching architecture for data center networks with unprecedented flexibility. *IEEE/ACM Transactions on Networking*, 22(2):498–511, 2014.

[38] Weiwei Chen, Dong Wang, and Keqin Li. Multi-user multi-task computation offloading in green mobile edge cloud computing. *IEEE Transactions on Services Computing*, 12(5):726–738, 2018.

[39] Kenneth Church, Albert G Greenberg, and James R Hamilton. On delivering embarrassingly distributed cloud services. In *HotNets*, pages 55–60. Citeseer, 2008.

[40] W. Cui and C. Qian. Scalable and load-balanced data center multicast. In *2015 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, 2015.

[41] Wenzhi Cui and Chen Qian. Dual-structure data center multicast using software defined networking. *arXiv preprint arXiv:1403.8065*, 2014.

[42] Mehiar Dabbagh, Bechir Hamdaoui, Mohsen Guizani, and Ammar Rayes. Energy-efficient cloud resource management. In *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 386–391. IEEE, 2014.

[43] Mehiar Dabbagh, Bechir Hamdaoui, Mohsen Guizani, and Ammar Rayes. Release-time aware vm placement. In *2014 IEEE Globecom Workshops (GC Wkshps)*, pages 122–126. IEEE, 2014.

[44] Mehiar Dabbagh, Bechir Hamdaoui, Mohsen Guizani, and Ammar Rayes. Efficient datacenter resource utilization through cloud resource overcommitment. In *2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 330–335. IEEE, 2015.

[45] Mehiar Dabbagh, Bechir Hamdaoui, Mohsen Guizani, and Ammar Rayes. Energy-efficient resource allocation and provisioning framework for cloud data centers. *IEEE Transactions on Network and Service Management*, 12(3):377–391, 2015.

[46] Mehiar Dabbagh, Bechir Hamdaoui, Mohsen Guizani, and Ammar Rayes. Toward energy-efficient cloud computing: Prediction, consolidation, and overcommitment. *IEEE network*, 29(2):56–61, 2015.

[47] Mehiar Dabbagh, Bechir Hamdaoui, Mohsen Guizani, and Ammar Rayes. An energy-efficient vm prediction and migration framework for overcommitted clouds. *IEEE Transactions on Cloud Computing*, 6(4):955–966, 2016.

[48] Mehiar Dabbagh, Bechir Hamdaoui, and Ammar Rayes. Peak power shaving for reduced electricity costs in cloud data centers: Opportunities and challenges. *IEEE Network*, 34(3):148–153, 2020.

[49] Mehiar Dabbagh, Bechir Hamdaoui, Ammar Rayes, and Mohsen Guizani. Shaving data center power demand peaks through energy storage and workload shifting control. *IEEE Transactions on Cloud Computing*, 7(4), 2019.

[50] Mehiar Dabbagh, Ammar Rayes, Bechir Hamdaoui, and Mohsen Guizani. Peak shaving through optimal energy storage control for data centers. In *2016 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2016.

[51] Jeff Daniels. Server virtualization architecture and implementation. *XRDS: Crossroads, The ACM Magazine for Students*, 16(1):8–12, 2009.

[52] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[53] Valentin Del Piccolo, Ahmed Amamou, Kamel Haddadou, and Guy Pujolle. A survey of network isolation solutions for multi-tenant data centers. *IEEE Communications Surveys & Tutorials*, 18(4):2787–2821, 2016.

[54] Christina Delimitrou, Sriram Sankar, Aman Kansal, and Christos Kozyrakis. Echo: Recreating network traffic maps for datacenters with tens of thousands of servers. In *Workload Characterization (IISWC), 2012 IEEE International Symposium on*, pages 14–24. IEEE, 2012.

[55] Jun Duan and Yuanyuan Yang. Placement and performance analysis of virtual multicast networks in fat-tree data center networks. *IEEE Transactions on Parallel and Distributed Systems*, 27(10), 2016.

