SplitCast: Optimizing Multicast Flows in Reconfigurable Datacenter Networks

Long Luo
University of Electronic Science and Technology of China
P.R. China

Klaus-Tycho Foerster
Faculty of Computer Science
University of Vienna
Austria

Stefan Schmid
Faculty of Computer Science
University of Vienna
Austria

Hongfang Yu
University of Electronic Science and Technology of China
P.R. China

***Abstract*—Many modern cloud applications frequently generate multicast traffic, which is becoming one of the primary communication patterns in datacenters. Emerging reconfigurable datacenter technologies enable interesting new opportunities to support such multicast traffic in the physical layer: novel circuit switches offer high-performance inter-rack multicast capabilities. However, not much is known today about the algorithmic challenges introduced by this new technology.**

This paper presents SplitCast, a preemptive multicast scheduling approach that fully exploits emerging physical-layer multicast capabilities to reduce flow times. SplitCast dynamically reconfigures the circuit switches to adapt to the multicast traffic, accounting for reconfiguration delays. In particular, SplitCast relies on simple single-hop routing and leverages flexibilities by supporting *splittable* multicast so that a transfer can already be delivered to just a subset of receivers when the circuit capacity is insufficient. Our evaluation results show that SplitCast can reduce flow times significantly compared to state-of-the-art solutions.

I. Introduction

With the vast popularity of data-centric applications, it is expected that datacenter traffic will continue to grow explosively in the coming decades, pushing today's datacenter designs to their limits. Accordingly, we currently witness great efforts to design innovative datacenter topologies, offering high throughput at low cost [1]–[8], or even allowing to adjust networks adaptively, in a demand-aware manner [9]–[11].

A particularly fast-growing communication pattern in datacenters today are *multicasts*. Data-centric applications increasingly rely on *one-to-many communications* [12, 13], including distributed machine learning [14], applications related to financial services and trading workloads [15, 16], virtual machine provisioning [17], pub/sub systems [18], etc. [19, 20]. In many of these applications, multicasts are frequent and high-volume and are becoming a bottleneck [12, 13, 21]–[23]. For example, in distributed machine learning frameworks, the training model has to be updated and communicated to all computation nodes frequently [14]. Despite the wide use of multicast, there is currently no support among cloud providers for efficient multicasting: a missed opportunity [12, 21].

Our paper is motivated by emerging optical technologies which can potentially improve the performance of transferring multicast flows by supporting in-network multicast on the physical layer. In particular, recent advancements for reconfigurable circuit switches (RCS) allow to set up high-bandwidth port-to-multiport circuit connections which support adaptive and demand-aware multicasting among top-of-rack (ToR) switches [11, 13, 22]–[26]. See the example in Fig. 1(a)

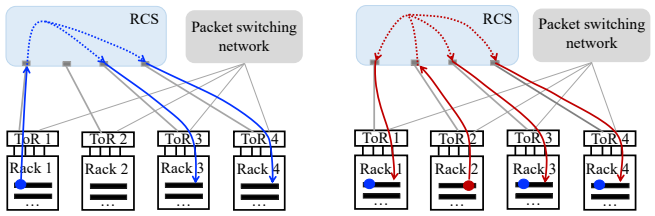


Fig. 1. A reconfigurable datacenter network based on a reconfigurable circuit switch (RCS) allows to enhance the conventional packet switching network with multicast capabilities: e.g. at time $t = 1$, RCS supports multicasting ToR 1 to ToRs 3 and 4, at time $t = 2$, from ToR 2 to ToRs 1, 3 and 4.

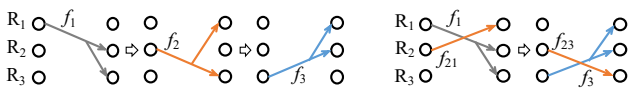
for an illustration: in this example, the RCS can be used to first create a circuit connection to directly multicast data from ToR 1 to ToRs 3 and 4, after which the RCS is reconfigured to support multicast from ToR 2 to ToRs 1, 3, and 4 in a demand-aware manner (see Fig. 1(b)). In general, the possibility to simultaneously fan out to multiple receiver racks over a single port-to-multiport circuit connection can greatly improve the delivery of multicast flows.

However, while this technology is an interesting *enabler*, we currently lack a good understanding of how to algorithmically *exploit* this technology to optimize multicast communications. While interesting first approaches such as Blast [22] and Creek [23] are emerging, these solutions are still limiting in that they cannot fully use the high capacity of the circuit switch as they only transfer data as long as a circuit connection can be created to match *all* the receivers of a flow.

Motivating Example. Consider the situation shown in Fig. 2, where there are three multicast flows f_1, f_2 and f_3, each of unit size. In this example, each node denotes one port of the circuit switch and connects a ToR. Using the approaches in [22, 23], the three flows will be delivered one by one: the output of a circuit (right) can only receive traffic from a single input (left) on the other side of the circuit, while any two of these three flows compete for one outport (f_1 contends with f_2, f_3 at outport R_3, R_2, respectively, and f_2 contends with f_3 at outport R_1). Hence, a circuit can be created only for matching one of the multicast flows at a time, as shown in Fig. 2(a), resulting in an average flow time (i.e., average of the multicast flow durations) of two units of time.

However, if just matching the multicasts, we observe that there are some *free* ports: an unused optimization opportunity we want to exploit in this work. For example, R_1, R_2, R_3 in Fig. 2(a) are unused when f_1, f_2, f_3 are delivered, respectively.

In principle, all these ports can be fully used if one splits and schedules f_2 in two steps, each transferring data to one of its receivers, as shown in Fig. 2(b). This reduces the average flow time by 16.7%.



(a) Non-splittable multicast (b) Splittable multicast

Fig. 2. A splittable multicast decreases the average flow time by 16.7%, compared to a non-splittable multicast.

Problem. This simple example motivates us, in this paper, to study the opportunities to improve the multicast performance by supporting *splittable multicasting*: multicast data which can be sent just to a subset of receivers at a time. Essentially, a circuit connecting an input port to multiple outports forms a hyperedge and creating port-to-multiport circuit connections for multicast data produces a hypergraph matching problem [27]. We are particularly interested in the *splittable multicast matching problem*: a sender port/rack is allowed to match to just a subset of the receiver ports/racks of a multicast flow at a time.

More specifically, our objective is to design a scheduler which reconfigures the circuit switch in a demand-aware manner, in order to minimize *multicast flow times*. In our model, a created port-to-multiport circuit allows one ToR to simultaneously transmit to multiple ToRs at bandwidth b_c. Inter-rack multicast flows arrive in an online manner and have a certain size, and also need to be transmitted from server to ToR first via a link with bandwidth $b_s \leq b_c$.

Our Contributions. This paper presents SplitCast, an efficient scheduler for multicast flows in reconfigrauble datacenter networks, which leverages splitting opportunities to improve performance of multicast transfers. By dynamically adjusting the network topology and supporting splittable multicasting, SplitCast fully exploits the available infrastructure, reducing flow times and improving throughput. SplitCast is also simple in the sense that it relies on segregated routing: similarly to reconfigurable unicast solutions, routing is either 1-hop along the reconfigurable circuit switch or along the (static) packet switching network, but never a combination.

We provide insights into the algorithmic complexity of a simplified problem variant and present a general problem formulation. We also show how to tackle the challenges of splittable flows and how to account for reconfiguration delays. Our extensive simulations indicate that SplitCast can significantly outperform state-of-the-art approaches: not only are the flow times reduced by up to 9× but also the throughput increased by up to 3×. We will also make our implementation and experimental results publicly available so as to facilitate future research and reproducibility[1].

