Shufflecast: An Optical, Data-Rate Agnostic and Low-Power Multicast Architecture for Next-Generation Compute Clusters

AQ:1 Sushovan Das, Afsaneh Rahbar, Xinyu Crystal Wu, *Graduate Student Member, IEEE*, Zhuang Wang, Weitao Wang, Ang Chen, and T. S. Eugene Ng

***Abstract*— An optical circuit-switched network core has the potential to overcome the inherent challenges of a conventional electrical packet-switched core of today's compute clusters. As optical circuit switches (OCS) directly handle the photon beams without any optical-electrical-optical (O/E/O) conversion and packet processing, OCS-based network cores have the following desirable properties: a) agnostic to data-rate, b) negligible/zero power consumption, c) no need of transceivers, d) negligible forwarding latency, and e) no need for frequent upgrade. Unfortunately, OCS can only provide point-to-point (unicast) circuits. They do not have built-in support for one-to-many (multicast) communication, yet multicast is fundamental to a plethora of data-intensive applications running on compute clusters nowadays. In this paper, we propose Shufflecast, a novel optical network architecture for next-generation compute clusters that can support high-performance multicast satisfying all the properties of an OCS-based network core. Shufflecast leverages small fanout, inexpensive, passive optical splitters to connect the Top-of-rack (ToR) switch ports, ensuring data-rate agnostic, low-power, physical-layer multicast. We thoroughly analyze Shufflecast's highly scalable data plane, light-weight control plane, and graceful failure handling. Further, we implement a complete prototype of Shufflecast in our testbed and extensively evaluate the network. Shufflecast is more power-efficient than the state-of-the-art multicast mechanisms. Also, Shufflecast is more cost-efficient than a conventional packet-switched network. By adding Shufflecast alongside an OCS-based unicast network, an all-optical network core with the aforementioned desirable properties supporting both unicast and multicast can be realized.**

***Index Terms*— Multicast architecture, next-generation compute clusters, optical circuit-switched core, data-rate agnostic, power, capital cost.**

I. Introduction

TRADITIONAL packet-switched network cores in today's compute clusters are not sustainable in the long run as CMOS-based electrical packet switches face the challenge posed by the end of Moore's Law [11], [52]. The power consumption of the commodity Ethernet switches escalates at a faster rate compared to the switching capacity, thus hindering the free scaling for next-generation compute clusters. For example, a 400 Gbps Ethernet switch with Broadcom Tomahawk III chip and bare metal hardware has $10.8\times$ more power consumption per port than a 25 Gbps Ethernet switch with Broadcom Trident III chip and similar features. Optical circuit switching technologies seem to be the most promising alternative. The major advantages of such optical circuit-switched network cores over the electrical packet-switched counterparts are as follows: a) optical circuit switches (OCS) are agnostic to data-rate as they forward the incoming photons directly, b) OCS have negligible/zero power consumption because they are bufferless and their operating principles are simple (e.g., mirror rotation, diffraction etc.), c) there is no need for transceivers at the network core because of no optical-electrical-optical (O/E/O) conversion, d) OCS have negligible forwarding latency as they do not need packet-by-packet processing, and e) the network core does not need frequent upgrade because OCS are data-rate agnostic. As a result, designing next-generation compute cluster architectures with optical circuit-switched cores has been gaining significant momentum during recent years. Different proposals have leveraged a wide range of OCS technologies e.g., 3D/2D MEMS [38], [43], [47], [51], arrayed waveguide grating router (AWGR) [11], [63], [65], [66], free-space optics mirror assembly [30] etc.

However, unlike the packet-switched network cores that can natively support one-to-many (multicast) communication, OCS-based network cores cannot inherently multicast packets to multiple destinations. The fundamental reason is that OCS are only capable of providing point-to-point (unicast) circuit connections between source-destination pairs with some form of dynamic reconfigurability. Having no support for multicast is a serious technological gap, as data-intensive applications are on the rise in large-scale compute clusters and they heavily rely on iterative big-data multicasts. For instance, consider distributed machine learning (ML) workloads in compute clusters today. Take the LDA algorithm [15] as an example. Gigabytes of data representing the word distribution of all the sampled topics are multicasted in each algorithm iteration. Since an LDA job runs for thousands of iterations, multicast traffic volume can easily reach terabytes. Other ML examples include the Logistic Regression algorithm for Twitter spam filtering and the Alternating Least Squares algorithm for Netflix movie

AQ:2 Manuscript received April 9, 2021; revised November 4, 2021; accepted February 14, 2022; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor B. Ramamurthy. This work was supported in part by NSF under Grant CNS-1718980, Grant CNS-1801884, and Grant CNS-1815525. *(Corresponding author: Sushovan Das.)*

AQ:3 The authors are with the Department of Computer Science, Rice University, Houston, TX 77005 USA (e-mail: sd68@rice.edu).

Digital Object Identifier 10.1109/TNET.2022.3158899

rating prediction [25]. Both jobs take hundreds of iterations, and multicast communications account for 30% and 45% of the job completion time, respectively. Next, consider high performance computing (HPC) workloads which include various scientific data analysis jobs [29], [34], [60]. Those applications perform iterative multicasts using MPI_Bcast [8], which is a primitive in the MPI framework for one-to-many message passing. Consider also data mining workloads (e.g., Apache Hive [54], Spark SQL [24]). In such workloads, one of the most critical and time-consuming operations is the distributed database join, in which one of the input tables is multicasted to all workers. These tables are up to 6.2 GB in a popular database benchmark [3].

Hence we believe, enabling high-performance multicast for next-generation compute clusters while preserving all the properties of OCS-based network core is the most necessary next step, as it will provide a crucial missing piece of the all-optical circuit-switched network puzzle. However, conventional solutions are not enough. On one hand, application-level peer-to-peer overlays on OCS-based cores would be a zero capital-cost solution, but it would suffer from poor multicast performance and high power consumption due to redundant data transmission. On the other hand, network-level multicast (a.k.a. IP-multicast) on a separate packet-switched core (complementing the OCS-based unicast-capable core), despite achieving ideal multicast performance, won't satisfy any of the OCS properties.

Passive optical splitter is a potentially adoptable technology which supports data-rate agnostic physical-layer multicast satisfying all the properties of OCS-based network core. However, designing a cluster-wide multicast capable network using optical splitters is not straightforward. A single giant splitter cannot span across all the ToRs to provide a cluster-wide multicast tree, because the insertion loss of a splitter proportionally increases with its fanout. No optical transceiver would be able to compensate such high insertion loss of that giant splitter. Also, splitter cannot make smart forwarding decisions when necessary, due to lack of software control.

We present a novel optical architecture called Shufflecast to support high performance multicast in next-generation compute clusters, which complements any unicast capable OCS-based network cores and preserves all the properties. Shufflecast has a unique optical-splitter topology which can scale to arbitrary network size even using small fanout splitters, ensuring data-rate agnostic multicast at scale. We show that ToR-to-ToR-level routing on Shufflecast can be static, yet such simplicity in routing still optimally exploits the topology and enables multiple one-to-all multicast to happen simultaneously at line-rate. Moreover, such static nature of routing eliminates the need for runtime ToR-to-ToR-level tree construction, group state exists only at the network edge; which makes its control plane light-weight. Shufflecast is robust enough against single relay failure. We design a failure recovery algorithm which completely restores the reachability with graceful performance degradation. Finally, we develop a prototype implementation of Shufflecast and perform comprehensive testbed evaluation. We demonstrate that Shufflecast is up to $1.77\times$ more power-efficient compared to a peer-to-peer overlay on an OCS-based unicast network core. Also, Shufflecast is up to $1.85\times$ more power-efficient and $1.89\times$ more cost-efficient compared to IP-multicast on a minimal-layer packet-switched network core. Shufflecast ensures high physical-layer reliability and works well with existing transport layer protocols. Furthermore, we show that real-world high-throughput and low-latency applications can leverage and benefit from Shufflecast with only minor modifications.

II. Motivation

A. Advantages of OCS-Based Network Core

The fundamental properties of OCS-based network cores are: a) data-rate agnostic nature, b) negligible/zero power consumption, c) no need of transceivers, d) negligible forwarding latency, and e) no need for frequent upgrade. OCS are agnostic to data-rate because they direct the incoming photon beams across predefined circuits irrespective of the modulation rate of the electronic signal. OCS intrinsically have negligible or zero power consumption due to their operating principles. For example, MEMS-based OCS consume very little power just to drive the DSP circuitry used for rotating the mirrors to setup the circuits among input/output ports. As another example, AWGR switches are fully passive (i.e., consumes no power) as they perform wavelength routing of the optical signals across the predefined input/output ports based on diffraction grating. As OCS deal with photons, they do not need optical transceivers for O/E/O conversion. As a consequence, OCS do not need any electronic data processing or buffering which leads to negligible forwarding latency. Due to the data-rate agnostic property and absence of transceivers, the OCS-based network cores need not be replaced even as the network edge (ToRs and servers) is upgraded to higher speeds. Finally, the combination of all these aspects results in OCS-based network cores to be sustainable in the long run, while achieving close to non-blocking network performance for point-to-point (unicast) communication. Hence, there is a major momentum shift towards building such OCS-based cores for next-generation compute cluster architectures [11], [30], [38], [43], [51], [65].

B. OCS-Based Network Core Lacks of Multicast Capability

Unlike the packet-switches, OCS are not capable of supporting point-to-multipoint (multicast) connectivity. However, distributed ML/HPC/database applications are dominating workloads in today's compute clusters and such applications heavily rely on multicast. Hence, there is an urgent need for the next-generation compute clusters to support high performance multicast while preserving all the properties of OCS-based network core. Under these circumstances, the easiest approach would be to deploy the application-level peer-to-peer overlay on OCS-based cores. Here, the application organizes its processes into an overlay network and the peers distribute multicast messages as TCP-based unicast flows [12], [21], [22], [27], [33], [35], [55]. Despite being a zero capital-cost solution with easy deployability, peer-to-peer overlay-based multicast suffers from bandwidth inefficiency because of significant data packet duplication at the end hosts and high control overhead. Such high data redundancy leads to non-negligible link stress

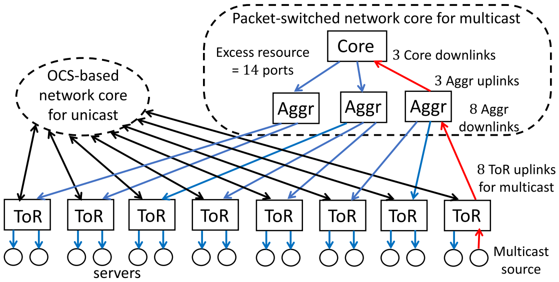


Fig. 1. A "hybrid" network architecture: OCS-based core serves unicast and a separate packet-switched network core serves multicast traffic. The minimal-layer packet-switched network core requires 175% excess resource to support one cluster-wide multicast tree across eight ToR (4-port) switches.

(e.g., $1.9 - 10\times$) which becomes worse with large multicast group size [18], [21], [26]. Even when very carefully optimized by experts, redundancy is still at 39% [18]. Additionally, application layer overlays can lead to unpredictable latency fluctuation in relay server performance with large multicast group size [14]. Based on our experiments, overlay multicast in state-of-the-art frameworks like MPI [8] and Spark [25], can be $3 - 5.7\times$ slower than optimal (see section VI-A). Overlay-based multicast also suffers from high power consumption due to redundant data transmission.