[56] Jun Duan and Yuanyuan Yang. Mcl: A cost-efficient nonblocking multicast interconnection network. *IEEE Transactions on Parallel and Distributed Systems*, 29(9):2046–2058, 2018.

[57] Fujie Fan, Bing Hu, and Kwan L Yeung. Distributed and dynamic multicast scheduling in fat-tree data center networks. In *2016 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2016.

[58] Fujie Fan, Bing Hu, Kwan L Yeung, and Minjian Zhao. Miniforest: Distributed

and dynamic multicasting in datacenter networks. *IEEE Transactions on Network and Service Management*, 16(3):1268–1281, 2019.

[59] Nathan Farrington, George Porter, Sivasankar Radhakrishnan, Hamid Hajabdolali Bazzaz, Vikram Subramanya, Yeshaiahu Fainman, George Papen, and Amin Vahdat. Helios: a hybrid electrical/optical switch architecture for modular data centers. *ACM SIGCOMM Computer Communication Review*, 40(4):339–350, 2010.

[60] Bill Fenner, Mark Handley, Hugh Holbrook, Isidor Kouvelas, Rishabh Parekh, Zhaohui Zhang, and Lianshu Zheng. Protocol independent multicast-sparse mode (pim-sm): Protocol specification (revised). *RFC*, 7761:1–137, 2016.

[61] Bill Fenner, Mark Handley, Hugh Holbrook, Isidor Kouvelas, Rishabh Parekh, Zhaohui Zhang, and Lianshu Zheng. Protocol independent multicast-sparse mode (pim-sm): Protocol specification (revised). *RFC*, 7761:1–137, 2016.

[62] Xiaofeng Gao, Tao Chen, Zongchen Chen, and Guihai Chen. Nemo: novel and efficient multicast routing schemes for hybrid data center networks. *Computer Networks*, 138:149–163, 2018.

[63] Saurabh Kumar Garg, Steve Versteeg, and Rajkumar Buyya. Smicloud: A framework for comparing and ranking cloud services. In *2011 Fourth IEEE International Conference on Utility and Cloud Computing*, pages 210–218. IEEE, 2011.

[64] Albert Greenberg, James R Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: a scalable and flexible data center network. In *ACM SIGCOMM computer communication review*, volume 39, pages 51–62. ACM, 2009.

[65] Albert Greenberg, James R Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: a scalable and flexible data center network. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, pages 51–62, 2009.

[66] Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu. Bcube: a high performance, server-centric network architecture for modular data centers. *ACM SIGCOMM Computer Communication Review*, 39(4):63–74, 2009.

[67] Chuanxiong Guo, Haitao Wu, Kun Tan, Lei Shi, Yongguang Zhang, and Songwu Lu. Dcell: a scalable and fault-tolerant network structure for data centers. In *ACM SIGCOMM Computer Communication Review*, volume 38, pages 75–86. ACM, 2008.

[68] Zehua Guo, Yang Xu, Marco Cello, Junjie Zhang, Zicheng Wang, Mingjian Liu, and H Jonathan Chao. Jumpflow: Reducing flow table usage in software-defined networks. *Computer Networks*, 92:300–315, 2015.

[69] Zhiyang Guo, Jun Duan, and Yuanyuan Yang. On-line multicast scheduling with bounded congestion in fat-tree data center networks. *IEEE Journal on Selected Areas in Communications*, 32(1):102–115, 2014.

[70] Bechir Hamdaoui, Mohamed Alkalbani, Ammar Rayes, and Nizar Zorba. Iot-share: A blockchain-enabled iot resource sharing on-demand protocol for smart city situation-awareness applications. *IEEE Internet of Things Journal*, 7(10):10548–10561, 2020.

[71] Bechir Hamdaoui, Mohamed Alkalbani, Taieb Znati, and Ammar Rayes. Unleashing the power of participatory iot with blockchains for increased safety and situation awareness of smart cities. *IEEE Network*, 34(2):202–209, 2019.