Organization. Our paper is organized as follows. We first provide some algorithmic background to the multicast transfer problem in §II. We next provide a formulation and motivating example for the generalized scheduling problem, and discuss the challenges of splittable multicast scheduling in §III. In §IV, we present our solution, SplitCast, in detail. After reporting on simulation results in §V, we review related work in §VI, and conclude in §VII.

[1]https://github.com/ilongluo/SplitCast.git

II. Background to the Algorithmic Problem

Before studying how to reconfigure circuit switches to optimally serve multicasts online, we first present some background insights into the fundamental *static* problem underlying the design of datacenter topologies optimized for multicasts.

Scheduling. Sun and Ng [23] proved that already the optimal scheduling of unicast transfers w.r.t. flow time is NP-hard, via the sum coloring problem. Hence it is NP-hard for both splittable and unsplittable multicast transfer scheduling.

Achievability. Sundararajan et al. [28] study whether given rate requirement vectors are achievable in multicast settings. Herein the question is, if there is a schedule to serve the flow rates, without unbounded queues. They prove that deciding achievability is NP-hard, for splittable and unsplittable settings. They also show connections to fractional graph coloring.

Matching. Computing a maximum matching for unicast transfers can be done in polynomial time, even for general settings [29]. However, the (unsplittable) multicast matching problem is essentially a specific hypergraph matching problem [27]. This connection has also been observed in [22], where the authors also provide some intuition that "*the multicast matching problem has a similar complexity*". We now show that the NP-hardness already holds for simple scenarios:

Theorem 2.1: Multicast matching is NP-hard, already in the unit weight case with at most $k = 2$ receivers per transfer.

Proof: We use a variant of the 3-dimensional matching problem for our proof, which is NP-hard as well [30]: Given a set $M \subseteq X \times Y \times Z$, where X, Y, Z are disjoint sets of q elements each. Does M contain a subset (matching) M' s.t. $|M| = q$ and no two elements in M' share a coordinate?

Observe that we can cast (in polynomial time) each such instance as a multicast matching problem, which will complete the proof. To this end, we translate M into transfers by setting X as the sources and Y, Z as the receiver nodes. Now, if q such transfers can be admitted simultaneously, it directly translates to a set M' of q elements and vice versa. ■

Moreover, restricting the number of transfers per source turns the problem tractable, but only to a certain degree:

Theorem 2.2: If each source node appears in at most a single transfer, the multicast matching problem is polynomial-time solvable for $k = 2$ receivers and NP-hard for every $k > 2$.

Proof: We begin with $k = 2$. Observe that if we want to admit a specific transfer, it will only block the usage of the receiver nodes. The source can never be a conflict, as it only takes part in at most one transfer. Hence, we can leverage standard matching algorithms [29] between the receivers to obtain a maximum result. We investigate $k > 2$ next.

To this end, we consider the case of $k = 3$, which is contained in every $k > 2$. We use the problem of hypergraph matching for our proof, which is NP-hard to maximize even when all edges contain exactly 3 nodes [27]. Similarly to the proof of Theorem 2.1, we cast the hypergraph matching as a multicast matching, this time with each transfer having exactly 3 receivers, where the source is ignored as for $k = 2$. ■

On the other hand, allowing for splittable multicast matchings expands the number of possible receivers beyond $k = 2$:

Theorem 2.3: If each source node appears in at most a single transfer, the splittable multicast matching problem is polynomial-time solvable for any value of k receivers.

Proof: For this proof we rely on a technique commonly used for normal matchings in bipartite graphs [31]: by adding a super-source (connected to all sources) and a super-sink (connected from all receivers), the bipartite matching problem can be cast as a directed max-flow problem, where the single-source/sink property guarantees the existence of an integral solution. As all edge capacities are unit size, the flow between sources and receivers corresponds to a maximum matching.

We can directly adapt this technique to splittable multicast matchings. To this end, we adapt each transfer from a source s_i to k_i receivers as follows: The edge from the super-source to s_i has a capacity of k_i and the transfer edge is cast as k_i unit capacity edges, directed from s_i to each of the k_i receivers.

In this setting, the integral max-flow solution guarantees that each receiver is only part of one admitted transfer, whereas the capacity of k_i to s_i guarantees that any subset of the single transfer containing s_i can be served. ■

III. Problem, Approach and Challenges

Given the first insights in §II, we now formulate the general multicast scheduling problem considered in this paper.

A. Problem Statement

We consider a hybrid datacenter including a reconfigurable circuit switching network and a packet switching network, recall Fig. 1. We focus on the former that is enabled by the high-bandwidth circuit switch, as in Blast [22] and Creek [23], and consider multicast flows that arrive over time and can be of arbitrary size. Our main goal is to minimize *flow time*, but we will also consider a throughput maximization objective.

In our reconfigurable network model, every ToR is directly connected to the circuit switch via an exclusive port. The bandwidth of a circuit switch port is generalized to b_c, which is typically multiple times the bandwidth b_s of the link between a server and a ToR. For example, b_c can be 10Gbps, 40Gbps, 100Gbps and beyond while b_s is 10Gbp [13, 22, 23].

The circuit switch can constantly reconfigure high-speed port-to-multiport circuit connections between ToRs, but comes with a reconfiguration delay and stops transmitting data during the reconfiguration period. Any two circuit connections can share neither sender port nor receiver ports [8, 13, 22, 23].

A multicast flow f is characterized by a tuple $(s_f, \boldsymbol{d}_f, v_f, t_f^{\text{arr}})$, where s_f, $\boldsymbol{d}_f$, v_f, and t_f^{arr} denote the sender rack, the *set* of receiver racks, the data volume, and the release time, respectively. By taking advantage of the application knowledge, the multicast group membership and traffic volume is available upon flow arrival [22, 32]. We assume segregated routing model in this work, where a flow routed using either a single-hop circuit switching or multihop packet switching, but not a combination of both [9, 33]. Such a single-hop segregated routing avoids moving traffic between the circuit and the packet switching network.

The question is to decide which circuit connections to create and which flows to transfer over these connections, optimizing the objective and accounting for the circuit switch constraints.

B. Scheduling Approach: Splittable and Preemptive Multicast

Flows are scheduled epoch by epoch, where in each epoch the circuit connections and the set of serving flows are fixed. At the beginning of an epoch, the circuit connections are determined as well as the flows to be served in this epoch. We also have to determine the epoch duration in order to properly trade off reconfiguration overhead against sub-optimal configurations. Overall, we adopt splittable and preemptive transfers.

We briefly explain the key idea of our solution and its advantages over only splittable respectively only preemptive solutions in Fig. 3. Using splittable transfers, the scheduling algorithm may transfer data to just a subset of the receivers of a multicast flow in an epoch. For example, for a flow f_2, the circuit switch may transfer data to its receiver 1 in the first epoch and its receivers 3, 6 in the second epoch, respectively, as shown in Fig. 3(b). Additionally, a circuit connection can transfer multiple matched flows to fully use the capacities.