Therefore, enabling high performance multicast in next-generation compute clusters while preserving all the properties of OCS-based network core is challenging. Conventionally we could imagine a "hybrid" network architecture, where OCS-based network core serves the unicast traffic and a separate hierarchical packet-switched network core serves the multicast traffic exclusively. Such a packet-switched network core would preserve the ideal multicast performance, as the packet-switches can inherently support IP-multicast forwarding without any data redundancy. However, it would violate all the OCS properties, as packet switches are not agnostic to data-rate; they have high power consumption; they need transceivers, packet-by-packet processing and short-term upgrade. Moreover, such a network would have high capital cost. Even constructing a minimal layer packet-switched network core using identical port-count packet switches (same as ToR switches) would require non-trivial amount of electronics. To quantify such effect, we define a metric "excess resource usage" which is the ratio of extra switch ports to total ToR uplink ports, expressed in percentage. As an illustrative example, consider a simple cluster with eight 4-port ToR switches shown in Fig. 1. To support a one-to-all multicast tree using a minimal-layer packet-switched network core, we need 14 extra switch ports apart from 8 uplink ToR ports, leading to 175% excess resource usage. Similarly, a cluster with 192 32-port ToR switches require at least 107% excess resources to enable a one-to-all multicast. Hence, deploying such a network will not be sustainable in the long run.

C. Explore Optical Splitter Technology

Fortunately there exists optical splitters, an alternative technology to enable high performance multicast without data duplication. Optical splitter is a small passive device that splits the incoming optical signal from one input fiber to multiple output fibers (defined as fanout), thus providing built-in physical layer support for line-rate multicast. Additionally, optical splitter satisfies all the properties of OCS i.e., agnostic to data rate as it has no electronic processing, passive and no power consumption, no O/E/O conversion, bufferless and negligible latency, long term sustainable and no frequent upgrade. Furthermore, splitters are inexpensive and commercially available [6].

But, designing a low-diameter yet cluster-wide scalable multicast capable architecture is still an open problem, as making use of splitters have several difficulties. Naïvely we could use one giant splitter to directly join all the ToRs in a cluster consuming one transceiver port from each. Such a design is unrealistic and practically infeasible because the insertion loss (in absolute scale) of a splitter increases proportionally with bigger fanout. Empirically, the insertion loss (in log scale) of a splitter with fanout p is given by $0.8 + 3.4 \log_2 p$ dB. Hence, a compute cluster with 1024 ToRs would require a giant splitter of fanout 1024, having insertion loss of 34.8 dB. Such high insertion loss cannot be compensated by any commercially available optical transceiver. A high-gain optical amplifier would be able to compensate such loss, but at the cost of higher power consumption, higher capital cost [10] and lower signal-to-noise ratio (SNR) at the receiver. Hence, such a network has limited scalability. Moreover, splitter is a dumb device, i.e., it does not have the ability to make smart decisions e.g., configure the multicast trees for different sources, redirect the traffic during failure etc.

We design Shufflecast, a highly scalable and low-diameter multicast-capable optical network architecture for next-generation compute clusters, which leverages small fanout passive optical splitters to connect the ToR ports. Thus, Shufflecast provides high performance multicast, while preserving all the OCS properties. By supporting multicast and complementing the unicast capable OCS-based network core, Shufflecast is a crucial component in the all-optical network core puzzle. In the next sections, we will show the following advantages of Shufflecast:

a) Shufflecast's data plane achieves high scalablility with low network diameter (Sec. III-A) using small fanout splitters, ensuring data-rate agnostic multicast. As Shufflecast can scale with limited number of ToR ports, it has low capital cost (Sec. VI-A)
b) The optimal ToR-to-ToR-level routing over Shufflecast (Sec. III-A) supports simultaneous one-to-all multicasts at line-rate. Also, Shufflecast is power efficient compared to the conventional multicast solutions (Sec. VI-A).
c) As the routing is static, ToR-to-ToR-level multicast tree construction at runtime is not necessary; group state exists only at the network edge. Hence, the control plane of Shufflecast is very simple and light-weight (Sec. III-B).
d) Shufflecast provides good failure resilience and graceful performance degradation after failure recovery (Sec. III-C and VI-C).
e) Shufflecast can reliably support multiple multicast groups using existing multicast transport protocols (Sec. VI-B).

Furthermore, real-world applications can benefit from Shufflecast with minor modifications (Sec. VI-D).

III. Shufflecast Architecture

In this section, we discuss the Shufflecast architecture in detail with data plane design, control plane design and failure handling.

A. Data Plane

In the Shufflecast data plane, passive optical splitters provide direct ToR-to-ToR connectivity. The optical transceivers and splitters are co-located at the ToRs without consuming extra rack space.

1) Topology: The Shufflecast topology is parameterized by p and k, where p denotes the number of ToRs that a single ToR connects to via a splitter, and k is the number of *logical* ToR columns in the topology. In general, a p, k-Shufflecast has $N = k \cdot p^k$ ToR switches forming a p-regular graph, with each column having p^k ToRs. Fig. 2 shows an example of $2, 2$-Shufflecast, where there are 8 ToRs arranged in 2 columns, with 4 ToRs per column and each ToR equipped with 1:2 optical splitter (nodal degree 2). More examples are in Appendix A-A1. Note that Shufflecast can also accommodate an arbitrary number of ToRs. Assume the total number of ToRs $= T$, where $k_1 \cdot p^{k_1} < \mathrm{T} < k_2 \cdot p^{k_2}$. To accommodate T ToRs, the network is wired as a p, k_2-Shufflecast where a few physical ToRs would act as additional logical nodes to maintain the connectivity pattern. Such a strategy would require more than one splitter at those ToR switches. For example, consider $T = 15$; the nearest Shufflecast instance for $p = 2$ is a $2, 3$-Shufflecast (24 ToRs), where 9 ToRs would be assigned two splitters and act as two distinct logical sources.

Logical ToR ID: We realize the Shufflecast topology using IP-based L2/L3 Ethernet switches. The "logical" ToR IDs are defined to explain the properties of the topology and the routing scheme. In a p, k-Shufflecast, the columns (c) are numbered as $0, 1 \ldots (k-1)$ from left to right, and the rows (r) are numbered as $0, 1 \ldots (p^k - 1)$ from top to bottom. Any ToR with a decimal representation 'i' ($i \in [0, N-1]$) is uniquely identified by the pair $\left(c^i, r^i\right)$ where column ID (c^i) is $\lfloor \frac{i}{p^k} \rfloor$ and row ID (r^i) denotes the k-tuple p-ary representation of ($i \bmod p^k$) given by $[r^i_{k-1} r^i_{k-2} \ldots r^i_1 r^i_0]$. For $2, 2$-Shufflecast shown in Fig. 2, each ToR has a binary 2-digit row ID $r_1 r_0$. Considering any ToR switch e.g., ToR 6, its column ID is $\lfloor \frac{6}{2^2} \rfloor = 1$ and row ID is the binary representation of $(6 \bmod 2^2) = 2$, i.e., 10, resulting in a combined ID $(1, 10)$.

ToR connectivity: We can further define the ToR connectivity pattern of Shufflecast topology using such logical IDs. Any ToR $(c^i, r^i_{k-1} r^i_{k-2} \ldots r^i_1 r^i_0)$ is connected to p other ToRs of the next column ($c^j = \left(c^i + 1\right) \bmod k$), having the row IDs as 1 place left-shift of its own row-ID digits with the least significant digit $m \in [0, p-1]$(i.e., $r^j = [r^i_{k-2} r^i_{k-3} \ldots r^i_0 m]$).

Partition: We *logically* partition the columns into p regions based on the logical ToR IDs. The partition ID of each ToR is defined by the most significant digit of the ToR's p-ary row ID (i.e., $r_{k-1} \in [0, p-1]$). For the $2, 2$-Shufflecast in Fig. 2, every column has two partitions with partition IDs 0

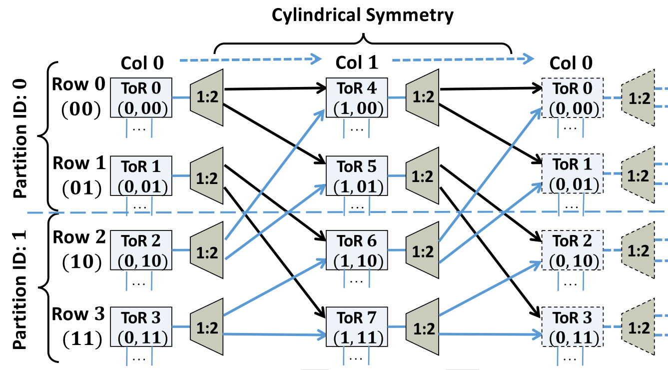


Fig. 2. Connectivity of $2, 2$-Shufflecast.

(ToRs $\{0, 1\}$ and $\{4, 5\}$) and 1 (ToRs $\{2, 3\}$ and $\{6, 7\}$). All the outgoing links from partition ID 0 are marked with darker arrows and those from partition ID 1 are marked with lighter arrows. The notion of partition has two important properties. a) A *logical* partition refers to an independent resource unit (i.e., subset of relays) of Shufflecast topology, which is evident from the connectivity structure. In general, a partition containing p^{k-1} ToRs is sufficient to forward the multicast traffic to all the p^k ToRs of next column. b) The number of partitions in a given column dictates the degree of parallelism for Shufflecast topology. Because, the relays from different partitions of a given column can forward multicast traffic in parallel without any interference. In Sec. III-A3, we discuss the ToR-to-ToR-level routing scheme, which cleverly exploits such parallelism of Shufflecast topology to support multiple one-to-all multicasts simultaneously at line-rate.

2) Topological Properties: The unique topology of Shufflecast has some highly desirable properties such as high scalability and bounded latency.

Scalability and port counts: Shufflecast topology can scale to an arbitrary network size ($N = kp^k$) with small splitter fan-out (p), by increasing the parameter k (independent of power-splitting limitations). The number of columns scales linearly (k) and the number of rows scales exponentially (p^k). At first glance, each ToR needs 1 transmit and p receive ports. However, one transmit and one receive port can be simultaneously handled by one transceiver in practice, which leads to p transceiver ports consumed per ToR. For example, a $2, 2$-Shufflecast can accommodate 8 ToRs. Similarly, a $2, 3$-Shufflecast (Fig. 9 in A-A1) scales to 24 ToRs. Both these instances only require 2 transceiver ports per ToR.

Hop counts: Leveraging the topological properties of Shufflecast, routing can be performed with low worst-case hop count ($\propto \log_p N \approx k$).

Lemma 1: For a p, k-Shufflecast all the ToRs are reachable from a given source by at most $2k - 1$ hops.

Intuitively, we generate the multicast tree along the splitter-based connectivity from any given source ToR, and all other ToRs can be reached from the source column within two complete traversals. For example, in $2, 2$-Shufflecast of figure 2, multicast packets from ToR 0 can reach ToR 4 and 5 in 1^{st} hop. At 2^{nd} hop, ToR 4 relays these packets to ToR 1, and ToR 5 relays to ToRs 2 and 3. During the second traversal,

Algorithm 1 Next-Hop Relay Computation Algorithm

1: $src = c^s,\ r^s_{k-1}\ldots r^s_1 r^s_0,\ dest = c^d,\ r^d_{k-1}\ldots r^d_1 r^d_0$
2: $cur = c',\ r'_{k-1}\ldots r'_1 r'_0$
3: **if** $c^d == c'$ **then**
4: $\quad X = k$
5: **else**
6: $\quad X = \left(k + c^d - c'\right) \bmod k$
7: **end if**
8: **if** $(X == k)$ OR $(r^d_{k-1}\ldots r^d_X == r'_{k-X-1}\ldots r'_0)$ **then**
9: $\quad next = (c'+1) \bmod k,\ r'_{k-2}\ldots r'_0 r^d_{X-1}$
10: **else**
11: $\quad X' = (k + c' - c^s) \bmod k$
12: $\quad next = (c'+1) \bmod k,\ r'_{k-2}\ldots r'_0 r^s_{k-X'-1}$
13: **end if**

Source ToR	Route to all other ToRs	Relay set for one-to-all multicast
0	**0→4→1 , 0→5→2 , 0→5→3 , 0→4 , 0→5 , 0→4→1→6 , 0→4→1→7**	**{ 0 , 1 , 4 , 5 }**
3	**3→6→0 , 3→6→1 , 3→7→2 , 3→7→2→4 , 3→7→2→5 , 3→6 , 3→7**	**{ 2 , 3 , 6 , 7 }**

Fig. 3. Relay sets for ToR-0 and ToR-3 in $2,2$-Shufflecast.

either of ToR 1 or 3 can relay the packets to ToRs 6 and 7 in 3^{rd} hop. Therefore, the maximum hop count is 3. A proof is given in Appendix A-A3.