[72] Bechir Hamdaoui, Nizar Zorba, and Ammar Rayes. Participatory iot networks-on-demand for safe reliable and responsive urban cities. *IEEE Blockchain Technical Briefs*, 2019.

[73] Keqiang He, Eric Rozner, Kanak Agarwal, Wes Felter, John Carter, and Aditya Akella. Presto: Edge-based load balancing for fast datacenter networks. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 465–478. ACM, 2015.

[74] Americas Headquarters. Cisco data center infrastructure 2.5 design guide. *Cisco Validated Design I*, 2013.

[75] Hugh Holbrook, Brad Cain, and Brian Haberman. Using internet group management protocol version 3 (igmpv3) and multicast listener discovery protocol version 2 (mldv2) for source-specific multicast. *RFC 4604 (Proposed Standard), Internet Engineering Task Force*, 2006.

[76] Christian Hopps. Analysis of an equal-cost multi-path algorithm. Technical report, RFC Editor, 2000.

[77] Christian Hopps. Analysis of an equal-cost multi-path algorithm. Technical report, 2000.

[78] Aakash Iyer, Praveen Kumar, and Vijay Mann. Avalanche: Data center multicast

using software defined networking. In *Int. conference on communication systems and networks (COMSNETS)*, pages 1–8. IEEE, 2014.

[79] Virajith Jalaparti, Peter Bodik, Srikanth Kandula, Ishai Menache, Mikhail Rybalkin, and Chenyu Yan. Speeding up distributed request-response workflows. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 219–230. ACM, 2013.

[80] Wen-Kang Jia. A scalable multicast source routing architecture for data center networks. *IEEE Journal on Selected Areas in Communications*, 32(1):116–123, 2013.

[81] Wen-Kang Jia. A scalable multicast source routing architecture for data center networks. *IEEE Journal on Selected Areas in Communications*, 32(1):116–123, 2013.

[82] Petri Jokela, András Zahemszky, Christian Esteve Rothenberg, Somaya Arianfar, and Pekka Nikander. Lipsin: line speed publish/subscribe inter-networking. *ACM SIGCOMM Computer Communication Review*, 39(4):195–206, 2009.

[83] Sangeetha Abdu Jyothi, Mo Dong, and P Brighten Godfrey. Towards a flexible data center fabric with source routing. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, pages 1–8, 2015.

[84] Sangeetha Abdu Jyothi, Ankit Singla, P Brighten Godfrey, and Alexandra Kolla. Measuring and understanding throughput of network topologies. In *High Performance Computing, Networking, Storage and Analysis, SC16: International Conference for*, pages 761–772. IEEE, 2016.

[85] Srikanth Kandula, Jitendra Padhye, and Paramvir Bahl. Flyways to de-congest data center networks. 2009.

[86] Srikanth Kandula, Sudipta Sengupta, Albert Greenberg, Parveen Patel, and Ronnie Chaiken. The nature of data center traffic: measurements & analysis. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement*, pages 202–208. ACM, 2009.

[87] Naga Katta, Mukesh Hira, Aditi Ghag, Changhoon Kim, Isaac Keslassy, and Jennifer Rexford. Clove: How i learned to stop worrying about the core and love the edge. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, pages 155–161. ACM, 2016.

[88] Teemu Koponen, Keith Amidon, Peter Balland, Martín Casado, Anupam Chanda, Bryan Fulton, Igor Ganichev, Jesse Gross, Paul Ingram, Ethan Jackson, et al. Network virtualization in multi-tenant datacenters. In *11th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 14)*, pages 203–216, 2014.

[89] Ammar Latif, Pradeep Kathail, Sachin Vishwarupe, Subha Dhesikan, Abdallah Khreishah, and Yaser Jararweh. Multicast optimization for clos fabric in media data centers. *IEEE Transactions on Network and Service Management*, 16(4):1855–1868, 2019.

[90] Dan Li, Yuanjie Li, Jianping Wu, Sen Su, and Jiangwei Yu. Esm: Efficient and scalable data center multicast routing. *IEEE/ACM Transactions on Networking*, 20(3):944–955, 2011.