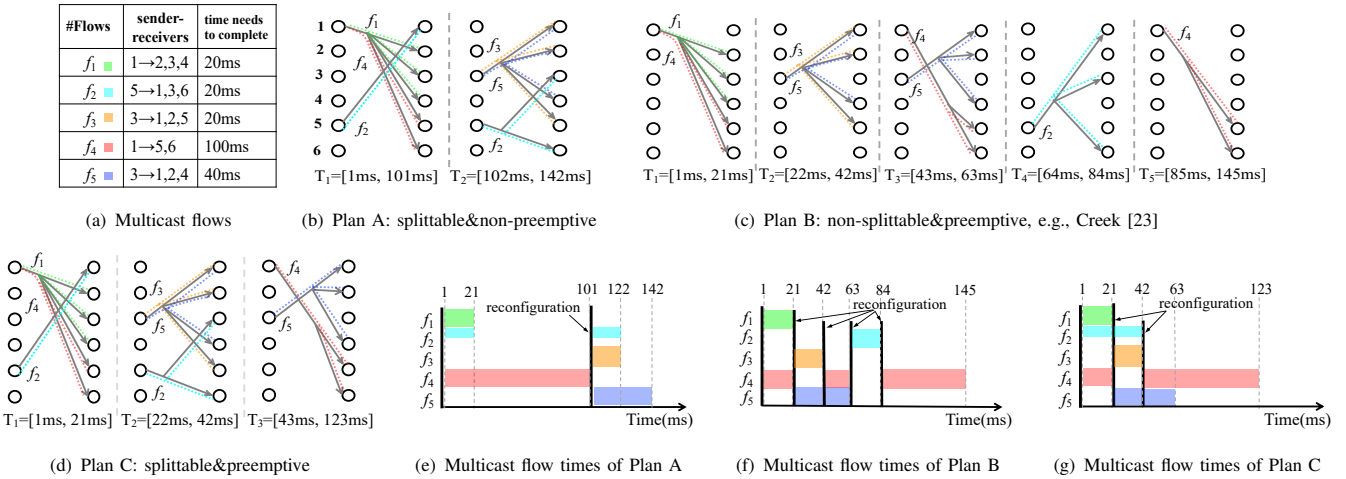
As the bandwidth (b_c) of per circuit switch port is assumed to be twice the fan-in rate (b_s) of each flow, f_1 and f_4 are simultaneously transferred over a circuit connecting inport 1 and outports 2 to 6 in the first epoch in Fig. 3(b). Therefore, f_1 also reaches the non-receiver ToRs via outports 5, 6. However, it is necessary to prevent the non-receiver ToRs from further forwarding f_1 to non-destination racks for storage efficiency. To this end, we install ToR forwarding rules only for the flows with this rack destination. Hence, the ToRs connecting outports 5, 6 will directly discard the packets of f_1 due to no matching rule.

Using preemptive transfers, the scheduling algorithm may reconfigure the circuit connections and reallocate the connections to the most critical flows before the completion of serving flows. For example, the circuit switch can be reconfigured with a delay of 1ms and flow f_4 is preempted by f_3 and f_5 after the completion of f_1 shown in Fig. 3(c). With preemption, the average flow time (see Fig. 3(f)) is sped up 1.43× over the splittable but non-preemptive plan shown in Fig. 3(e).

Moreover, by combining splittable and preemptive scheduling (Fig. 3(d)), the average flow time (Fig. 3(g)) can be sped up 1.74× and 1.22× over the only splittable plan (Fig. 3(b)) and the only preemptive plan (Fig. 3(c)), respectively.

C. Formulations

We now present our formulation for the splittable multicast problem in an epoch, using the notations shown in Table I.



(a) Multicast flows (b) Plan A: splittable&non-preemptive (c) Plan B: non-splittable&preemptive, e.g., Creek [23]

(d) Plan C: splittable&preemptive (e) Multicast flow times of Plan A (f) Multicast flow times of Plan B (g) Multicast flow times of Plan C

Fig. 3. Preemptive and splittable scheduling can significantly speed up the flow time, compared to the solutions that are just based on splittable or preemptive.

TABLE I
KEY NOTATIONS USED PROBLEM FORMULATIONS

Network model	
n	the number of all racks connecting to circuit switch
b_s	bandwidth of per server NIC port
b_c	bandwidth of per circuit switch port, $b_c \geq b_s$
δ	reconfiguration time of switch circuit
Multicast flow f	
s_f	the sender rack
$a_{f,i}$	indicator whether rack i is the sender rack
$\boldsymbol{d}_f$	the set of receiver racks
v_f	the (remaining) flow size
t_f^{arr}	the arrival time
Internal and decision variables for epoch t	
$x_{i,j}^t$	binary: indicate whether there is a circuit connection destining to rack j from rack i
w_f^t	binary: indicate whether flow f is to be scheduled
$w_{f,d}^t$	binary: indicate whether flow f transfers data to its receiver d
θ^t	the time duration of the concerned epoch

Herein, we determine the circuit connections, the flows to-be-scheduled and the length of a concerned epoch t. We first point out the constraints of building circuit connections and of scheduling flows and then formalize the objectives.

Constraints: Constraints (1) express that the output of a circuit connection can only receive traffic from a single input on the other side of the circuit. Constraints (2) constrict the transmission rate of flows from a rack i to the circuit should not exceed the circuit port bandwidth b_c. Constraints (3) state that a flow f could transfer data to its receiver rack $d \in \boldsymbol{d}_f$ via the circuit switch only if there is a circuit connection originating from the sender rack s_f to d. Constraints (4) together with (5) express that a flow f is to be scheduled as long as at least one of its receivers is to be served.

$$\forall j : \sum_i x_{i,j}^t \leq 1 \tag{1}$$

$$\forall i : \sum_f b_s a_{f,i} w_f^t \leq b_c \tag{2}$$

$$\forall f, d \in \boldsymbol{d}_f : w_{f,d}^t \leq x_{s_f,d}^t \tag{3}$$

$$\forall f : w_f^t \leq \sum_{d \in \boldsymbol{d}_f} w_{f,d}^t \tag{4}$$

$$\forall f, d \in \boldsymbol{d}_f : w_f^t \geq w_{f,d}^t \tag{5}$$

Objectives: A most fundamental objective is to maximize the average throughput

$$\max g(\boldsymbol{w}^t, \theta^t) = \frac{\sum_f \sum_{d \in \boldsymbol{d}_f} \min(v_{f,d}^t, b_s \theta^t) w_{f,d}^t}{(\theta^t + \delta)} \tag{6}$$

$\sum_f \sum_{d \in \boldsymbol{d}_f} \min(v_{f,d}^t, b_s \theta^t) w_{f,d}^t$ is the total size of data transferred over the circuit switch in epoch t.

A second important objective is to minimize the flow times. As we are just planning epoch-by-epoch, we cannot know the exact flow times of unfinished flows. However, we could know and optimize the lower bound of the minimum flow times

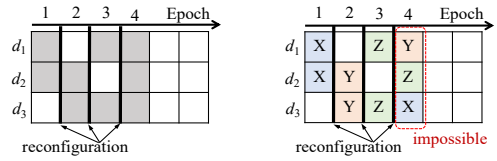
$$\min h(\boldsymbol{w}^t, \theta^t) = \sum_f \sum_{d \in \boldsymbol{d}_f} (I(v_{f,d}^t > b_s \theta^t w_{f,d}^t)(\theta^t + \delta) + I(v_{f,d}^t < b_s \theta^t w_{f,d}^t) \frac{v_{f,d}^t}{b_s} + t^{\text{start}} - t_f^{\text{arr}}) \tag{7}$$

where $I(v_{f,d}^t > b_s \theta^t w_{f,d}^t)(\theta^t + \delta)$ is the time experienced by the receiver d of flow f if it has not finished by the end of epoch t, and $I(v_{f,d}^t < b_s \theta^t w_{f,d}^t) \frac{v_{f,d}^t}{b_s}$ otherwise. t^{start} is the start time of epoch t and t_f^{arr} is the arrival time of flow f.