3) Multicast-Aware Routing: To multicast packets from a source, every ToR along the path needs to know whether packets should be relayed via its optical splitter. Our multicast-aware routing provides static ToR-to-ToR-level relaying rules that depend only on the source ToR ID, without needing runtime switch reconfigurations. Separately, ToR-to-server forwarding is dynamically configured based on the multicast group as needed by the applications.

The objective of the multicast-aware routing is to maximize the utilization of disjoint one-to-all multicast trees exploiting the degree of parallelism of the Shufflecast topology. Algorithm 1 illustrates next-hop relay computation. It takes the source (src), destination ($dest$) and current (cur: initialized to src) ToR IDs as input (lines 1 and 2), and computes the next-hop ($next$) ToR ID which acts as the relay for routing packets from that source towards the given destination. At a high level, the algorithm determines whether the destination ToR is reachable from the source ToR during the first traversal or second traversal cycle. Accordingly, it finds the next-hop ($next$) ToR ID by shifting the current ToR's row-ID to the left by one digit; and putting pre-calculated row-ID digit from either destination or source ToR ID as a least significant digit (lines 9 and 12). As shown in Fig. 3, we calculate all routes and relay sets for multicast sources ToR-0 $(0, 00)$ and ToR-3 $(0, 11)$ of $2,2$-Shufflecast using Algorithm 1.

The routing algorithm enables any source ToR to perform one-to-all multicast while choosing the relays from each column in a compact manner. More specifically, a given source ToR uses the subset of relay ToRs from each column which belong to the partition IDs defined by the source row-ID digits, termed as *partition criteria*. Such selective inclusion of relays ensures the maximal utilization of the Shufflecast topology, which we generalize in the next section. As shown in Fig. 3, ToR $0(0, 00)$ and ToR $3(0, 11)$ in $2,2$-Shufflecast relay through partition IDs 0 (ToRs $0, 1$ and $4, 5$) and partition IDs 1 (ToRs $2, 3$ and $6, 7$) of both the columns respectively, maintaining the partition criteria. As a consequence, ToRs 0 and 3 have disjoint relay sets and they can perform one-to-all multicasts simultaneously at line-rate.

4) Routing Properties: Shufflecast has the ability to exploit all degrees of network parallelism, with careful choices of relay ToRs, enabling high multicast performance. Next, we formally state the properties of multicast-aware routing with high level insights. All the proofs are in Appendix A-A3.

Lemma 2: Using the multicast-aware routing for a p,k-Shufflecast, any given source ToR can perform one-to-all multicast following the partition criteria i.e., using the relays from each column belonging to the partition IDs predefined by its k row-ID digits.

The intuition is from the construction of next-hop relay computation algorithm. For computing the next-hop relay ToR ID, the algorithm 1 carefully uses pre-calculated source or destination ToR row-ID digits. Eventually, those source ToR row-ID digits govern the partition for choosing the relays.

Lemma 3: Using the multicast-aware routing for a p,k-Shufflecast, p ToRs in one column can perform one-to-all multicasts simultaneously at line-rate, $2p$ ToRs at half of line-rate, $3p$ ToRs at one-third of line-rate, and all p^k ToRs in one column at p^{k-1} fraction of line-rate.

The result is directly obtainable from Lemma 2 and the definition of partition (Sec. III-A1). Multicast-aware routing effectively exploits all degrees of network parallelism.

Lemma 4: Multicast-aware routing is optimal in terms of minimizing the relay usage and maximizing the number of one-to-all simultaneous multicast at line-rate.

The first part of this lemma is directly obtainable from Lemma 2 and properties of partition discussed in Sec. III-A1. Any given source ToR uses one partition of relays from each column by multicast-aware routing, which indeed is the minimum number of ToRs required to reach all the ToRs in the next column. Further, the second part of this lemma is obtainable by extending this intuition along with Lemma 3.

B. Control Plane

We assume that ToR switches support direct control of forwarding rules (e.g., OpenFlow or P4 switches). These switches identify and forward the multicast packets sent by applications (IP datagrams with Class D destination addresses).

1) Static ToR-to-ToR Relaying: For a given instance of Shufflecast, we need to apply the relay computation algorithm for each multicast source ToR once to obtain the list of relays on the routes towards all destination ToRs. Then we insert one forwarding rule on these relay switches in regard to that source ToR. With these relay forwarding rules, data can flow from a source to all other ToRs through the designated relays. As the forwarding rules can be precomputed, they can be pre-installed on the ToR switches, eliminating the need for computing routes at runtime. Moreover, the number of such fixed rules are not significant compared to the memory capacity of modern switches. As discussed in Sec. III-A1, each ToR in a p, k-Shufflecast needs to install kp^{k-1} fixed forwarding rules as it relays multicast packets for kp^{k-1} source ToRs. For example, a $4, 4$-Shufflecast covering 1024 ToRs needs only 256 static forwarding rules to install on each ToR where the modern OpenFlow-based SDN switches can accommodate more than 10k rules. Hence, the scheme is highly scalable.

2) Application-Directed ToR-to-Server Forwarding: We enable dynamic ToR-to-server forwarding rule update based on application defined multicast server group membership. All the ToRs are managed by a logically centralized controller. The application interacts with the switches via the controller. When the application starts, one of its processes proactively sends the multicast group membership configuration request to the controller and waits for its response. Then the controller identifies the active servers (of that multicast group) under each ToR switch, converts them into corresponding multicast rules (capable of forwarding incoming packets to multiple ports simultaneously) and install those rules on the switches. Finally the application proceeds after getting the acknowledgement from the controller. By doing so, multicast data is confined to only the servers who belong to the respective multicast group defined by the application, which avoids unnecessary contention.

C. Failure Handling

Fault tolerance is another important consideration for architecture design. Next, we discuss data and control plane failure handling of Shufflecast in detail.

1) Data Plane Failure Handling: The primary sources of the Shufflecast data plane failure are bad optical transceiver, bent fiber, damaged splitter and dirty connector [67]. We consider any such component failure as a complete failure of the associated relay. We discuss the performance impact of single relay failure and our re-routing algorithm to get around such a failure, as correlated multiple relay failures would be relatively rare.

Reachability impact of single relay failure: First we model the reachability impact of single relay failure on p, k-Shufflecast. Fig. 4 illustrates different reachability scenarios for an example case and provides the intuition to formulate the general case. Consider when ToR relay number 8 fails in a $2, 3$-Shufflecast (Fig. 9 in A-A1). As shown in Fig. 4, there are six configurations ((a)-(f)) showing unique locations of the failed relay 8 on one-to-all multicast trees of different source ToRs. All these multicast trees have similar structure; a major spine consisting of three (i.e., k) ToRs with source ToR as the root and one perfect binary (i.e., $p = 2$) subtree (defined as *islands*) of height three (i.e., k), hanging from each ToR in the spine. As we vary the source ToR, the location of the failed relay on the multicast tree varies (24 different locations for 24 possible sources) and correspondingly that leads to one of these six configurations along with certain number of unreachable ToRs.

Configuration (a) shows the case where the failed relay 8 is a leaf in island 1, i.e., ToR 8 does not relay the multicast packet for that source and there are 12 such leaf locations across three

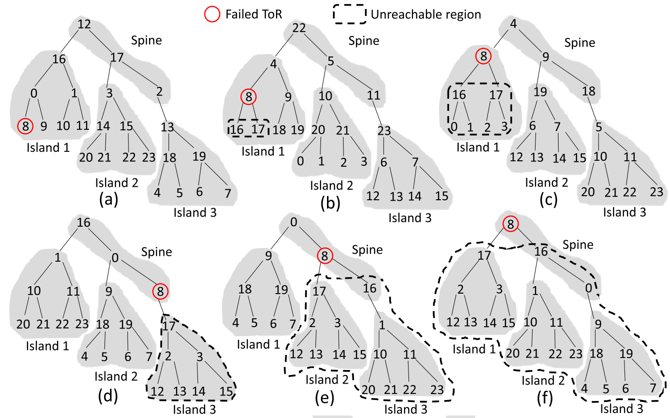


Fig. 4. Different reachability scenarios when relay 8 fails in a $2, 3$-Shufflecast. configurations (a)-(f) illustrates the unique locations of the failed relay (i.e. ToR 8) on one-to-all multicast trees considering different source ToRs.

islands. Hence, there are 12 source ToRs for which there will be no impact on reachability if relay 8 fails. In configuration (b), the failed relay 8 is located at one-level above the leaf in island 1, i.e., ToR 8 relays the multicast packet to two (i.e., p) other non-relay ToRs (leafs). As there are 6 such possible locations across the three islands, there exists 6 source ToRs which can't send multicast data to 2 leaf ToRs (marked with dashed contour) if the relay 8 fails. Similarly in configuration (c), the failed relay 8 is the root of island 1. Hence the number of unreachable ToRs is 6 (i.e., $p + p^2$) and 3 source ToRs will have such impact, as there are 3 such equivalent locations across the islands.

Next, in configurations (d)-(f), the failed relay 8 is located on the major spine of the multicast tree. As these locations are unique, there is a unique source associated with each of these cases. Specifically in configuration (d), the source ToR is 16 and the failed relay 8 is at the lowest level of the spine. Hence, ToR 16 can't send multicast data to all 7 (i.e., $p^k - 1$) ToRs in island 3. Similarly in configuration (e), all the ToRs in island 2 and 3 along with the lowest relay of the spine (i.e., total $2p^k - 1 = 15$) are unreachable from the source ToR 0. Finally, configuration (f) shows the trivial case where failed relay 8 is the source i.e., root of the multicast tree. Hence, all $kp^k - 1 = 23$ other ToRs are unreachable from ToR 8. Extending this idea, we compute the distribution of reachability impact of single relay failure on p, k-Shufflecast, which we further evaluate in Sec. VI-C.

Single relay failure recovery: For p, k-Shufflecast, a given ToR in any column is directly connected from p ToR relays (one from each partition) of the previous column. For example, in $2, 3$-Shufflecast (Fig. 9 in A-A1), ToR relays $0\,(0, 000)$ and $4\,(0, 100)$ are situated at 0^{th} location of partition IDs 0 and 1, respectively, and both are connected to ToR $8\,(1, 000)$. We define these ToR relays as "mirrored relays," where their row-ID digits are the same except the most significant digit which dictates the partition. Note that there exist more than one path to reach a set of ToRs from a given source, allowing Shufflecast to reroute packets upon relay failure. Algorithm 2 shows how to handle a single relay failure for p, k-Shufflecast. Depending on the failed relay ToR ID, we need to deactivate

Algorithm 2 Single Relay Failure Recovery Algorithm

1: $failed_{relay} = c,\ r_{k-1}r_{k-2}\ldots r_1r_0$
2: $y = (r_{k-1} + 1) \bmod p$
3: $mirror_{failed} = c,\ yr_{k-2}\ldots r_1r_0$
4: Deactivate all relaying rules on $failed_{relay}$
5: Activate all $failed_{relay}$ rules on $mirror_{failed}$
6: $precedent_{relay} = (c-1) \bmod k,\ r_0y\ldots r_2r_1$
7: $y' = (r_0 + 1) \bmod p$
8: $mirror_{precedent} = (c-1) \bmod k,\ y'y\ldots r_2r_1$
9: **for** $i \leftarrow 1$ to $(k-1)$ **do**
10: $\quad n_i = (c-i) \bmod k,\ r_{i-1}\ldots r_0r_{k-1}\ldots r_i$ ▷ *Circular right shift of $failed_{relay}$ row ID by i positions*
11: $\quad$ Deactivate relaying for n_i on $precedent_{relay}$
12: $\quad$ Activate relaying for n_i on $mirror_{precedent}$
13: **end for**

some relaying rules on two specific ToR relays (including the failed relay) and activate those on two other ToR relays, regardless of network size.

