Published in IEEE Journal on Selected Areas in Communications ©2020 IEEE

Efficient File Dissemination in Data Center Networks with Priority-based Adaptive Multicast

Shouxi Luo, Hongfang Yu, Ke Li, Huanlai Xing

***Abstract*—In today's data center networks (DCN), cloud applications commonly disseminate files from a single source to a group of receivers for service deployment, data replication, software upgrade, and etc. For these group communication tasks, recent advantages of software-defined networking (SDN) provide bandwidth-efficient ways—they enable DCN to establish and control a large number of explicit multicast trees on demand. Yet, the benefits of data center multicast are severely limited, since there does not exist a scheme that could prioritize multicast transfers respecting the performance metrics wanted by today's cloud applications, such as pursuing small mean completion times or meeting soft-time deadlines with high probability.**

To this end, we propose PAM (Priority-based Adaptive Multicast), a preemptive, decentralized, and ready-deployable rate control protocol for data center multicast. At the core, switches in PAM explicitly control the sending rates of concurrent multicast transfers based on their desired priorities and the available link bandwidth. With different policies of priority generation, PAM supports a range of scheduling goals. We not only prototype PAM upon the emerged P4-based programmable switch with novel approximation designs, but also evaluate its performance with ns3-based extensive simulations. Results imply that PAM is ready-deployable; it converges very fast, has negligible impacts on coexisting TCP traffic, and always performs near-optimal priority-based multicast scheduling.

***Index Terms*—SDN, data center multicast, flow scheduling, P4.**

I. Introduction

ALTHOUGH IP multicast has been proposed more than two decades and is widely supported, it was rarely used in data center due to its high cost on routing construction and maintenance [1]–[3]. Fortunately, the situation is changing: recent advantages of software-defined networking (SDN) enable data center networks (DCNs) to set up and control a large number of explicit multicast trees in very efficient ways [2]–[4]. For instance, through controller-assisted address partitions and local multicast forwarding rule aggregations, X. Li and M. Freedman scale IP multicast to support thousands of multicast groups in data center [2]; likewise, by encoding the multicast tree information in each packet's header at the sender, while employing customized forwarding at switches, M. Shahbaz *et al.* enable DCNs to precisely control how packets would get duplicated and routed to their receivers on demand [3].

These advantages of data center multicast indeed open the door to achieve the widely-existing group communication tasks for cloud applications in more efficient ways. In modern data centers, cloud applications commonly need to disseminate files from a single source to groups of receivers. For instance, to deploy a new job or microservice, the cluster manager would distribute the corresponding docker image file(s) from the registry to all selected slave servers [5]; to improve the reliability of critical data, backup systems would replicate each data chunk to multiple nodes [6]; and to perform software or OS upgrades for a cluster, the operator would disseminate patch files to all involved hosts. Obviously, for these tasks, network-level multicast is the best solution as it naively saves network bandwidth and reduces server overheads [1]–[3].

Despite the construction of multicast tree in data center has been solved already, using multicast to achieve efficient file disseminations in DCNs is still non-trivial. Mainly, similar to the schedule of unicast [7]–[9], data center applications typically want their disseminations to complete in some prioritized orders, for a smaller average completion time or a smaller maximum lateness, or to meet some soft-real-time deadlines with high probability. Accordingly, they prefer priority-based bandwidth allocations, while the designs widely adopted by today's multicast protocols are far from optimal since they pursue TCP-alike fair sharing [10]–[12]. In contrast to the abundant research on the schedule of unicast transfers [7]–[9], to the best of our knowledge, no prior effort has looked into the priority-based schedule of multicast yet. Moreover, different from unicast flow, multicast transfers generally involve more than one receivers and they do not employ per-packet ACKs. Thus, all those schemes designed for prioritized unicast scheduling [7]–[9] can not deal with multicast.