The above formulations are Integer Linear Programs (ILPs) for the given epoch length θ^t. However, if the epoch length is also subject to optimization, then the objective function becomes non-linear, introducing additional challenges (see next subsection).

D. Challenges

Even though the above formulation is linear, solving the resulting ILP can be time-consuming. Furthermore, we would like to exploit the flexibility of adapting the epoch lengths dynamically, and also schedule multicasts over time. However, we can see that the circuit connections $\boldsymbol{x}^t$, the flow scheduling decisions $\boldsymbol{w}^t$ and the epoch duration θ^t actually interact with each other and they together determine the network throughput and the sum of flow times. This renders it challenging to find an optimal solution. In addition to the above challenges in the optimization formulations, there is also a synchronization issue introduced, as we discuss in more details in the following.



(a) (b)

Fig. 4. The challenge of receiver asynchronization: All the receivers of a splittable flow have not yet received all requested data (see (b)) despite the equivalent volume of requested data (see (a)).

Let's consider an illustrative flow f_1 with three receivers d_1, d_2 and d_3 that request three units of data. Assuming that the circuit switch has unit capacity per port and can only build a circuit connecting two of the three receivers in the first three epochs with unit time length, due to one port that has been taken by other circuit connections. In particular, considering that a unit of data has been transferred to (d_1, d_2), (d_2, d_3), and (d_1, d_3) in the first, the second, and the third epoch, respectively, as shown in Fig. 4(a). When it comes to the fourth epoch, the circuit switch now can create a circuit connecting all the receivers. Seemingly, f_1 could be completed as d_1, d_2, d_3 will receive three units of data, the total requested data size, at the end of the fourth epoch.

However, f_1 cannot been completed because two receivers did not receive the data they request in the last epoch. Assume that the requested data is {X, Y, Z} (each has a unit size) and that X, Y and Z are transferred in the first, the second and third epoch, respectively, as illustrated in Fig. 4(b). In the fourth epoch, as the circuit can only transfer either X or Y or Z, two receivers cannot receive their lastly requested data. We refer to this as the receiver asynchronization problem.

When scheduling splittable flows, one should be careful to handle receiver asynchronization to ensure that every multicast receiver obtains all requested data, which is challenging in general. A naive solution is to partition the receivers of a flow into multiple non-neighboring subsets, consider each subset of receivers together with the sender as a subflow and independently scheduled these subflows without allowing further splitting during the transmission. However, such a premature fixed partition cannot adapt to the traffic dynamics, and it is difficult to determine an optimal partition in advance [34].

IV. SplitCast: Optimcal Multicast Scheduling over Reconfigurable Networks

Given the motivation and challenges discussed above, we are now ready to present details of our scheduler, SplitCast. SplitCast solves the multicast scheduling problem in reconfigurable datacenter networks, in an online and efficient manner.

Overview. In a nutshell, SplitCast works in an epoch by epoch manner. At the beginning of an epoch, SplitCast creates the circuit connections, chooses the flows to be transferred and determines the time duration of the epoch. SplitCast chooses the to-be-served flows and creates circuit connections in a hierarchical fashion, see the pseudo-code in Alg. 1. It firstly picks the flows where all receivers can be served without splitting and creates the circuit connections that match them (line 1, Alg. 1), and then determines the epoch duration according to the picked flows (line 2, Alg. 1). Subsequently, SplitCast searches for more flows where subsets of their receivers can be served by employing or extending the created circuit connections (line 3, Alg. 1).

A. Creating Circuit Connections and Scheduling Flows

In our algorithm, a circuit configuration is modeled as a directed hypergraph H, where each node denotes a rack and each directed hyperedge denotes a circuit connection, as in [22, 23]. The creation of circuit connections is modeled as adding directed hyperedges (without sharing tail node and head nodes) to the hypergraph H. Initially, the hyperedge set is empty.

Given a set F of multicast flows, we consider flows in a shortest remaining processing time first (SRPT) manner to determine a subset of flows where all their receivers can be matched by creating circuit connections under the circuit switch constraints. We consider flows and create circuit connections in two rounds. In the first round, we only consider the multicast flows with all receivers (line 9-12, Alg. 1). In the second round, we consider the subflows of multicast flows (line 13-16, Alg. 1), where subflows are products of splitting flows, and a subflow includes a subset of receivers, as explained later.

The scheduler considers a flow to be servable if the following two conditions are satisfied: i) the number of to-be-served flows that use the hyperedge originating from its sender is less than $\frac{b_c}{b_s}$ (line 2, Alg. 2), ii) all the receivers of this flow are included in the hyperedge originating from its sender (line 13, Alg. 2). Once such a flow is found, a directed hyperedge is added from the sender to all the receivers if no hyperedge originates from the sender (line 19, Alg. 2). Otherwise, the directed hyperedge originating from the sender is extended to include the unconnected receivers (line 21, Alg. 2).

B. Calculating The Epoch Length

We determine the epoch length θ according to the created circuit connections (via the hypergraph H) and the to-be-served unsplittable flows found in the last step. Given the set of to-be-scheduled flows ($\boldsymbol{w}^t$ is known), the network throughput function $g(\boldsymbol{w}^t, \theta^t)$ and the flow time function $h(\boldsymbol{w}^t, \theta^t)$ can be proven to have unique extreme values [23]. The optimal epoch length θ is related to the completion times of the flows to be served. Thus, we enumerate all possible epoch lengths and pick the one that achieves the optimal value of (6) or (7). After determining the θ, we know whether a flow can be completed in the epoch because every to-be-served flow can send up to θb_s of data in the epoch.

C. Scheduling Splittable Flows

In order to fully use the remaining circuit capacity, we further schedule flows for which subsets of receivers can still be served (line 15, Alg. 2). Recall the example flow f_1 in Fig. 4, where the switch could only send a unit of data to d_1 and d_2 in the first epoch. At the end of this epoch, our solution will decrease the remaining size of f_1 by one and

Algorithm 1 Splittable Multicast Scheduling Algorithm

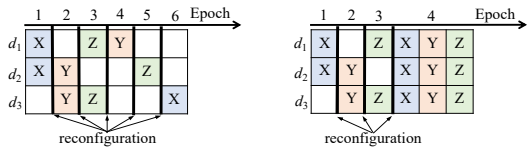
Input: A set F of flows to be scheduled in an epoch;
Output: A hypergraph H of the circuit configuration, a set F^{serve} of to-be-served flows and the epoch duration θ;
1: $(H, F^{\text{serve}}) \leftarrow$ NONSPLITSCHEDULE(F);
2: $\theta \leftarrow$ CALCULATEEPOCHLENGTH(F^{serve}, δ);
3: $(H, F^{\text{serve}}_{\text{split}}) \leftarrow$ SPLITSCHEDULE($H, F \setminus F^{\text{serve}}, \theta$);
4: $F^{\text{serve}} \leftarrow F^{\text{serve}} \cup F^{\text{serve}}_{\text{split}}$;

5: **procedure** NONSPLITSCHEDULE(F)
6: $F^{\text{serve}} \leftarrow \varnothing$;
7: F.order($policy$ = SRPT);
8: Initialize a hypergraph H to include the nodes of all racks and an empty hyperedge set;
9: **for** $f \in F$ **do** ▷ Check every multicast f
10: $schedule \leftarrow$ CREATECIRCUIT(H, f, *False*);
11: Add f to F^{serve} if $schedule$ is *True*;
12: **end for**
13: **for** $(f_s, f) \in F$ **do** ▷ Check every subflow f_s of each multicast f
14: $schedule \leftarrow$ CREATECIRCUIT(H, f_s, *False*);
15: Add f_s to the to-be-served subflow list of f and add f to F^{serve} if $schedule$ is *True*;
16: **end for**
17: **return** (H, F^{serve})
18: **end procedure**