We explain the algorithm using the example below. Consider a $2,3$-Shufflecast, where relay $8\,(1,000)$ fails ($failed_{relay}$) and source $0\,(0,000)$ needs to perform one-to-all multicast. Based on Algorithm 2, the four specific ToRs are marked in Fig. 5, for which the relay rules will be affected. All relaying rules on failed relay 8 are deactivated and its mirrored relay $12\,(1,100)$ ($mirror_{failed}$) activates those rules on its behalf (lines 1-5). Additionally, the precedent relay $2\,(0,010)$ ($precedent_{relay}$) deactivates the relaying rules of a subset of source ToRs and it's mirrored relay $6\,(0,110)$ ($mirror_{precedent}$) activates those rules (lines 6-12).

Note that, only activating the relay rules on $mirror_{failed}$ on behalf of $failed_{relay}$ is not enough. Because, after the first traversal cycle through all the columns, packets from ToR 0 can only reach to the ToRs of partition ID 1 (ToRs $4,5,6$ and 7) at its own column; the ToRs from its own partition ID 0 (i.e., ToRs $1,2$ and 3) have not received them yet. Unfortunately, none of those relays from partition ID 1 can forward the packets as per the routing rule. Similar situation happens for source $16\,(2,000)$ too. Specifically, the relay 12 ($mirror_{failed}$) cannot get the packets from its designated precedent relay 2 ($precedent_{relay}$). Hence, relaying of source 0 and 16 (n_1 and n_2 respectively, at line 10 inside the loop) are deactivated on relay 2, while relay 6 ($mirror_{precedent}$) activates those rules on its behalf. Now, ToR 0 can successfully perform one-to-all multicast, where the outgoing links from newly activated relays are marked with darker arrows and all other required links are marked with lighter arrows. Thus, Shufflecast can recover 100% reachability from a relay failure (except for the servers under the ToR of the failed relay can no longer be multicast sources) by re-routing packets. Moreover, such failure recovery results in graceful performance degradation, evaluated in Sec. VI-C.

Note that our single relay failure recovery algorithm is general enough to handle many concurrent failures. Each relay failure is treated independently and the algorithm turns on and off appropriate relays accordingly. In general, a p,k-Shufflecast can always handle any concurrent

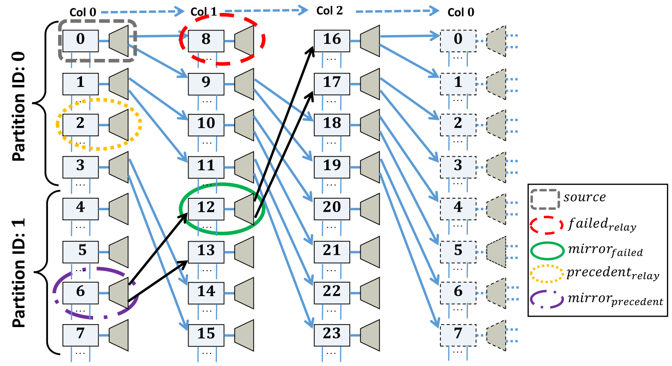


Fig. 5. Failure recovery in a $2,3$-Shufflecast when ToR 0 needs to make a one-to-all multicast while ToR 8 fails.

failure involving less than p ToRs. However, the reachability may not be restored for some failures involving p or more relays if all the p mirrored relays in one column fail. The probability of such a failure event involving p relays is $\frac{k\cdot p^{k-1}}{\binom{k\cdot p^k}{p}}$, which decreases rapidly with the size of the network.

2) Control Plane Failure Handling: Controller failure does not affect ToR-to-ToR forwarding in Shufflecast, as those relaying rules are static and pre-installed offline. However, it affects the server-level multicast group membership configuration, as Shufflecast still needs dynamic application-directed ToR-to-server forwarding update at runtime. To handle such controller failure, the logically centralized controller can be realized as a small cluster of controllers, where one can act as primary controller and others can be as backup controllers. When the primary controller fails, a backup controller can be elected as the leader, which can be used by the application for runtime switch configuration.

IV. Discussions

In this section, we discuss several practical advantages in the Shufflecast architecture.

A. Leveraging Idle Edge Bandwidth

Shufflecast can potentially leverage idle edge bandwidth, as often there exists unused switch ports at ToRs due to design constraints on space, power, and network oversubscription. This observation is first made by recent works [20], [23], [45] and confirmed by large network operators we consulted. Additionally, we conduct an analysis to quantify the likelihood of unused ToR ports (details in Appendix A-B). We consider a wide range of network configurations. The results show that unused ports, as well as a large amount of unused bandwidth, often exist. The existence of 2+ unused ports and 100 Gbps of unused bandwidth can be seen in nearly 79% and 73% of the cases, respectively. Under 1:1 oversubscription (o/s), 54% of cases have at least 10 unused ports and 500 Gbps of unused bandwidth. We also observe that the likelihood of having unused ports do not correlate with o/s ratios, rack sizes, and server port speeds etc., indicating that unused ports can exist throughout the continuum of configuration choices.

B. Simplifying Network Management

Shufflecast incurs very little need for runtime switch configurations as it uses static optimal ToR-level routing rules. Except for the forwarding behaviors to end hosts at the ToRs, all ToR-to-ToR forwarding rules are precomputed and preinstalled on switches. These preconfigurable and static switch actions make Shufflecast much less prone to configuration errors, which is the primary source of network management complexities. In addition, the physical wiring of Shufflecast is easy to deploy. For a p, k-Shufflecast topology, the optical transceivers and splitters are co-located at the ToRs, meaning that we only need to install p incoming and outgoing optical fiber cables. In terms of wiring, the mapping from the logical ToRs to physical ToR locations is based on the logical columnwise placement, bundling fibers across partitions. Also, most physical wiring is between the adjacent physical rows of racks, and the length of fibers would not incur significant attenuation (0.36 dB/km at 1310 nm [6]).

C. End-to-End Reliability

Shufflecast is dedicated to multicast traffic and leverages optical splitters to enable physical-layer multicast. Below we concretely argue how Shufflecast can ensure reliability from different aspects.

a) Physical layer reliability: Typically, the chances of packet loss in the optical devices are extremely rare. The optical transceivers have bit-error rate less than 10^{-12}. Even though passive optical splitters have insertion loss, the optical link can be made completely lossless when choosing compatible optical transceivers with a feasible power budget (Table I). Moreover, as shown in Sec. III-C, Shufflecast can gracefully handle and reroute traffic in presence of single relay failure. Hence, Shufflecast has inherent physical layer reliability.

b) Higher layer reliability: In presence of multiple applications, the occasional packet losses in Shufflecast links can be handled by transport layer solutions such as NORM [4], an off-the-shelf reliable multicast protocol enabled with congestion control [58], [59]. As shown in Sec. VI-B, Shufflecast can handle concurrent multicast applications using NORM with high reliability. Additionally, multiple applications can also coordinate based on the explicit knowledge of the topology, static relaying pattern and design capacity of Shufflecast network. For example, two applications can inject multicast traffic simultaneously at line-rate if they use disjoint partitions of Shufflecast; otherwise, they can take turn at line-rate based on their arrival time (FCFS) if they have common relays, thus maximizing the network utilization and minimizing packet losses between ToR-to-ToR links.

V. Implementation

We implement a prototype of $2, 2$-Shufflecast in our testbed. Our setup uses 3 OpenFlow switches, 8 optical splitters (1:2), and 16 servers. We divide logically 2 OpenFlow switches to emulate 4 ToR switches each, and 2 servers are connected to each logical ToR. We wire the Shufflecast network using optical splitters on these 8 logical ToR switches. The 3rd OpenFlow switch is used for comparative evaluation, it connects to the logical ToRs, creating a 2-layer full-bisection bandwidth network across ToR switches and emulating a non-blocking network core. Each server has 6 3.5GHz CPU cores with 12 hyperthreads and 128 GB RAM. All connections are 10 Gbps Ethernet. To minimize the number of ports used, while wiring the $2, 2$-Shufflecast, at each logical ToR switch we connect the outgoing fiber (to its own splitter) and one of the 2 incoming fibers (from 2 other splitters) to a single transceiver port. Thus, each logical ToR consumes only 2 transceiver ports (optimal for $2, 2$-Shufflecast). The forwarding rules are installed on the switches using the Ryu OpenFlow controller [5], running on one of the servers.

The controller program consists of two parts. The first part runs Algorithm 1 (Next-hop relay computation algorithm) and pre-installs the static ToR-to-ToR forwarding rules for $2, 2$-Shufflecast ($<$100 lines of python code). The second part translates application-based multicast group membership information into the ToR-to-server multicast rules and installs them on the switches at runtime ($<$30 lines of python code). We make simple modifications to applications to interact with the controller program ($\approx$10 lines of C++ code).

VI. Evaluation

In this section, we present comprehensive testbed experimental results to demonstrate that Shufflecast can achieve a) line-rate multicast throughput with low power consumption and capital cost, b) high end-to-end reliability while supporting concurrent multicast groups, c) high robustness against single relay failure and graceful performance degradation after failure recovery and d) improved application performance for both high-bandwidth and low-latency applications.

A. Shufflecast Achieves Line-Rate Multicast Performance With Low Power Consumption and Capital Cost

We perform experiments and analysis to evaluate the multicast performance of Shufflecast. Also, our analysis shows that Shufflecast is power and cost efficient across network scale.

a) Multicast performance of Shufflecast vs. state-of-the-art multicast mechanisms: For comparing throughput, our baseline mechanisms are state-of-the-art multicast solutions i.e., 1) peer-to-peer mechanisms such as MPI_Bcast [8] and Spark-Cornet [25] and 2) IP-multicast. For both the baselines, we use full-bisection bandwidth network to measure their ideal maximal performance.

We perform a 1:15 multicast with varying data size (from 200 MB to 1.4 GB) and measure the multicast reading time (i.e. the duration between receiving program issues reading request and finishes reading it). Fig. 6(a) shows the multicast throughput (averaged over 10 runs) defined as the ratio of multicast data size to multicast reading time. We observe that Shufflecast achieves line-rate multicast throughput, same as the upper-bound performance of IP-multicast (over full-bisection bandwidth network), irrespective of the multicast group size. We also observe that, even without any competing traffic on full-bisection bandwidth network, both MPI_Bcast and Spark-Cornet achieve the multicast throughput only upto

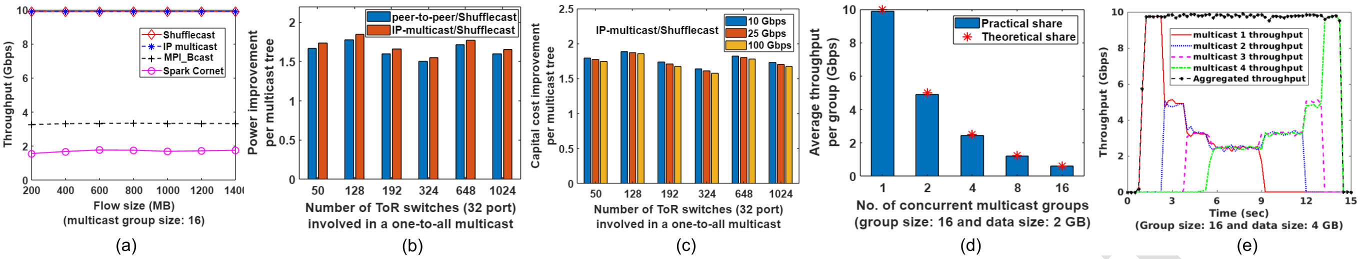


Fig. 6. (a) Throughput (averaged over 10 runs) for different multicast mechanisms (Shufflecast, IP-multicast and peer-to-peer overlay) across different data size for a 1:15 multicast flow. Shufflecast achieves the line-rate multicast performance, (b) Improvement in power consumption (ratio of baseline power consumption to that of Shufflecast) per one-to-all multicast tree of Shufflecast compared to peer-to-peer overlay (on optical circuit-switched network core) and IP-multicast (on minimal layer packet-switched network core) with scale. The improvement factor is the same across different data rates (10 Gbps, 25 Gbps, 100 Gbps), as it only depends on the relative count of switch ports (same as transceivers), (c) Improvement in capital cost per (ratio of baseline capital cost to that of Shufflecast) one-to-all multicast tree of Shufflecast compared to IP-multicast (on minimal layer packet-switched network core) with scale at different data rates (10 Gbps, 25 Gbps, 100 Gbps), (d) Practical and theoretical average multicast throughput per group with varying number of concurrent multicast groups on Shufflecast, (e) Throughput (averaged over 10 runs) of multicast flows launched in a staggered way on Shufflecast.