To achieve priority-based multicast for file dissemination in modern data center, several critical challenges must be carefully addressed. Firstly, file dissemination transfers in data center commonly coexist with other time-sensitive traffic, making a link's bandwidth that multicast transfers can use varying with time; to make efficient use of bandwidth while adapting to traffic dynamic quickly, the proposed multicast scheduling scheme must perform adaptive rate allocations in RTT scales [7]. Secondly, current data centers might involve up to hundreds of thousands of servers and millions of concurrent transfers; to avoid the single point of failure and be scalable, the proposed scheme should adopt decentralized designs [13]. Thirdly, existing commodity switches only provide a very limited number of priority levels and there might be only one queue left for multicast [7], [9]; to be practical, the proposed design could not rely on specified hardware priority queues. Last but not least, current commodity switch does not support complex computations in dataplane; to be efficient and ready-

S. Luo, K. Li, and H. Xing are with Southwest Jiaotong University, Chengdu 611756, China (e-mail: {sxluo, keli, hxx}@swjtu.edu.cn). *(Corresponding author: Shouxi Luo)*

H. Yu is with University of Electronic Science and Technology of China, Chengdu 611731, China (e-mail: yuhf@uestc.edu.cn).

This version corrects the missing of Fig.12b, Fig.12c, and Fig.12d in JSAC.

deployable, the proposed scheme must be implementable and the introduced hardware overheads must keep small [14], [15].

In this paper, we address above challenges and design PAM (**P**riority-based, **A**daptive Multicast), a preemptive, decentralized, ready-deployable rate control protocol for data center multicast. At its core, PAM lets multicast transfers carry with scheduling headers specifying their desired priorities during the multicast; then, based on their priority orders and the real-time link loads, switches explicitly compute each transfer's sending rate for next epoch in a preemptive manner; finally, with the feedbacks from receivers, the multicast source knows at what rate it should send the data in next epoch. Indeed, with this design, by generating multicast transfer's desired priorities according to policies like *Smallest Remaining Size First* (SRSF), *Earliest Deadline First* (EDF), and even *First-come First-reserve* (i.e., FIFO), the system can minimize *the average completion time*, *the number of missed deadlines*, and *the maximum lateness* for data center file disseminations in decentralized manners, respectively.

To make PAM implementable on today's commodity switch, we not only simplify its computation of preemptive rate allocation based on observations,[1] but also reform the corresponding switch operations with novel hardware-based approximation designs (e.g., match-action table based division). To reduce the overheads of explicit rate controls, we let transfers that get rate *zero* generate probe packets adaptively, and configure receivers to trigger feedbacks only when i) the change of rate is larger than the configured threshold, or ii) previous rate feedbacks are likely to get lost. These optimizations make PAM implementable on the emerged P4-based programmable hardware [16], [17] and greatly reduce the scheduling overhead of PAM without deteriorating the performance.

We prototype PAM in both P4 and ns-3. Through extensive simulations, we find that PAM is flexible and efficient to achieve priority-based multicast. Even though switches make preemptive bandwidth allocations only for the most critical transfers, PAM is near-optimal, providing strong benefits over existing data center multicast mechanisms. For instance, on minimizing the average transfer completion times (or missed deadlines, respectively), the gap between PAM and its optimal/ideal variant is less than 3.7% (2.1%, respectively), outperforming the default fair sharing scheme up to 5.7× (75%, respectively). Moreover, PAM has negligible impacts on the completion of coexisting TCP flow and always reacts to network dynamic in RTTs.

In summary, the key contributions of this paper are:

- We design and implement PAM, a distributed, priority-based, preemptive scheduling layer for data center multicast, which can approximate a range of scheduling disciplines while reacting to network dynamic very quickly.
- We use novel hardware-based approximation designs to make PAM implementable on emerged P4-based programmable switches, enabling the schedule computation of PAM switch to run at line-rate.
- We build on PAM to implement multicast scheduling disciplines that minimize the average multicast completion times and missed deadlines. Extensive simulations show that PAM is near-optimal, outperforming the default fair sharing scheme up to 5.7× and 75%, respectively.

The rest of the paper is organized as follows. Section II first introduces the theory and design challenges of data center multicast scheduling, then overviews the design of PAM. Section III presents PAM in detail and Section IV implements it upon the emerged programmable hardware with approximations. After that, extensive simulations are presented in Section V. Finally, related work and conclusions follow in Section VI and Section VII, respectively.