19: **procedure** SPLITSCHEDULE(H, F', θ)
20: $F^{\text{serve}}_{\text{split}} \leftarrow \varnothing$;
21: F'.order($policy$ = SRPT);
22: **for** $f \in F'$ **do**
23: **try** CONSOLIDATESUBFLOW(SF_f); ▷ SF_f stores all unfinished subflows of f
24: **for** $f_s \in SF_f$ **do**
25: $schedule \leftarrow$ CREATECIRCUIT(H, f_s, *True*);
26: **if** $schedule$ **then**
27: Add f_s to the to-be-served subflow list of f;
28: $F^{\text{serve}}_{\text{split}}$.add($f$);
29: Create a subflow f'_s if f has unserved receivers;
30: **try** MERGESUBFLOW(SF_f, f'_s);
31: **end if**
32: **end for**
33: **end for**
34: **return** (H, $F^{\text{serve}}_{\text{split}}$)
35: **end procedure**

Algorithm 2 Create circuit connections

1: **procedure** CREATECIRCUIT(H, f, $split$)
2: **if not** H.hasfreeCapacity(s_f) **then**
3: **return** H, *False*
4: **end if**
5: $r^{list}_{cover} \leftarrow \varnothing$, $r^{list}_{outlier} \leftarrow \varnothing$;
6: **for** $d \in \boldsymbol{d}_f$ **do**
7: **if** H.predecessor(d) = s_f **then**
8: r^{list}_{cover}.add(d);
9: **else if** H.inDegree(d) = 0 **then**
10: $r^{list}_{outlier}$.add(d);
11: **end if**
12: **end for**
13: **if not** $split$ and len($r^{list}_{cover} + r^{list}_{outlier}$)< $|\boldsymbol{d}_f|$ **then**
14: **return** *False*
15: **else if** $split$ and len($r^{list}_{cover} + r^{list}_{outlier}$)= 0 **then**
16: **return** *False*
17: **end if**
18: **if** H.outDegree(s_f)= 0 **then**
19: H.addHyperedge(s_f, $r^{list}_{outlier}$);
20: **else**
21: H.extendHyperedge(s_f, $r^{list}_{outlier}$);
22: **end if**
23: H.decreaseCapacity(s_f);
24: $\boldsymbol{d}^{\text{unserved}}_f \leftarrow \boldsymbol{d}_f \setminus \left(r^{list}_{cover} \cup r^{list}_{outlier}\right)$;
25: **return** *True*
26: **end procedure**



(a) Possible scheduling solution A (b) Possible scheduling solution B

Fig. 5. Subflow scheduling

create a subflow f_{11} with the unserved receiver d_3: the flow size equals to the amount of untransmitted data in this epoch. The algorithm works similarly in the second and the third epoch: it decreases the remaining size of f_1 by one and creates a subflow f_{12} with the unserved receiver d_1 and f_{13} with the unserved receiver d_2, respectively. In the following epochs, subflows are scheduled independently with other flows, which is easy to operate. In addition, if a newly created subflow has the same receivers as other unfinished subflows, it is natural to merge them into a larger one: the resulting flow size equals to the sum of the sizes of the merged subflows (line 30, Alg. 1).

D. Complexity Analysis and Discussion

We first analyze the complexity of our algorithm and then discuss an opportunity for further performance improvement.

The computation time of Alg. 1 depends on the time to schedule non-splittable flows, the time to compute the epoch length, and the time to schedule the splittable flows. They have a time complexity of $\mathcal{O}(m \log m + mn + mn^2)$, $\mathcal{O}(m)$, $\mathcal{O}(m \log m + mn^2)$, respectively, if the algorithm schedules m flows in a network containing n racks. In total, the computation complexity of our scheduling algorithm is therefore $\mathcal{O}(m \log(m) + mn^2)$.

While our algorithm performs well as we will see in the evaluation section, it offers an interesting opportunity to further improve the transfer performance through subflow consolidation when scheduling splittable flows. Recall that the multicast flow f_1 in Fig. 4 has three subflows, f_{11}, f_{12} and f_{13}, at the end of the third epoch. If the fourth epoch has only one unit of time, it is impractical to simultaneously finish f_{11}, f_{12} and f_{13} over a circuit connection between the sender and all receivers due to the receiver asynchronization problem. The circuit switch has to schedule these subflows one by one and reconfigures the circuit connections for scheduling each of them, as shown in Fig. 5(a). However, if the fourth epoch lasts for three units of time, we could consolidate these subflows as a larger one and finish them over the circuit connection to all their receivers without more reconfigurations, as shown in Fig. 5(b). In our algorithm, we use a greedy subflow consolidation method (line 23, Alg. 1). For every considered multicast flow,

we first determine which of its receivers could be matched, if one more hyperedge is added subject to the capacity constraints. Then, we greedily consolidate the subflows which can be matched by this hyperedge, and we stop before the remaining sizes of the consolidated subflows exceeds θb_s.

V. Evaluation

We conduct extensive simulations to study the performance of SplitCast in different scenarios and to compare our results to that of the state-of-the-art. In the following, we first introduce our setup in §V-A, and then discuss our results in §V-B. We find that SplitCast can significantly improve the performance of multicast transfers across multiple networks and workloads.

A. Evaluation Setup

Our evaluation setup largely follows [22, 23].

Network topologies: We run simulations over four typical datacenter sizes, which have 32, 64, 128, and 256 racks, respectively. According to recent circuit switch designs, the bandwidth per circuit switch port is chosen from 10Gbps, 40Gbps, and 100Gbps, and the circuit reconfiguration time ranges from 0.1ms to 100ms. The bandwidth between the server and the ToR switch is 10Gbps.

Workloads: We use synthetic multicast traffic resulting from real datacenter applications similar to related work [13, 23]. The distribution of data sizes (in GB) follows a beta distribution B(0.7, 1.7). The sender and the receivers of every multicast flow are randomly chosen from the racks in the network. The number of receiver racks follows a uniform distribution $U[2, n\gamma]$, where n is the total number of racks in a network and γ is chosen from $[10\%, 20\%, 30\%]$. The total simulation time T of every experiment ranges from 5,000ms to 10,000ms and flows uniformly arrive between 1ms and $\frac{T}{10}$ms.

Compared approaches: We compare SplitCast against the state-of-the-art approaches for scheduling multicast demands using a reconfigurable circuit switch.

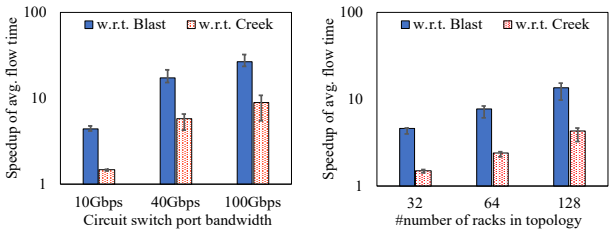
- Blast [22] performs non-preemptive scheduling and iteratively schedules the flows in an decreasing order of a "score" defined by ($\frac{size}{\#receivers}$). The latest flow from a set of flows that can be simultaneously transferred in an epoch determines the epoch duration.
- Creek [23] adopts preemptive scheduling, uses the SRPT policy to schedule flows, and chooses the epoch duration that can maximize the circuit switch utilization. We first let Creek use the 1-hop segregated routing model, but will later show how SplitCast compares when Creek may use a multi-hop routing model as well (see Fig. 10).