35% and 20% of the line-rate throughput across data size, which is far from optimal.

In Spark-Cornet, a node first locates a block of data it needs from another node then performs a block transfer. We observe that although each individual block transfer can reach near line-rate throughput, far more time is taken up by control communications to locate and wait for data blocks, which becomes the bottleneck for overall throughput. MPI_Bcast adopts different approaches based on multicast data size [8], [13]. For comparatively smaller data size, MPI_Bcast uses binomial tree approach. In the first round, the multicast sender process sends data to one receiver. In the second round, these two processes send the same data to one additional receiver each and so on. For the medium and bigger data sizes, MPI_Bcast adopts scatter + altogether approach. The altogether is realized by recursive doubling or ring algorithm, where the data is pipelined from one node to the next. In this case, the software handling of data from input to output and the need to ensure reliability across the pipeline become the bottleneck for overall throughput.

We consider the case of a 64 byte packet to compare Shufflecast and IP-multicast architectures in terms of average latency for a one-to-all multicast. Commercial 100 Gbps packet-switches have forwarding delay of at most 1 microsecond. The propagation delay for a 100m fiber link is 0.5 microsecond. The transmission delay for a 64 byte packet at 100 Gbps is 0.005 microsecond. Thus, the approximate per-hop latency is at most 1.505 microsecond. In our analysis, we choose the number of ToRs in such a way that it can be realized with some instance of p, k-Shufflecast having $p <= 8$, e.g., $50 \equiv 5, 2$-Shufflecast, $128 \equiv 8, 2$-Shufflecast and so on upto $1024 \equiv 4, 4$-Shufflecast. Note that, if the number can be realized by more than one Shufflecast instances, we choose the specific instance with the smallest k value to minimize the hop-count. For IP-multicast, we consider the minimal-layer packet-switched network core with identical port-count packet switches as shown in Fig. 1. We observe that a p, k-Shufflecast with $k <= 3$ has smaller average latency than that of a minimal-layer IP-multicast network. However, for the scenarios with $k = 4$, Shufflecast has slightly higher average latency (around one per-hop latency) than IP-multicast.

b) Power consumption analysis: Shufflecast is power efficient compared to both a) peer-to-peer overlay multicast and b) IP-multicast. Ethernet switch ports and the optical transceivers consume power. Passive splitters and fiber optic cables do not consume any power. We count the number of active switch ports and transceivers (similar methodology as [11]) involved in one cluster-wide multicast tree for all three network architectures.

For peer-to-peer overlay multicast on optical circuit-switched core, we assume the lowest possible power consumption, where the data propagates through a chain across all the ToR switches at line-rate. Thus it consumes two switch ports (with two transceivers) from each ToR (both receive and transmit). A minimal-layer IP-multicast network (Fig. 1) would consume excess switch ports (with same number of excess transceivers) in addition to one port (with one transceiver) per ToR. Although for IP-multicast, the data can be instantaneously forwarded from one port to multiple ports in a switch, each port still needs to physically transmit the data to other switches. Thus, more active transmissions result in high power consumption. Finally, Shufflecast requires two active ports (with two transceivers) on each relay ToR (both receive and transmit) and one port (with one transceiver) on each non-relay ToR (only receive) to realize a one-to-all multicast tree. Shufflecast saves the number of active port (and transceiver) usage significantly, because it needs only one transmit port (with one transceiver) on any relay ToR to send the data into optical splitter. Then the splitter performs physical layer multicast without consuming power. For a simple example, in a cluster with 8 ToRs (4-port switches), the number of active ports to support a one-to-all multicast (one server per ToR) will be 16, 22 and 12 for peer-to-peer, IP-multicast and Shufflecast respectively. The corresponding transceiver count will also be the same.

To evaluate power consumption, we vary the number of ToRs using the methodology as given in average latency analysis. The typical power consumption values [6] of different Ethernet switch ports and optical transceivers are given in Table I. For Shufflecast we consider the optical transceiver having sufficient power budget to compensate the insertion loss of different optical splitters. As shown in Fig. 6(b), Shufflecast

TABLE I

MOST RECENT POWER CONSUMPTION AND COST VALUES OF DIFFERENT COMPONENTS

Ethernet switch	Power per port (Watt)	Cost per port (USD)	Optical Transceiver	Max Power Budget (dB)	Power (Watt)	Cost (USD)	Optical Splitter	Insertion Loss (dB)	Cost (USD)
10 Gbps	2.64	55.5	10 Gbps, 10km	14.9	1	27	1:2	4	7.5
25 Gbps	3.75	86.2	25 Gbps, 10km	18	1	59	1:4	7.3	9.3
100 Gbps	14.06	237.5	100 Gbps, 2km	14	3.5	189	1:8	10.6	12

is $1.5 - 1.77\times$ more power efficient than peer-to-peer overlay. Note that for peer-to-peer overlay, we consider the active power consumption only from the network. But in reality, the power consumption will be even more because it also involves the host peers (servers) to receive and transmit the multicast data repeatedly. Also, Shufflecast is $1.55 - 1.85\times$ more power efficient than IP-multicast over the minimal-layer packet-switched network core. The improvement factors (ratio of baseline power consumption to that of Shufflecast) are the same across different data rates, as it only depends on the relative count of switch ports (same as transceivers).

c) Capital cost analysis: The deployment of Shufflecast incurs very little extra hardware cost since the optical devices including passive optical splitters, optical transceivers, and fiber-optic cables are all inexpensive. For a p, k-Shufflecast, each ToR requires one optical splitter, p optical transceivers, p outgoing fiber cables and p switch ports. Table I summarizes the most recent costs [6] of different components. The approximate cost of a duplex single-mode fiber per 100 meter is 37.37 USD. Given a $4, 4$-Shufflecast (spanning 1024 ToRs) with 100 meter fiber optic cable as an example, the capital cost per ToR are approximately $487 - 1867$ USD across different data rates, which is fairly inexpensive for large clusters. Fig. 6(c) shows the improvement in capital cost (ratio of baseline capital cost to that of Shufflecast) per one-to-all multicast tree of Shufflecast compared to IP-multicast (on minimal layer packet-switched network core) with scale at different data rates (10 Gbps, 25 Gbps, 100 Gbps). We consider the necessary components involved in one cluster-wide multicast tree for both Shufflecast (switch ports, transceivers, splitters and fiber-optic cables) and IP-multicast (switch ports, transceivers and fiber-optic cables) architectures. Based on our evaluation, Shufflecast is $1.57 - 1.89\times$ more cost efficient compared to IP-multicast over minimal-layer packet-switched core, across different network scale and data rates. We observe that the improvement factor decreases slightly with higher data rate. The reason is that switch port and transceiver costs are data-rate dependent and start dominating the fiber cost (data-rate independent) at higher data rate. As a result, the higher fiber cost for IP-multicast matters less at higher data rates. If the costs of higher speed switch port and transceiver continue to rise while fiber/splitter cost remain constant, the improvement factor will converge to the relative count of switch ports (same as transceivers) i.e., $1.55 - 1.85\times$.

B. Shufflecast Achieves High Reliability While Supporting Concurrent Multicast Groups With Negligible Overhead

We measure the responsiveness of Shufflecast control plane and experimentally demonstrate that Shufflecast achieves high reliability in presence of concurrent multicast groups using off-the-shelf transport layer solutions [4].

a) Shufflecast has highly responsive control plane: Although Shufflecast has pre-installed static ToR-to-ToR relaying rules, application-directed dynamic ToR-to-server multicast forwarding rule update is required before the multicast starts (Sec. III-B). Based on our measurement, such a multicast rule update on a Quanta T3048-LY2R OpenFlow switch only takes 0.6 msec. Moreover, Shufflecast controller sends parallel requests to the ToRs simultaneously. For big data applications, such latency is negligible compared to their multicast durations, which can easily reach tens of seconds (Sec. VI-D).

b) Shufflecast achieves high reliability while supporting concurrent multicast groups: We perform multicast of 2 GB data size over $2, 2$-Shufflecast with a group size of 16 (1:15 multicast) using NORM [4], a well-known off-the-shelf reliable multicast solution. NORM [4] is a NACK-based reliable multicast protocol enabled with forward error correction (FEC) and the TCP-Friendly Multicast Congestion Control (TFMCC) scheme [58], [59]. We vary the number of concurrent multicast groups from 1 to 16 by running parallel norm sessions on each destination server and invoking the corresponding number of servers as multicast senders. We observe that all the multicast flows get close to fair-share throughput at steady state and the packet loss is below 0.28%. Fig. 6(d) shows that the observed average multicast throughput per group at steady state is almost same as the theoretical fair-share. Therefore, the aggregate network throughput in presence of such concurrent multicast groups is always close to line-rate. Next, we launch 4 multicast flows (group size is 16 and data size is 4 GB) with a progressive staggering of 1.5 sec. Fig. 6(e) shows the individual flow throughput and aggregate network utilization (averaged over 10 runs) variation with time. We observe that multicast flows achieve their fair-share quickly and the overall network utilization is close to line-rate. Note that even when multiple sources send data towards a common destination ToR, the performance will still be predictable due to the following reasons. First, the upper-bound of per-flow fair-share can be pre-computed based on Lemma 3. Second, due to the inherent load balancing of our multicast-aware routing, the relay usage will be evenly distributed and there will be a low chance of a hotspot. When feasible for an application, topology-aware placement of the multicast sources could further improve bandwidth utilization of the network.