[91] Dan Li, Mingwei Xu, Ying Liu, Xia Xie, Yong Cui, Jingyi Wang, and Guihai Chen. Reliable multicast in data center networks. *IEEE Transactions on Computers*, 63(8):2011–2024, 2013.

[92] Dan Li, Mingwei Xu, Ming-chen Zhao, Chuanxiong Guo, Yongguang Zhang, and Min-you Wu. Rdcm: Reliable data center multicast. In *2011 Proceedings IEEE INFOCOM*, pages 56–60. IEEE, 2011.

[93] Xiaozhou Li and Michael J Freedman. Scaling ip multicast on datacenter topologies. In *Proceedinsgs of the ninth ACM conference on Emerging networking experiments and technologies*, pages 61–72, 2013.

[94] Zhuofan Liao, Ruiming Zhang, Shiming He, Daojian Zeng, Jin Wang, and Hye-Jin Kim. Deep learning-based data storage for low latency in data center networks. *IEEE Access*, 7:26411–26417, 2019.

[95] Alextian Liberato, Magnos Martinello, Roberta L Gomes, Arash F Beldachi, Emilio Salas, Rodolfo Villaca, Moisés RN Ribeiro, Koteswararao Kondepu, George Kanellos, Reza Nejabati, et al. Rdna: Residue-defined networking architecture enabling ultra-reliable low-latency datacenters. *IEEE Transactions on Network and Service Management*, 15(4):1473–1487, 2018.

[96] Vincent Liu, Daniel Halperin, Arvind Krishnamurthy, and Thomas Anderson. F10: A fault-tolerant engineered network. In *Presented as part of the 10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13)*, pages 399–412, 2013.

[97] Long Luo, Hongfang Yu, Klaus-Tycho Foerster, Max Noormohammadpour, and

Stefan Schmid. Inter-datacenter bulk transfers: Trends and challenges. *IEEE Network*, 34(5):240–246, 2020.

[98] S. Luo, H. Yu, K. Li, and H. Xing. Efficient file dissemination in data center networks with priority-based adaptive multicast. *IEEE Journal on Selected Areas in Communications*, 38(6), 2020.

[99] Mallik Mahalingam, Dinesh G Dutt, Kenneth Duda, Puneet Agarwal, Lawrence Kreeger, T Sridhar, Mike Bursell, and Chris Wright. Virtual extensible local area network (vxlan): A framework for overlaying virtualized layer 2 networks over layer 3 networks. *RFC*, 7348:1–22, 2014.

[100] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[101] Xiaoqiao Meng, Vasileios Pappas, and Li Zhang. Improving the scalability of data center networks with traffic-aware virtual machine placement. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. IEEE, 2010.

[102] Cisco Visual networking Index. Forecast and methodology, 2016-2021, white paper. *San Jose, CA, USA*, 1, 2016.

[103] Cisco Visual networking Index. Forecast and methodology, 2016-2021, white paper. *San Jose, CA, USA*, 1, 2016.

[104] Tuan Anh Nguyen, Dugki Min, Eunmi Choi, and Thang Duc Tran. Reliability and

availability evaluation for cloud data center networks using hierarchical models. *IEEE Access*, 7:9273–9313, 2019.

[105] Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya, and Amin Vahdat. Portland: a scalable fault-tolerant layer 2 data center network fabric. In *ACM SIGCOMM Computer Communication Review*, volume 39, pages 39–50. ACM, 2009.

[106] Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya, and Amin Vahdat. Portland: a scalable fault-tolerant layer 2 data center network fabric. In *Proc. of the ACM SIGCOMM conference on Data communication*, pages 39–50, 2009.

[107] V Nithin, A Rathod, V Badarla, Tim Humernbrum, and Sergei Gorlatch. Efficient load balancing for multicast traffic in data center networks using sdn. In *2018 10th International Conference on Communication Systems & Networks (COMSNETS)*, pages 113–120. IEEE, 2018.