As in Blast [22] and Creek [23], we focus on the circuit switching network and hence the performance of flows delivered by it. We conduct at least 100 runs for every experiment setting over each network and report the average results (e.g., flow time, throughput) below unless otherwise specified.

B. Evaluation Results

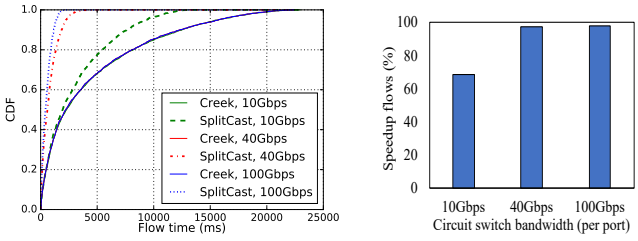
The impact of circuit port bandwidth: In this group of experiments, we evaluate the impact of circuit port bandwidth by varying it from 10Gbps to 100Gbps. The receiver fraction γ is fixed to 10% and the circuit reconfiguration time is 0.1ms in these experiments. Fig. 6 shows the speedup of the average flow time of SplitCast in log scale. We can see from Fig. 6(a) that the speedup increases with an increasing circuit switch port bandwidth in the topology with 256 racks. We also see that SplitCast outperforms Blast and Creek in reducing the average flow time by up to a factor of 27× and 9×, respectively. Additionally, our experiments for three other simulated topologies also show a similar trend. We here present only the experimental results (see Fig. 6(b)) of these topologies obtained under 40Gbps port bandwidth due to the space constraints. We can see that SplitCast also speeds up the average flow time over Blast and Creek in these topologies and that the speedup increases as the topology size grows. Thanks to the preemptive scheduling, both Creek and SplitCast outperform Blast. However, Creek cannot beat SplitCast as Creek only transfers data when the circuit connections can match all the receivers of a multicast flow, even through a subset of receivers could be matched. In contrast, SplitCast allows data to be transferred to partially matched receivers, fully using the circuit capacity and achieving the shortest average flow time.

Fig. 7 reports more detailed results obtained from the

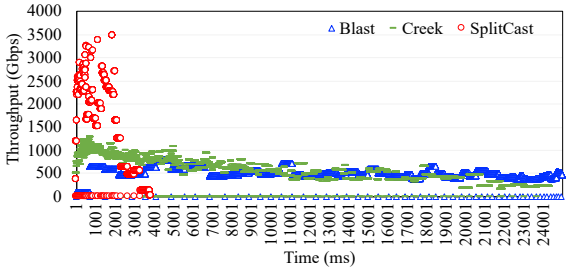


(a) Network with 256 racks (b) 40Gbps circuit port bandwidth

Fig. 6. Impact of the circuit switch port bandwidth. (a), (b) show the speedup of the average flow time of SplitCast to Blast and to Creek over all runs.

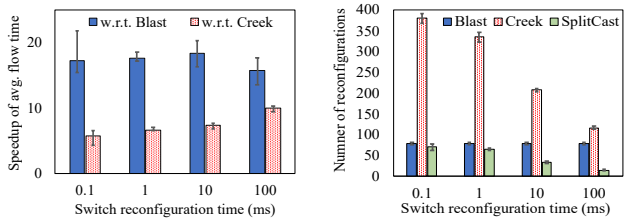


(a) Flow time (b) Flows experience speedup



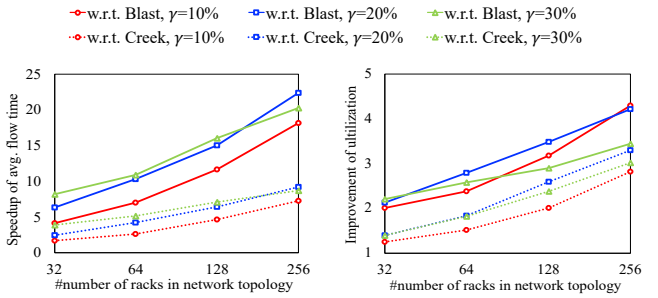
(c) Throughput over time

Fig. 7. (a)-(c) show the CDF of the flow time, the percentage of flows experiencing speedup, and the throughput over time in the 256 rack topology.



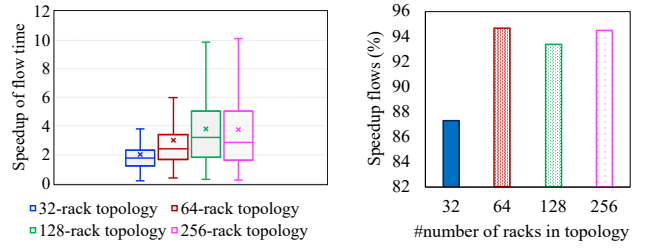
(a) Speedup of average flow time (b) Number of circuit reconfigurations

Fig. 8. Impact of the reconfiguration time of circuit switch. (a-b) show the maximum-average-minimum speedups of average flow time and the number of reconfigurations of SplitCast over all runs in the 256-rack topology. The simulation results show that the improvement of SplitCast in flow time is quite stable for all common switch reconfiguration times.



(a) Speedup of average flow time (b) Improvement of circuit utilization

Fig. 9. Impact of the number of receivers. (a) and (b) show the speedup of the average flow time and the improvement of the switch utilization, respectively, in experiments where γ of racks are the receivers of multicast flows.



(a) Speedup of flow time (b) Flows experience speedup

Fig. 10. SplitCast even outperforms Creek when Creek does m-hop circuit routing, on average speeding up the flow time 2× to 4×.

topology with 256 racks. Fig. 7(a) shows that SplitCast reduces the flow time of the latest flow from around 20s to less than 5s when the circuit port bandwidth is 40Gbps and 100Gbps, and Fig. 7(b) shows that around 68%-98% of the flows experience speedups, compared to Creek. This is consistent with the results in Fig. 6. Fig. 7(a) also shows that Creek obtains very close flow times under different circuit port bandwidth, which indicates that it cannot efficiently use the high-bandwidth circuit capacity. Fig. 7(c) shows the throughput over time (averaged in every epoch) in one experiment simulated with 40Gbps circuit port bandwidth. SplitCast achieves the highest throughput in the early stage, Creek comes second and Blast follows. The high throughput of SplitCast is also related to its flow time improvements.

The impact of circuit switch reconfiguration time: As the circuit switch will stop transmitting data during circuit reconfiguration, we evaluate how different reconfiguration times impact scheduling approaches in this group of experiments. Fig. 8(a) shows that the speedup of the average flow time of SplitCast over Blast and Creek is stable as the reconfiguration time increases from 0.1ms to 100ms over the 256-rack topology. We omit the presentations of the similar results obtained from other three topologies due to the limited space. Additionally, Fig. 8(b) shows that SplitCast has the minimum number of reconfigurations, Blast comes second, and Creek is last. The very small number of reconfigurations of SplitCast also indicates lower operating cost, compared to Creek and Blast. Additionally, the average number of reconfigurations of SplitCast and that of Creek drop as the reconfiguration time increases. In contrast, the number of reconfigurations of Blast is unadapted to reconfiguration times. This is as expected because SplitCast and Creek consider the reconfiguration time when determining the epoch length, while Blast does not.