C. Shufflecast Achieves High Robustness Against Single Relay Failure and Graceful Performance Degradation After Failure Recovery

We evaluate the reachability impact on Shufflecast under single relay failure. We also evaluate the impact of latency and throughput degradation of Shufflecast after enabling the single relay failure recovery.

a) Shufflecast is robust enough against single-relay failure: Based on our reachability analysis (Sec. III-C), we compute the distribution of reachability impact after a single relay failure on p, k-Shufflecast. Fig. 7(a) shows the

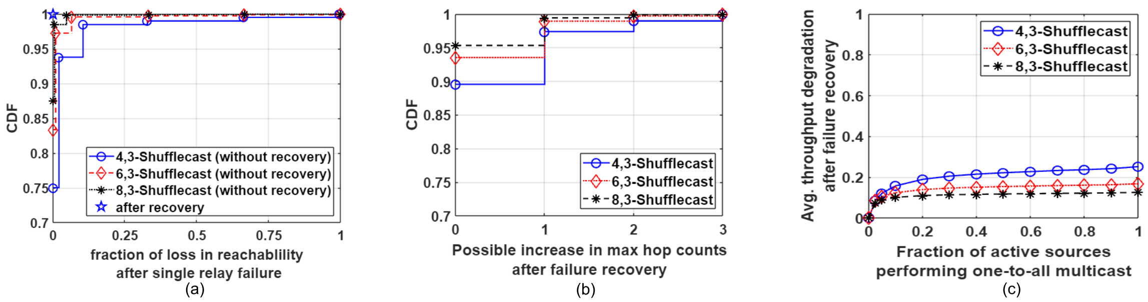


Fig. 7. (a) CDF of fraction of loss in reachability of Shufflecast under single relay failure. Without failure recovery, there is no reachability impact for majority of sources performing one-to-all multicast. With failure recovery, the reachability is completely restored, (b) CDF of excess latency (in terms of max hop count) after single relay failure recovery in Shufflecast. After failure recovery, latency is unchanged for majority of the sources performing one-to-all multicast, (c) Average throughput degrades gracefully after single relay failure recovery in Shufflecast, with varying the number of active sources performing one-to-all multicast.

distribution for different Shufflecast instances. We observe that, the majority of sources does not have any impact in reachability under a single relay failure even before enabling the failure recovery. Also, the size of this majority increases with bigger network scale. As shown in Fig. 7(a), for Shfflecast instances with $p = 4$, $p = 6$, and $p = 8$, 75%, 83%, and 88% of the source ToRs do not lose reachability from a relay failure, respectively. Moreover, we also observe that reachability is completely restored after enabling the single relay failure recovery (Algorithm 2). Hence, Shufflecast is robust enough against single relay failure. On the other hand, as the IP-multicast architecture is hierarchical, the impact of a single link failure would be much worse if a higher-layer link fails. Moreover, the reachability cannot be restored without physical backup switches because such a single link failure will partition the multicast tree.

b) Shufflecast has graceful performance degradation after failure recovery: According to Lemma 1, in a healthy p, k-Shufflecast any source ToR can reach all other ToRs within two complete traversals i.e., maximum hop count is $(2k - 1)$. Based on our analysis, after enabling the failure recovery, the maximum hop count is unchanged for the majority of sources. Also, the upper bound of maximum hop count now becomes $(3k - 1)$, i.e., any source ToR can reach all other ToRs within three complete traversals in the worst case. Fig. 7(b) demonstrates the CDF of possible increase in latency (in terms of maximum hop count) after single relay failure recovery for different Shufflecast instances ($p = 4, 6$ and 8). We observe that, after single relay failure recovery the maximum hop count remains unchanged for $90 - 95\%$ of the sources and the possible increase in maximum hop count is upper bounded by $k = 3$.

In Fig. 7(c) we vary the fraction of active ToR sources performing one-to-all multicast and observe the multicast throughput degradation for different Shufflecast instances after enabling failure recovery. For a given fraction of active sources, we uniformly sample random set of ToRs and compute the relative multicast throughput degradation of those ToR sources between the healthy and failed network (after the failure recovery) averaged over the samples. The throughput for an individual ToR source is defined as the inverse of maximum fair-share for that source in presence of other active sources. As shown in Fig. 7(c) shows that the average multicast throughput of Shufflecast degrades gracefully after failure recovery and the degradation reduces with bigger network scale. For Shufflecast instances with $p = 4, 6$ and 8, the throughput degradation is upper-bounded by $25\%, 16.7\%$ and 12.5% respectively. Such graceful degradation also reflects on the simultaneous multicast capability of Shufflecast. For a healthy p, k-Shufflecast, p ToRs in one column, having their set of relays from disjoint partitions, can simultaneously perform a one-to-all multicast at line-rate. After the single relay failure, two partitions of at least one column are shared, so the degree of parallelism now becomes $(p - 1)$, i.e., $(p - 1)$ ToRs can in parallel perform one-to-all multicast at line-rate.

D. Shufflecast Achieves Improved Application Performance for High-Bandwidth and Low-Latency Applications

We briefly discuss three different workloads and experimentally demonstrate that real-world applications can leverage Shufflecast with only minor modifications.

a) Spark ML: Under Spark Machine Learning applications, we focus on Latent Dirichlet Allocation (LDA), one of the popular iterative machine learning algorithms. We use the Spark LDA implementation [16] with the dataset of 20 Newsgroups as the input corpus [44] which performs the one-to-all multicast for the training vocabulary model (735 MB in size). We use a cluster of 8 servers to run LDA, where the application randomly chooses one server with four cores and 88 GB RAM as the master, while the other seven servers with two cores and 44 GB RAM serve as 14 slave executors. Currently, the application uses Spark's native multicast mechanisms like Cornet [25] and HTTP (repeated unicasts to all receivers) over full-bisection bandwidth network. We use an extension to Spark that can perform multicast [53] over Shufflecast network and compare the application performance with Cornet and HTTP. We obtain the total multicast reading times and application running times averaged over 10 runs, as shown in Fig. 8(a). Shufflecast achieves $3.25\times$ and $6.24\times$ speedup in multicast reading time compared to Cornet and HTTP respectively, with corresponding improvements of 23.41% and 43.1% in overall application runtime.

b) Spark distributed database: TPC-H is a widely used database benchmark of 22 business-oriented queries with high

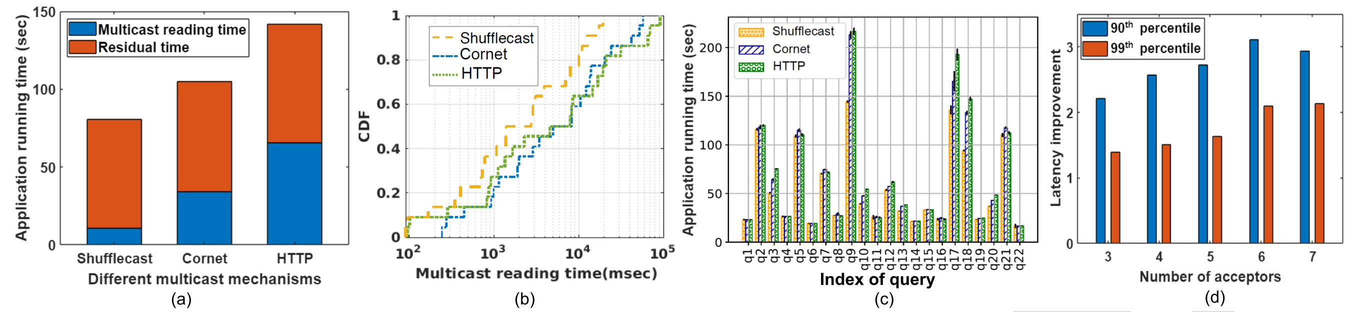


Fig. 8. Application performance improvements of Shufflecast compared to native multicast mechanisms over full-bisection bandwidth network. (a) For LDA, the speedup in multicast reading time are $3.25\times$ and $6.24\times$ compared to Cornet and HTTP, respectively. The corresponding improvement in application running time are 23.41% and 43.1% (b) CDF of TPC-H multicast reading time. Shufflecast improves the distribution and achieves $2.7\times$ and $3.5\times$ speedup in total multicast reading time compared to Cornet and HTTP respectively, (c) The average running time of each TPC-H query (q1 to q22) over the three multicast mechanisms. For certain queries (e.g., 1, 4, 6, 14, 15, 22), the amount of multicast data is either very small (under 200 MB) or non-existent, so there is no visible difference between Shufflecast, Cornet, and HTTP. However, for other queries (e.g., 9, 17, 18), the multicast data is large (5 GB). Shufflecast improves the total query running time by 13.7% compared to Cornet and 17% compared to HTTP, (d) The latency improvements (ratio of baseline latency to that of Shufflecast) of multicast Paxos over Shufflecast compared to unicast Paxos over full-bisection bandwidth network with one sender and varying number of acceptors. The 90^{th} and 99^{th} percentile latency improvements are $2.21 - 2.93\times$ and $1.39 - 2.13\times$ respectively.

complexity and concurrent data modifications [3]. We run these queries using the Spark SQL framework [24]. The database tables are 16 GB in size overall, and the multicast data is one of such tables with size ranging from 4 MB to 6.2 GB for the distributed database join, making a total of 48.3 GB of multicast data across queries. We compare the performance of TPC-H with and without Shufflecast keeping the same server configuration as Spark ML. Fig. 8(b) shows the multicast reading time distribution of different multicast mechanisms across all TPC-H queries (queries 1 to 22). Shufflecast improves the distribution and gets speedup of $2.7\times$ and $3.5\times$ in total multicast reading time compared to Cornet and HTTP respectively. Fig. 8(c) shows the application running time of each TPC-H query averaged over 10 runs. For certain queries (e.g., 1, 4, 6, 14, 15, 22), the amount of multicast data is either very small (<200 MB) or non-existent, showing no visible difference between Shufflecast, Cornet, and HTTP. However, for other queries (e.g., 9, 17, 18), multicast data is large (5 GB). The improvement of total query running time is 13.7% compared to Cornet and 17% compared to HTTP.

c) Paxos-based consensus protocol: Paxos [36], [37] is a consensus protocol that provides the foundation for building distributed fault-tolerant systems. Paxos has distributed entities called proposers, acceptors and learners. The execution of the protocol consists of four major steps, out of which three steps require one-to-many communications. As the messages tend to be small, the performance of Paxos is sensitive to latency. We run Paxos where the client repeatedly (100 times) sends 1 Byte values to the proposer. The client sends the next value as soon as the previous is successful, and repeats for one hundred iterations; each iteration provides a latency measurement. All acceptors are placed on different servers. We run multicast-based Paxos [1] (natively leverage network-level multicast) over Shufflecast network (no application modification required) and compare the latency with unicast-based Paxos [2] (repeated-unicasts to realize multicast) running over full-bisection bandwidth network, with one sender and varying number of acceptors. Fig. 8(d) shows that Shufflecast improves the tail latency significantly, e.g., 90^{th} and 99^{th} percentile latency improvements (ratio of baseline latency to that of Shufflecast) are $2.21 - 2.93\times$ and $1.39 - 2.13\times$ respectively across different number of acceptors.

VII. Related Work

Recent work has explored how software-defined networks (SDN) can be leveraged to improve IP-multicast support on packet-switched network (e.g. tree construction, group forwarding state maintenance, and packet retransmissions) in the cloud data center setting, which is related to the compute cluster environment [18], [39]–[42], [50], [55]. Our Shufflecast architecture directly connects the ToR switches which significantly reduces the excess resource usage. Also, shufflecast eliminates the need for run-time ToR-to-ToR-level multicast tree construction, group state exists only at the network edge. There have been proposals [48], [49], [56], [57], [64] that use a MEMS-based OCS as a connectivity substrate to construct optical multicast trees via optical splitters. However, they are not scalable, they cannot achieve predictable performance, and they incur significant cost. Their scalability is limited by the centralized OCS, which has only a few hundred ports [17], [31], [46], and these ports need to interconnect all ToRs and all in/out ports of optical splitters. Scalability is further limited by the need for optical power amplification, which is difficult and expensive when the tree gets large. The performance predictability of these proposals is hurt by long circuit switch configuration delays that are exacerbated by the need to concatenate multiple optical circuits through splitters to form the tree. Moreover, OCS incurs significant cost which restricts such proposals from large scale deployment. In contrast, Shufflecast provides simple, scalable and data-rate agnostic multicast in a more power efficient and economical way. [61] proposes a topology that eliminates the centralized OCS, but its scalability is inherently limited by splitter fan-out and the entire proposal consists of only the topology design. In contrast, Shufflecast's topology can scale to an arbitrary size even with a small splitter fanout and we have demonstrated the complete system's effectiveness using end-to-end applications.

VIII. Conclusion

Optical circuit-switched (OCS) network core has several advantages to be a potential candidate for next-generation compute clusters. However, there is no inherent support for multicast by such networks. Shufflecast architecture can complement those high performance OCS-based core and support data-rate agnostic multicast maintaining low power and low capital cost. Shufflecast's data plane is scalable and supports line-rate throughput; its control plane is simple and responsive; Shufflecast is robust enough against failure. Experiments using a complete hardware and software prototype of Shufflecast show that Shufflecast can improve the performance of real-world applications with minor modifications.