[108] MohammadJavad NoroozOliaee, Bechir Hamdaoui, Mohsen Guizani, and Mahdi Ben Ghorbel. Online multi-resource scheduling for minimum task completion time in cloud servers. In *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 375–379. IEEE, 2014.

[109] Jonathan Perry, Amy Ousterhout, Hari Balakrishnan, Devavrat Shah, and Hans Fugal. Fastpass: A centralized zero-queue datacenter network. *ACM SIGCOMM Computer Communication Review*, 44(4):307–318, 2015.

[110] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, et al. The design and implementation of open vswitch. In *12th {USENIX} Symposium on Networked Systems Design and Implementation*, pages 117–130, 2015.

[111] Lucian Popa, Sylvia Ratnasamy, Gianluca Iannaccone, Arvind Krishnamurthy, and Ion Stoica. A cost comparison of datacenter network architectures. In *Proceedings of the 6th International COnference*, page 16. ACM, 2010.

[112] Sylvia Ratnasamy, Andrey Ermolinskiy, and Scott Shenker. Revisiting ip multicast. In *Proc. of the conf. on Applications, technologies, architectures, and protocols for computer communications*, pages 15–26, 2006.

[113] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C Snoeren. Inside the social network's (datacenter) network. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 123–137. ACM, 2015.

[114] Muhammad Shahbaz, Lalith Suresh, Jennifer Rexford, Nick Feamster, Ori Rottenstreich, and Mukesh Hira. Elmo: Source routed multicast for public clouds. In *Proceedings of the ACM Special Interest Group on Data Communication*, pages 458–471. 2019.

[115] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *Mass storage systems and technologies (MSST), 2010 IEEE 26th symposium on*, pages 1–10. Ieee, 2010.

[116] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, et al. Jupiter

rising: A decade of clos topologies and centralized control in google's datacenter network. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 183–197. ACM, 2015.

[117] Ankit Singla, Chi-Yao Hong, Lucian Popa, and Philip Brighten Godfrey. Jellyfish: Networking data centers, randomly. In *NSDI*, volume 12, pages 17–17, 2012.

[118] Hassan Sinky, Bassem Khalfi, Bechir Hamdaoui, and Ammar Rayes. Responsive content-centric delivery in large urban communication networks: A linknyc use-case. *IEEE Transactions on Wireless Communications*, 17(3):1688–1699, 2017.

[119] Hassan Sinky, Bassem Khalfi, Bechir Hamdaoui, and Ammar Rayes. Adaptive edge-centric cloud content placement for responsive smart cities. *IEEE Network*, 33(3):177–183, 2019.

[120] Guohui Wang, David G Andersen, Michael Kaminsky, Konstantina Papagiannaki, TS Ng, Michael Kozuch, and Michael Ryan. c-through: Part-time optics in data centers. In *ACM SIGCOMM Computer Communication Review*, volume 40, pages 327–338. ACM, 2010.

[121] Jianyu Wang, Jianli Pan, Flavio Esposito, Prasad Calyam, Zhicheng Yang, and Prasant Mohapatra. Edge cloud offloading algorithms: Issues, methods, and perspectives. *ACM Computing Surveys (CSUR)*, 52(1):1–23, 2019.

[122] Peng Wang, George Trimponias, Hong Xu, and Yanhui Geng. Luopan: Sampling-based load balancing in data center networks. *IEEE Transactions on Parallel and Distributed Systems*, 30(1):133–145, 2019.

[123] Ruozhou Yu, Guoliang Xue, Vishnu Teja Kilari, and Xiang Zhang. Network function virtualization in the multi-tenant cloud. *IEEE Network*, 29(3):42–47, 2015.

[124] Qiao Zhang, Vincent Liu, Hongyi Zeng, and Arvind Krishnamurthy. High-resolution measurement of data center microbursts. In *Proceedings of the 2017 Internet Measurement Conference*, pages 78–85. ACM, 2017.