The impact of the number of receivers: In this part, we evaluate how the receiver scale impacts the performance of the scheduling approaches. To this end, we set the number of receivers of every multicast to be different percentages γ of the racks and vary γ from 10% to 30%. Fig. 9(a) shows the speedups of the average flow time of SplitCast over Blast and Creek. We can see that the speedup of the average flow time almost linearly increases with an increasing number of receivers and topology size. In addition, we collect the circuit utilization which is defined as a ratio: the total data size actually transferred, compared to the theoretical total data size that can be delivered with full capacity of the circuit switch for every epoch. We compare the average circuit utilization over all epochs. Fig. 9(b) shows that SplitCast improves the average circuit utilization up to 4.3× and 3× over Blast and Creek, respectively. Also, the improvement of the average circuit utilization increases as the network size scales up.

Additionally, we count the number of subflows that SplitCast creates for every multicast flow in these experiments. The experimental results show that SplitCast splits only 55% of multicast flows with three receivers on average and creates just one additional subflow for mutlicast flows with an average of three to nine receivers in 32-rack network. Also, SplitCast creates two more subflows for 30%-35% flows and at most four subflows for 80% flows in 256-rack network. The small number of subflows created indicates that the cost of maintaining the state of subflows is low or negligible.

SplitCast vs. Creek with multi-hopping: In the last group of experiments, we evaluate how our 1-hop solution SplitCast compares against the multi-hop version of Creek. Fig. 10 shows that SplitCast can still outperform Creek, speeding up the flow times for around 87% to 94% flows and achieving 2× to 4× speedups on average. These results indicate the great potential of splittable multicast for improving the performance of multicast transfers in reconfigurable datacenter networks. Also, it motivates us to exploit splittable multicast in multi-hop routing enabled reconfigurable networks in future work.

VI. Related Work

Multicast communication in datacenter networks: Multicast is a typical communication pattern of many datacenter applications, such as the data dissemination of publish-subscribe

services [18], distributed caching infrastructures updates [21] and state machine replication [35]. Multicast communications are growing explosively in scale due to the proliferation of applications based on big data processing frameworks [36]–[38]. In order to improve the performance of multicast transfers, many works turn to IP multicast and focus on issues like scalability and reliability [12, 39, 40] of the deployment of IP multicast in the datacenter context. These above proposals consider multicast over static networks.

Reconfigurable datacenter networks: One of the recent technology innovations in networking research is the possibility of providing reconfigurable high-bandwidth inter-rack connections at runtime, with reconfigurable circuit switches. Importantly, such technology also supports high-performance inter-rack multicasting. This capability of data multicast is enabled by circuit switching technologies [13, §I], e.g., optical circuit switches (OCS) [22, 23], free-space optics (FSO) [9, 41], and 60 GHZ wireless links [42]–[44].

This work is in turn motivated by such novel circuit switching technologies and focuses on scheduling algorithms for multicast flows. The most related works are Blast [22] and Creek [23]. Both transfer data only when there exists a circuit connection matching all the receivers of a multicast flow. In contrast, we also allow a flow to transfer data when a circuit connection matches only a subset of its receivers.

Such splitting has also been studied by Sundararajan et al. [28] for optical switches. The authors extend the Birkhoff-von Neumann strategy to multicast switching, and investigate its complexity. Their main focus is on rate regions and achievability of rate requirements, whereas we focus on minimizing flow times via scheduling.

There also emerge further several works [10, 33, 45]–[49] on demand-aware reconfigurable datacenter networks. Schmid *et al.* [10, 33, 45]–[47] focus on designing topologies towards better network-wide performances, and Salman *et al.* [48] and Wang *et al.* [49] focus on learning the topology, whereas Xia et al. [50] adapt between Clos and approximate random graph topologies. They focus on unicast flows and are all orthogonal to this work. We refer to [11] for a recent survey.

Reconfigurable wide area networks: Wide area networks have also benefited from programmable physical layers, by e.g. leveraging reconfigurable optical add/drop multiplexers (ROADMs). Various networking research directions have been investigated, such as scheduling and completion times [51]–[53], robustness [54], abstractions [55], connectivity as a service [56], and variable bandwidth links [57, 58]. Herein benefits of multicast were recently studied by Luo et al. [59], in the context of bulk transfers.

VII. Conclusion

This paper studied the multicast scheduling problem in datacenter networks that are capable of high-bandwidth circuit switching and fast reconfigurations at runtime. We first discussed the unexploited potential of splittable multicast and analyzed the algorithmic complexity of splittable multicast matching. We proposed a scheduler, SplitCast, which relies on simple single-hop segregated routing and minimizes the flow times by leveraging the potential of splittable multicast, preemptive scheduling, and circuit switch reconfiguration. SplitCast employs a simple but fast algorithm that jointly optimizes both the flows and the circuit configuration schedules. Our extensive simulations on real-world topologies show that SplitCast significantly reduces the flow time and improves the throughput over the prior solutions.

We understand our work as a first step and believe that it opens several interesting avenues for future research. For example, it would be interesting to account for more specific application-level objectives, e.g., flow deadlines.

Acknowledgements. We would like to thank Xiaoye Steven Sun for his support regarding prior work [22, 23]. We would like to thank the reviewers for their comments, especially for pointing us to the work of Sundararajan et al. [28]. This project has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement No. 864228, AdjustNet: Self-Adjusting Networks). This work was also supported by the 111 Project (B14039) and the project PCL Future Greater-Bay Area Network Facilities for Large-scale Experiments and Applications (PCL2018KP001).

References

[1] W. M. Mellette, R. McGuinness, A. Roy *et al.*, "Rotornet: A scalable, low-complexity, optical datacenter network," in *SIGCOMM*, 2017, pp. 267–280.
[2] W. M. Mellette, R. Das, Y. Guo *et al.*, "Expanding across time to deliver bandwidth efficiency and low latency," in *NSDI*, 2020.
[3] Q. Cheng, M. Bahadori, M. Glick *et al.*, "Recent advances in optical technologies for data centers: a review," *Optica*, vol. 5, no. 11, pp. 1354–1370, 2018.
[4] A. Valadarsky, G. Shahaf, M. Dinitz, and M. Schapira, "Xpander: Towards optimal-performance datacenters," in *CoNEXT*, 2016.
[5] C. Guo, G. Lu, D. Li *et al.*, "BCube: a high performance, server-centric network architecture for modular data centers," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4, pp. 63–74, 2009.
[6] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey, "Jellyfish: Networking data centers randomly," in *NSDI*, 2012.
[7] D. Wu, W. Wang *et al.*, "Say no to rack boundaries: Towards a reconfigurable pod-centric dcn architecture," in *SOSR*, 2019.
[8] D. Wu, X. Sun, Y. Xia *et al.*, "Hyperoptics: A high throughput and low latency multicast architecture for datacenters," in *HotCloud*, 2016.
[9] M. Ghobadi, R. Mahajan, A. Phanishayee *et al.*, "Projector: Agile reconfigurable data center interconnect," in *SIGCOMM*, 2016, pp. 216–229.
[10] K.-T. Foerster, M. Pacut, and S. Schmid, "On the complexity of non-segregated routing in reconfigurable data center architectures," *ACM SIGCOMM Computer Communication Review*, vol. 49, no. 2, pp. 2–8, 2019.
[11] K.-T. Foerster and S. Schmid, "Survey of reconfigurable data center networks: Enablers, algorithms, complexity," *ACM SIGACT News*, vol. 50, no. 2, pp. 62–79, Jul. 2019.
[12] M. Shahbaz, L. Suresh, J. Rexford *et al.*, "Elmo: Source-routed multicast for cloud services," in *SIGCOMM*, 2019.
[13] X. S. Sun, Y. Xia, S. Dzinamarira *et al.*, "Republic: Data multicast meets hybrid rack-level interconnections in data center," in *ICNP*, 2018.
[14] L. Mai, C. Hong, and P. Costa, "Optimizing network performance in distributed machine learning," in *HotCloud*, 2015.
[15] "Deploying secure multicast market data services for financial services environments," https://www.juniper.net/documentation/en_US/release-independent/nce/information-products/pathway-pages/nce/nce-161-deploying-secure-multicast-for-finserv.html, accessed: 2019-07-31.