Appendix A

A. Details of Shufflecast Data Plane

1) Scalability of Shufflecast Fabric: Shufflecast can scale easily even with small fanout splitters. Fig. 9 shows $2,3$-Shufflecast consisting of $3 \cdot 2^3 = 24$ ToRs arranged in 3 columns connected via 1:2 optical splitters, and each column has $2^3 = 8$ ToRs. We can realize even bigger instances of Shufflecast with small p. For example, $4,4$-Shufflecast uses 1:4 splitters, covering 1024 ToRs.

2) Detailed Analysis of Multicast-Aware Routing: First, the next-hop relay computation algorithm (Algorithm 1) computes the column-difference parameter X ($<=k$) between the destination ToR ($dest$) and the current ToR (cur: initialized to src). If both the ToRs belong to the same column, X is considered as k (lines 3 and 4), otherwise X ($<k$) is computed as stated by line 6. From the construction of Lemma 1, we observe that X dictates the hop count from cur to $dest$. If both the ToRs belong to the same column, both X and the hop count are k (reachable at the end of the first cycle). Otherwise, $dest$ is reachable either in hop count X (during the first cycle) or $k+X$ (during the second cycle). Next, the algorithm checks whether the hop count from cur to $dest$ is $<=k$ (line 8) by matching their partial row-ID digits ($k-X$ most and least significant row-ID digits of the $dest$ and cur respectively). Finally, the next-hop ($next$) ToR ID is determined by shifting the current ToR's row-ID to the left by one digit, and then putting the $(X-1)^{th}$ digit of the destination ToR's row-ID (line 9) if the condition is true, or putting the $(k-X'-1)^{th}$ digit of the source ToR's row-ID (line 12) if the condition is false, where X' ($<k$) is the column-difference parameter between cur and src (line 11).

3) Proofs of Lemmas: **Proof of Lemma 1:** By construction of a p,k-Shufflecast, any given source ToR has 1:p splitter connecting p ToRs of the next column in 1st hop, again from those p ToRs another p^2 ToRs at two-columns ahead from the source are reachable in 2nd hop and so on. Eventually p^{k-1} ToRs belonging to one partition at previous column of source are reachable in $(k-1)$ hops which is sufficient for reaching all p^k ToRs of its own column in the next k^{th} hop. During the second cycle, the remaining ToRs of next column from the source are all reachable from any of the partitions of p^{k-1} ToRs at source column. The same scenario follows for all the consecutive columns during the second

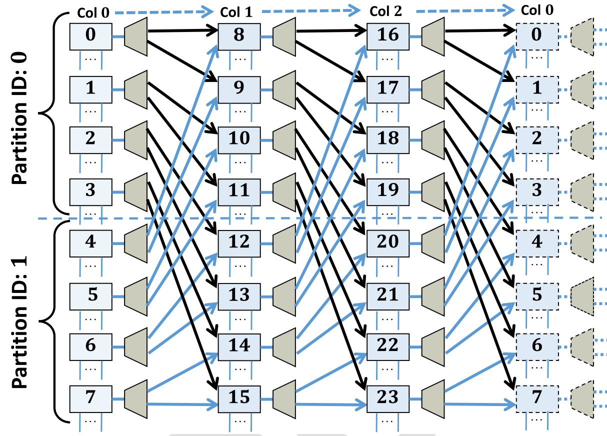


Fig. 9. Connectivity of $2,3$-Shufflecast.

cycle, reaching the remaining ToRs of all the other columns. Finally, the remaining ToRs at the previous column of source can be reached in another $(k-1)$ hops. Therefore, all the ToRs are reachable within two cycles of traversal i.e., the hop count is at most $(k+k-1) = 2k-1$.

Proof of Lemma 2: By construction of p,k-Shufflecast, for the destinations reachable in at most k hops (i.e., during the first cycle), the chosen relays are at most $(k-1)$ hops away from the source, with most significant digit as source row-ID digits left shifted by at most $(k-1)$ places. As a result, the relays are inherently chosen from the partition IDs defined by the source row-ID digits. Hence, appending the pre-calculated destination digit (r^d_{X-1}) as the least significant digit ensures the shortest-path next-hop relay ID following the *partition criteria*. After first cycle, all the k source row-ID digits are ignored due to k effective left shifts. Therefore, for all the remaining ToRs reachable in the second cycle, the algorithm ensures the *partition criteria* by appending the pre-calculated source row-ID digit ($r^s_{k-X'-1}$) as the least significant digit during the first cycle. These digits govern the selective choice of relays from proper partition IDs during the second cycle. Hence, any given source ToR can perform one-to-all multicast following the partition criteria.

Proof of Lemma 3: Following the partition criteria in 2, a given source ToR $(c^s, r^s_{k-1}, r^s_{k-2} \dots r^s_1 r^s_0)$ in a p,k-Shufflecast performs one-to-all multicast using relays from its own column with partition ID r^s_{k-1}, from next column with partition ID r^s_{k-2} and so on, finally from previous column with partition ID r^s_0. We also know, each column contains p partitions as every row-ID digit can have p distinct values ($\in [0, p-1]$). Eventually, to perform one-to-all multicast at line-rate, the group of source ToRs are to be chosen so that the relays are disjoint i.e., from distinct partitions at every column. Thus for the given source, the group of other source ToRs from the same column must have all distinct k row-ID digits. Intuitively, we must choose one ToR from each of the p partitions which at least makes all the most significant digits distinct. For example, given source ToR row-ID, if we choose one $j \in [0, p-1]$ and perform $(r^s_i + j) \bmod p$ for all $i \in [0, k-1]$, eventually we get p ToRs having all distinct k row-ID digits and hence they can perform one-to-all multicast

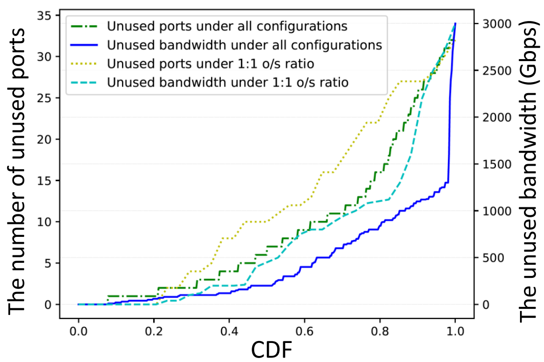


Fig. 10. CDF of unused ports and unused bandwidth under all configurations and 1:1 oversubscription ratio configuration.

simultaneously at line-rate using relays from distinct partition. Now, if we choose two such groups of p ToRs, effectively we have two ToRs from each of the p partitions. Thus, for each of the k places, there exist two unique ToRs using the same digit twice a given place which results them uniquely sharing the relays from same partition. Hence, those $2p$ ToRs can make one-to-all multicast simultaneously at half of the line-rate. Extending this idea, we can choose all such p^{k-1} groups of p ToRs i.e., all the p^k ToRs of one column using the relays from same partition and hence they can make one-to-all multicast at p^{k-1} fraction of the line-rate.

Proof of Lemma 4: In a p, k-Shufflecast, every ToR of a given column is connected to another p ToRs of its next column, and every column has p^k ToRs. Therefore, we need at least p^{k-1} ToRs of a given column to reach all the ToRs of the next column. Hence a given source ToR must require at least p^{k-1} number of relays from each of the k column to perform one-to-all multicast. In Lemma 2 we have already proved, with multicast-aware routing any source ToR can perform one-to-all multicast using the relays from one partition at each column. From the definition of partition we know, every partition has p^{k-1} ToRs which is the same as the minimum relay requirement. Thus, multicast-aware routing minimizes the relay usage. Also, we know there are p partitions per column. Hence, with such minimum relay requirement, maximum p sources in one column can possibly use disjoint set of relays from every column and consequently can perform one-to-all multicast simultaneously at line-rate. This is indeed the number of simultaneous one-to-all multicast supported by multicast-aware routing at line-rate as proved in Lemma 3. Thus, multicast-aware routing is optimal in terms of relay usage and multicast performance.

B. Analysis to Show Unused ToR Ports Often Exists

Our methodology considers a wide range of network configurations. For each configuration, we choose the ToR switch that minimizes the amount of unused bandwidth. We study 14 types of ToR switches with different port configurations from several well-known companies. Specifically, we use 2 HP switches with either 24×10 Gbps ports or $48 \times 10 + 4 \times 40$ Gbps ports, 2 Juniper switches with either 32×10 Gbps ports or $48 \times 10 + 4 \times 100$ Gbps ports, 3 Arista switches ranging from $32 \times 10 + 4 \times 40$ Gbps ports to $96 \times 10 + 8 \times 40$ Gbps ports, and 8 Cisco switches ranging from 32×40 Gbps ports to 64×100 Gbps ports. For the network configurations, we adopt several oversubscription (o/s) ratios reported in the literature, i.e., 1:1, 3:2, 3:1, 4:1, 5:1, 8:1, 10:1 and 20:1 [19], [28], [32], [62]. We also include a few additional o/s ratios: x:1 where $x \in [1, 10]$. We consider commercially available standard rack cabinet sizes ranging from 18U to 48U [7], [9], and five different per-server network port speed configurations – 10 Gbps, 2×10 Gbps, 25 Gbps, 40 Gbps and 2×25 Gbps. The detailed results are shown in Fig. 10. Indeed, unused ports, as well as a large amount of unused bandwidth, often exist. Among all cases, the configuration of 1:1 o/s is unique and the unused ToR ports truly cannot be used to add more bandwidth into the network core.

Acknowledgment

The authors thank the editors and anonymous reviewers for their valuable feedback. They also thank Prof. Debasish Datta for very helpful discussions.

References

[1] *Libfastpaxos*. [Online]. Available: https://sourceforge.net/projects/libpaxos/files/LibFastPaxos/src-rev-17/ AQ:4
[2] *Libpaxos3*. [Online]. Available: https://sourceforge.net/projects/libpaxos/files/LibPaxos3/
[3] (2001). *TPC Benchmark H*. [Online]. Available: http://www.tpc.org/tpch/
[4] (2009). *Nack-Oriented Reliable Multicast (Norm) Transport Protocol*. [Online]. Available: https://tools.ietf.org/html/rfc5740
[5] (2015). *Ryu Openflow Controller*. [Online]. Available: http://osrg.github.io/ryu/
[6] (2019). *Fs (Fiberstore)—Leading Communication Systems Integrator and Optical Solutions Provider for Data Centers*. [Online]. Available: http://www.fs.com
[7] (2019). *HPE—Rack Cabinet Provider for Data Centers*. [Online]. Available: https://buy.hpe.com/us/en/rack-power-infrastructure/racks/server-racks/racks/hpe-g2-enterprise-series-racks/p/1009803311
[8] *MPI Broadcast and Collective Communication*. [Online]. Available: http://mpitutorial.com/tutorials/mpi-broadcast-and-collective-communication/
[9] *Server Cabinet Enclosures—Common Standard Rack Cabinet for Data Centers*. [Online]. Available: https://www.racksolutions.com/rack-mount-enclosure.html
[10] (2020). *25 dB Gain DWDM EDFA Pre-Amplifier*. [Online]. Available: https://www.fs.com/products/107367.html
[11] H. Ballani *et al.*, "Sirius: A flat datacenter network with nanosecond optical switching," in *Proc. Annu. Conf. ACM Special Interest Group Data Commun. Appl., Technol., Archit., Protocols Comput. Commun.*, Jul. 2020, pp. 782–797.
[12] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast," in *Proc. Conf. Appl., Technol., Archit., Protocols Comput. Commun.*, 2002, pp. 205–217.
[13] J. Bao, D. Dong, B. Zhao, and Z. Gong, "ICAST: Accelerating high-performance data center applications by hybrid electrical and optical multicast," in *Proc. IEEE 23rd Int. Conf. Parallel Distrib. Syst. (ICPADS)*, Feb. 2017, pp. 302–309.
[14] D. Basin, K. Birman, I. Keidar, and Y. Vigfusson, "Sources of instability in data center multicast," in *Proc. 4th Int. Workshop Large Scale Distrib. Syst. Middleware*, 2010, pp. 32–37.
[15] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, Jan. 2003.
[16] Z. Cai, Z. J. Gao, S. Luo, L. L. Perez, Z. Vagena, and C. Jermaine, "A comparison of platforms for implementing and running very large scale machine learning algorithms," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2014, pp. 1371–1382.
[17] Calient. (2019) *Series Optical Circuit Switch*. [Online]. Available: http://www.calient.net

[18] J. Cao *et al.*, "Datacast: A scalable and efficient reliable group data delivery service for data centers," *IEEE J. Sel. Areas Commun.*, vol. 31, no. 12, pp. 2632–2645, Dec. 2013.