II. Problem Analysis

In this section, we first introduce the model and optimal theory of file dissemination oriented multicast (§II-A), then analyze the practical limits and design challenges for achieving optimized multicast in current data center networks (§II-B), and finally overview our proposed solution (§II-C).

A. Problem in Theory

For file dissemination tasks, once their multicast trees are established, the remaining task is to dynamically adjust their multicast rates respecting the scheduling goal and the network workloads without overloading links. Consider that $\mathcal{F}$ is the set of file dissemination tasks, in which task i (i.e., transfer f_i) is to disseminate a file with volume v_i from the source node to a group of receivers via tree $\mathcal{T}_i$. Let st_i and ct_i be this transfer's start/appear date and completion duration, respectively. In some cases, the transfer might also have an expired time et_i, implying the duration time by which the dissemination should expire. We further let $r_i(t)$ be the sending rate of f_i at time t, which is determined by the slowest receiver, and let $C_{u,v}(t)$ be the available bandwidth for multicast at time t on link (u, v). Then, in theory, the problem of minimizing their total completion times and missed deadlines can be formulated as minimizing $\sum_{i:f_i\in\mathcal{F}} ct_i$ and $\sum_{i:f_i\in\mathcal{F}} [ct_i > et_i]$, respectively, as Program (1) shows.

$$\text{minimize} \quad \sum_{i:f_i\in\mathcal{F}} ct_i \quad \text{(or } [ct_i > et_i] \text{, respectively)} \tag{1a}$$

$$\text{subject to} \quad \int_{st_i}^{st_i+ct_i} r_i(t)dt = v_i, \forall f_i \in \mathcal{F} \tag{1b}$$

$$\sum_{i:(u,v)\in\mathcal{T}_i} r_i(t) \le C_{u,v}(t), \forall t, \forall (u,v) \tag{1c}$$

$$r_i(t) \ge 0, ct_i \ge 0, \forall (i,t) \tag{1d}$$

The reader might notice that, even though the route of multicast is much more complex than that of unicast, they do share the same mathematical model on the optimization of bandwidth allocations. Indeed, over the last decade, a lot of

[1] Ideally, PAM switch should allocate each link's capacity to active multicast transfers in descending of their priorities. However, such a procedure is too complex to run at line-rate, thus unimplementable in switch data plane [14], [15]. Instead, motived by the observation that, under priority-based scheduling, the most critical transfer generally occupies almost the entire bandwidth of bottleneck links [9], PAM performs preemptive allocation only for the most critical transfer, and lets others share the remaining bandwidth fairly. Tests in §V confirm that this simplification has little impact on performance.

transport protocols and enhancements have been proposed to optimize the average (or total) flow completion time [7]–[9], [18]–[20], missed deadlines [7], [8], [20]–[22], or a mixed of these two targets [8], [23] for intra-datacenter unicast transfers. They show that finding the optimal solutions is *NP-hard* in theory; however, near-optimal results can be obtained by heuristically allocating bandwidth to transfers following the policy of *SRSF* or *EDF* [7]–[9], [20], [23].

Thus, same to the schedule of unicast transfers, to achieve efficient file disseminations for cloud applications, DCNs need a transport protocol that supports priority-based multicast.

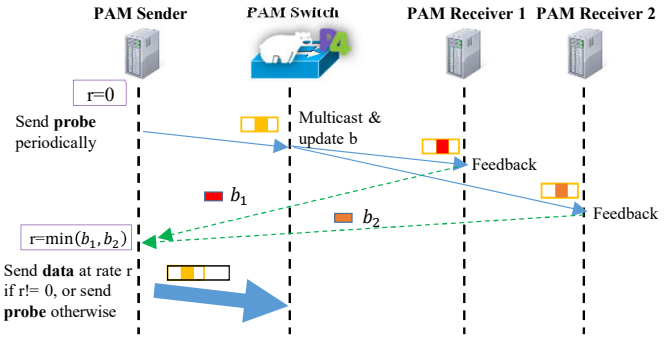


Fig. 1: PAM Overview

B. Design Challenges

To realize prioritized multicast in DCNs, many design challenges must be addressed.