[16] "Trading floor architecture," https://www.cisco.com/c/en/us/td/docs/solutions/Verticals/Trading_Floor_Architecture-E.html, accessed: 2019-07-31.

[17] VMWARE, "Nsx network virtualization & security software," https://www.vmware.com/products/nsx.html, accessed: 2019-07-31.

[18] GOOGLE, "Cloud pub/sub," https://cloud.google.com/pubsub/, accessed: 2019-07-31.

[19] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[20] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "Coded mapreduce," in *53rd Annual Allerton Conference on Communication, Control, and Computing*, 2015.

[21] "Cloud networking: Ip broadcasting and multicasting in amazon ec2," https://blogs.oracle.com/ravello/cloud-networking-ip-broadcasting-multicasting-amazon-ec2, accessed: 2019-07-31.

[22] Y. Xia, T. E. Ng, and X. S. Sun, "Blast: Accelerating high-performance data analytics applications by optical multicast," in *INFOCOM*, 2015.

[23] X. S. Sun and T. E. Ng, "When creek meets river: Exploiting high-bandwidth circuit switch in scheduling multicast data," in *ICNP*, 2017.

[24] J. Bao, D. Dong, B. Zhao *et al.*, "Flycast: Free-space optics accelerating multicast communications in physical layer," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 97–98, 2015.

[25] H. Wang, Y. Xia, K. Bergman *et al.*, "Rethinking the physical layer of data center networks of the next decade: Using optics to enable efficient*-cast connectivity," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 3, pp. 52–58, 2013.

[26] L. Chen, K. Chen, Z. Zhu *et al.*, "Enabling wide-spread communications on optical fabric with megaswitch," in *NSDI*, 2017.

[27] L. Lovász and M. D. Plummer, *Matching theory*. American Mathematical Soc., 2009, vol. 367.

[28] J. K. Sundararajan, S. Deb, and M. Médard, "Extending the birkhoff-von neumann switching strategy for multicast - on the use of optical splitting in switches," *IEEE Journal on Selected Areas in Communications*, vol. 25, no. S-6, pp. 36–50, 2007.

[29] J. Edmonds, "Paths, trees, and flowers," *Canadian Journal of mathematics*, vol. 17, pp. 449–467, 1965.

[30] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[31] A. Schrijver, *Combinatorial optimization: polyhedra and efficiency*. Springer Science & Business Media, 2003, vol. 24.

[32] M. Zaharia, M. Chowdhury *et al.*, "Spark: Cluster computing with working sets," in *HotCloud*, 2010.

[33] T. Fenz, K.-T. Foerster, S. Schmid, and A. Villedieu, "Efficient non-segregated routing for reconfigurable demand-aware networks," in *IFIP Networking*, 2019.

[34] M. Noormohammadpour, C. S. Raghavendra, S. Kandula, and S. Rao, "Quickcast: Fast and efficient inter-datacenter transfers using forwarding tree cohorts," in *INFOCOM*, 2018.

[35] D. R. Ports, J. Li, V. Liu *et al.*, "Designing distributed systems using approximate synchrony in data center networks," in *NSDI*, 2015.

[36] "Apache spark," http://spark.apache.org/.

[37] "Tensorflow," https://www.tensorflow.org/.

[38] "Apache tez," https://tez.apache.org/.

[39] Y. Vigfusson, H. Abu-Libdeh, M. Balakrishnan *et al.*, "Dr. multicast: Rx for data center communication scalability," in *EuroSys*, 2010.

[40] X. Li and M. J. Freedman, "Scaling ip multicast on datacenter topologies," in *CoNEXT*, 2013.

[41] J. Bao, D. Dong, B. Zhao *et al.*, "Flycast: Free-space optics accelerating multicast communications in physical layer," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 97–98, 2015.

[42] X. Zhou, Z. Zhang, Y. Zhu *et al.*, "Mirror mirror on the ceiling: Flexible wireless links for data centers," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 443–454, 2012.

[43] Y.-J. Yu, C.-C. Chuang, H.-P. Lin, and A.-C. Pang, "Efficient multicast delivery for wireless data center networks," in *The 38th Annual IEEE Conference on Local Computer Networks*, 2013.

[44] C. Shepard, A. Javed, and L. Zhong, "Control channel design for many-antenna mu-mimo," in *MobiCom*, 2015.

[45] C. Avin and S. Schmid, "Renets: Toward statically optimal self-adjusting networks," *arXiv preprint arXiv:1904.03263*, 2019.

[46] C. Avin, I. Salem, and S. Schmid, "Working set theorems for routing in self-adjusting skip list networks," in *INFOCOM*, 2020.

[47] K.-T. Foerster, M. Ghobadi, and S. Schmid, "Characterizing the algorithmic complexity of reconfigurable data center architectures," in *ANCS*, 2018.

[48] S. Salman, C. Streiffer, H. Chen, T. Benson, and A. Kadav, "Deepconf: Automating data center network topologies management with machine learning," in *NetAI@SIGCOMM*, 2018.

[49] M. Wang, Y. Cui, S. Xiao *et al.*, "Neural network meets dcn: Traffic-driven topology adaptation with deep learning," *Proc. of the ACM on Measurement and Analysis of Computing Systems*, vol. 2, no. 2, 2018.

[50] Y. Xia, X. S. Sun, S. Dzinamarira *et al.*, "A tale of two topologies: Exploring convertible data center network architectures with flat-tree," in *SIGCOMM*, 2017.

[51] X. Jin, Y. Li, D. Wei *et al.*, "Optimizing bulk transfers with software-defined optical WAN," in *SIGCOMM*, 2016.

[52] S. Jia, X. Jin *et al.*, "Competitive analysis for online scheduling in software-defined optical WAN," in *INFOCOM*, 2017.

[53] M. Dinitz and B. Moseley, "Scheduling for weighted flow and completion times in reconfigurable networks," in *INFOCOM*, 2020.

[54] J. Gossels, G. Choudhury, and J. Rexford, *IEEE/OSA Journal of Optical Communications and Networking*, vol. 11, no. 8, pp. 478–490, 2019.

[55] K.-T. Foerster, L. Luo, and M. Ghobadi, "Optflow: A flow-based abstraction for programmable topologies," in *SOSR*, 2020.

[56] R. Durairajan, P. Barford, J. Sommers, and W. Willinger, "Greyfiber: A system for providing flexible access to wide-area connectivity," *arXiv preprint arXiv:1807.05242*, 2018.

[57] R. Singh, M. Ghobadi, K.-T. Foerster *et al.*, "Run, walk, crawl: Towards dynamic link capacities," in *HotNets*, 2017.

[58] ——, "RADWAN: rate adaptive wide area network," in *SIGCOMM*, 2018.

[59] L. Luo, K.-T. Foerster, S. Schmid, and H. Yu, "Dartree: deadline-aware multicast transfers in reconfigurable wide-area networks," in *IEEE/ACM IWQoS*, 2019.