[19] Z. Cao, R. Proietti, and S. J. B. Yoo, "Hi-LION: Hierarchical large-scale interconnection optical network with AWGRs," *J. Opt. Commun. Netw.*, vol. 7, no. 1, p. A97, Jan. 2015.

[20] R. Carlson, "Considerations for choosing top-of-rack in today's fat-tree switch fabric configurations," *Cabling Installation Maintenance Mag.*, Feb. 2014. AQ:5

[21] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "Splitstream: high-bandwidth multicast in cooperative environments," *ACM SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 298–313, 2003.

[22] M. Castro *et al.*, "An evaluation of scalable application-level multicast built using peer-to-peer overlays," in *Proc. 23nd Annu. Joint Conf. Comput. Commun. Soc.*, 2003, pp. 1510–1520.

[23] A. Chatzieleftheriou, S. Legtchenko, H. Williams, and A. Rowstron, "Larry: Practical network reconfigurability in the data center," in *Proc. 15th USENIX Symp. Netw. Syst. Design Implement.*, 2018, pp. 141–156.

[24] T. Chiba and T. Onodera, "Workload characterization and optimization of TPC-H queries on apache spark," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Apr. 2016, pp. 112–121.

[25] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, "Managing data transfers in computer clusters with orchestra," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 98–109, Aug. 2011.

[26] Y.-H. Chu, S. G. Rao, S. Seshan, and H. Zhang, "A case for end system multicast," *IEEE J. Sel. Areas Commun.*, vol. 20, no. 8, pp. 1456–1471, Oct. 2002.

[27] A. Das, I. Gupta, and A. Motivala, "SWIM: Scalable weakly-consistent infection-style process group membership protocol," in *Proc. Int. Conf. Dependable Syst. Netw.*, 2002, pp. 303–312.

[28] A. Dixit, P. Prakash, Y. Hu, and R. Kompella, "On the impact of packet spraying in data center networks," in *Proc. INFOCOM*, Apr. 2013, pp. 2130–2138.

[29] R. Furrer, M. G. Genton, and D. Nychka, "Covariance tapering for interpolation of large spatial datasets," *J. Comput. Graph. Statist.*, vol. 15, no. 3, pp. 502–523, 2006.

[30] M. Ghobadi *et al.*, "Projector: Agile reconfigurable data center interconnect," in *Proc. ACM SIGCOMM Conf.*, 2016, pp. 216–229.

[31] Glimmerglass. (2018). *Intelligent Optical System*. [Online]. Available: http://www.glimmerglass.com

[32] A. Greenberg *et al.*, "Vl2: A scalable and flexible data center network," in *Proc. SIGCOMM*, Aug. 2009, pp. 51–62.

[33] M. Hosseini, D. T. Ahmed, S. Shirmohammadi, and N. D. Georganas, "A survey of application-layer multicast protocols," *IEEE Commun. Surveys Tuts.*, vol. 9, no. 3, pp. 58–74, 3rd Quart., 2007.

[34] S. Itoh, P. Ordejón, and R. M. Martin, "Order-n tight-binding molecular dynamics on parallel computers," *Comput. Phys. Commun.*, vol. 88, nos. 2–3, pp. 173–185, 1995.

[35] J. Jannotti *et al.*, "OverCast: Reliable multicasting with on overlay network," in *Proc. 4th Conf. Symp. Oper. Syst. Design Implement.*, vol. 4, 2000, p. 14.

[36] L. Lamport, "The part-time parliament," *ACM Trans. Comput. Syst.*, vol. 16, no. 2, pp. 133–169, 1998.

[37] L. Lamport *et al.*, "Paxos made simple," *ACM Sigact News*, vol. 32, no. 4, pp. 18–25, 2001.

[38] S. Legtchenko, N. Chen, D. Cletheroe, A. Rowstron, H. Williams, and X. Zhao, "XFabric: A reconfigurable in-rack network for rack-scale computers," in *Proc. 13th USENIX Symp. Netw. Syst. Design Implement.*, 2016, pp. 15–29.

[39] D. Li, H. Cui, Y. Hu, Y. Xia, and X. Wang, "Scalable data center multicast using multi-class Bloom filter," in *Proc. 19th IEEE Int. Conf. Netw. Protocols*, 2011, pp. 266–275.

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

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

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

[43] W. M. Mellette *et al.*, "RotorNet: A scalable, low-complexity, optical datacenter network," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2017, pp. 267–280.

[44] T. Mitchell. (1999). *20 Newsgroups*. [Online]. Available: http://kdd.ics.uci.edu/databases/20newsgroups/20newsgroups.html

[45] Y. Ohsita and M. Murata, "Optical data center networks: Architecture, performance, and energy efficiency," in *Handbook Data Centers*. Springer, 2015, pp. 351–391. AQ:6

[46] Polatis. (2016). *Series 7000 Software Defined Optical Switch*. [Online]. Available: http://www.polatis.com

[47] G. Porter *et al.*, "Integrating microsecond circuit switching into the data center," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 447–458, 2013.

[48] P. Samadi, D. Calhoun, H. Wang, and K. Bergman, "Accelerating cast traffic delivery in data centers leveraging physical layer optics and sdn," in *Proc. Int. Conf. Opt. Netw. Design Modeling*, 2014, pp. 73–77.

[49] P. Samadi, V. Gupta, J. Xu, H. Wang, G. Zussman, and K. Bergman, "Optical multicast system for data center networks," *Opt. Exp.*, vol. 23, no. 17, pp. 22162–22180, 2015.

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

[51] V. Shrivastav *et al.*, "Shoal: A network architecture for disaggregated racks," in *Proc. 16th USENIX Symp. Netw. Syst. Design Implement.*, 2019, pp. 255–270.

[52] A. Singh *et al.*, "Jupiter rising: A decade of CLOS topologies and centralized control in Google's datacenter network," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 183–197, 2015.

[53] X. S. Sun, Y. Xia, S. Dzinamarira, X. S. Huang, D. Wu, and T. S. E. Ng, "Republic: Data multicast meets hybrid rack-level interconnections in data center," in *Proc. IEEE 26th Int. Conf. Netw. Protocols (ICNP)*, Sep. 2018, pp. 77–87.

[54] A. Thusoo *et al.*, "Hive: A warehousing solution over a map-reduce framework," *VLDB Endowment*, vol. 2, no. 2, pp. 1626–1629, 2009.

[55] Y. Vigfusson, H. Abu-Libdeh, M. Balakrishnan, K. Birman, and Y. Tock, "Dr. Multicast: Rx for data center communication scalability," in *Proc. 2nd Workshop Large-Scale Distrib. Syst. Middleware*, 2008, pp. 349–362.

[56] H. Wang, C. Chen, K. Sripanidkulchai, S. Sahu, and K. Bergman, "Dynamically reconfigurable photonic resources for optically connected data center networks," in *Proc. Opt. Fiber Commun. Conf.*, 2012, p. 2.

[57] H. Wang, Y. Xia, K. Bergman, T. S. E. Ng, S. Sahu, and K. Sripanidkulchai, "Rethinking the physical layer of data center networks of the next decade: Using optics to enable efficient*-cast connectivity," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 3, pp. 52–58, Jul. 2013.

[58] J. Widmer and M. Handley, "Extending equation-based congestion control to multicast applications," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 31, no. 4, pp. 275–285, Oct. 2001.

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

[60] B. D. Wozniak, F. D. Witherden, F. P. Russell, P. E. Vincent, and P. H. Kelly, "Gimmik-generating bespoke matrix multiplication kernels for accelerators: Application to high-order computational fluid dynamics," *Comput. Phys. Commun.*, vol. 202, pp. 12–22, Oct. 2016.

[61] D. Wu, X. Sun, Y. Xia, X. S. Huang, and T. E. Ng, "HyperOptics: A high throughput and low latency multicast architecture for datacenters," in *Proc. HotCloud*, 2016, pp. 1–5.

[62] D. Wu, W. Wang, A. Chen, and T. S. E. Ng, "Say no to rack boundaries: Towards a reconfigurable pod-centric dcn architecture," in *Proc. ACM Symp. SDN Res.*, Apr. 2019, pp. 112–118.

[63] G. Wu, H. Gu, K. Wang, X. Yu, and Y. Guo, "A scalable AWG-based data center network for cloud computing," *Opt. Switching Netw.*, vol. 16, pp. 46–51, Oct. 2015.

[64] Y. Xia, T. S. E. Ng, and X. S. Sun, "Blast: Accelerating high-performance data analytics applications by optical multicast," in *Proc. INFOCOMM*, 2015, pp. 1930–1938.

[65] M. Xu, C. Liu, and S. Subramaniam, "PODCA: A passive optical data center network architecture," *J. Opt. Commun. Netw.*, vol. 10, no. 4, pp. 409–420, 2018.

[66] T. Ye, T. T. Lee, M. Ge, and W. Hu, "Modular AWG-based interconnection for large-scale data center networks," *IEEE Trans. Cloud Comput.*, vol. 6, no. 3, pp. 785–799, Oct. 2016.

[67] D. Zhuo, M. Ghobadi, R. Mahajan, K.-T. Förster, A. Krishnamurthy, and T. Anderson, "Understanding and mitigating packet corruption in data center networks," in *Proc. Conf. ACM Special Interest Group Data Commun.*, 2017, pp. 362–375.

Sushovan Das is currently pursuing the Ph.D. degree with the Department of Computer Science, Rice University. His research interest includes novel optical architecture design to optimize application performance for future generation cloud infrastructures.

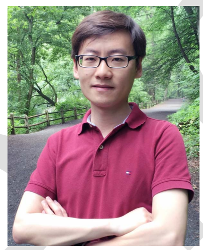
Afsaneh Rahbar received the Ph.D. degree from the Department of Computer Science, Rice University. She is currently a Research Scholar at Rice University. Her research interests include static analysis, high performance computing, and data center networks.

AQ:7

Xinyu Crystal Wu (Graduate Student Member, IEEE) is currently pursuing the Ph.D. degree with the Department of Computer Science, Rice University. Her research interests include programmable networks and distributed systems, including lossless networks, network infrastructure, distributed training optimization, network monitoring, and diagnose.

Zhuang Wang is currently pursuing the Ph.D. degree with the Department of Computer Science, Rice University. His research interests include distributed machine learning and networking systems, including efficient distributed training, gradient compression algorithms, and performance isolation in public clouds.

Weitao Wang is currently pursuing the Ph.D. degree with the Department of Computer Science, Rice University. His research interests include application-infrastructure co-design, including congestion control, network monitoring, and network scheduling.



Ang Chen is currently an Assistant Professor with the Department of Computer Science, Rice University. His research interests span networking, security, and systems, with a particular focus on making networked systems more reliable, efficient, and secure.

T. S. Eugene Ng received the Ph.D. degree in computer science from Carnegie Mellon University in 2003. He is currently a Full Professor of computer science at Rice University. His research interests lie in developing new network models, network architectures, and holistic networked systems that enable a robust and manageable network infrastructure. He received the U.S. National Science Foundation (NSF) CAREER Award in 2005 and the Alfred P. Sloan Fellowship in 2009.