Dynamic of available bandwidth. Generally, compared with flows triggered by online applications such as web search, key-value stores, and data processing, file dissemination transfers are much less time-sensitive, acting as background traffic. Due to the highly dynamic nature of intra-datacenter traffic [24]–[26], the bandwidth that file disseminations can use varies with time. Thus, the proposed scheme must quickly adapt to bandwidth dynamic without hurting other critical flows.

Decentralized scheduling. Intuitively, DCN can enable multicast prioritization with a central scheduler. However, centralized scheduling i) suffers from a single point of failure, ii) introduces non-trivial scheduling latency which results in bandwidth under-utilization, and iii) might impact other traffic as it is agnostic on burst link congestion. Hence, to make efficient use of link capacities and react to bandwidth dynamic quickly, decentralized designs are preferred.

One queue available. By letting each multicast packet carry a unique priority value according to the task's remaining transfer size and deadline [8], one could approximate SRSF and EDF scheduling distributively. However, modern switch hardware supports only $2 \sim 8$ priority queues per port, and even worse, many of them might be reserved for other purposes [9], [19], or already occupied by other unicast protocols [8], [13], [19], [27], [28]. Therefore, there is only one queue left for multicast scheduling and this queue might also be used by other traffic.

Line-rate processing. One possible way to implement decentralized multicast prioritization is to let bottleneck switches compute then allocate bandwidth to concurrent transfers explicitly in the descending order of their priorities like PDQ [7]. However, to compute each transfer's new rate, PDQ switches must loop through the list of all active flows for each packet's processing. Such operations are time-consuming and expensive, thus could not run at line rate and are hard to implement on today's switch hardware [15]. To be practical and ready-deployable, the proposed scheme must be simple enough to run at line rate and work upon today's commercial hardware.

Low feedback overheads. To converge to network dynamic quickly, the proposed protocol must provide a time-efficient mechanism to notify each transfer sender with the latest path condition (e.g., level of congestion [18], available bandwidth [7]). For unicast transfers like TCP, this is easy to implement since the feedback can be piggybacked on ACKs. However, this design is not applicable to multicast—a multicast transfer generally involves more than one receiver and employs negative-ACK (NACK) to retransmit lost packets [29], [30]. Thus, to avoid serious link overheads, the feedback mechanism used for multicast scheduling must be light-weighted.

To support priority-based multicast scheduling in DCNs, we propose PAM, a preemptive, decentralized, adaptive rate control protocol, for multicast transfers. Similar to prior rate based protocols designed for unicast transfers [7], [31], PAM explicitly controls each transfer's multicast rate based on the values calculated by switches.

C. Solution Overview

The core idea of PAM is as Figure 1 shows. Basically, each multicast data packet carries a scheduling header, made of the transfer's desired priority (p), current sending rate (r), and the possible next sending rate for negotiation (b), along the journey. Based on the desired priority as well as the link's available bandwidth, each switch computes the maximum bandwidth it can allocate to this transfer and update b in the packet header. When this packet reaches receivers, feedback is triggered to inform the sender with the new sending rate. Then, the sender updates its multicast rate accordingly. If the sending rate is zero, instead of data packets, the sender sends probe packets involving only scheduling headers at predefined intervals to get rate information from the switches.

Such a design enables PAM to perform distributed scheduling without the need of priority queues. We further use three novel designs to address the remaining challenges listed in §II-B, which make PAM distinguished from the prior art. Firstly, PAM receivers generate rate feedbacks only when they have to. This design avoids unnecessary feedbacks, making the introduced overheads low. Secondly, PAM switches allocate bandwidth to multicast transfers based on the measured load of non-multicast traffic and real-time queue occupies. With this, multicast transfers can be aware of burst congestions and is able to make efficient use of a link's remaining bandwidth immediately without hurting other traffic's performance. Thirdly, on each egress port/link, PAM switch performs preemptive bandwidth allocations only for the most critical multicast transfer; all other transfers then share the remaining bandwidth fairly. Such a design is inspired by the observation that, under

priority-based scheduling, the most critical transfer generally occupies almost the entire bandwidth of bottleneck links [9]. Based on this observation, we greatly simplify the logic that each PAM switch performs for multicast scheduling, and further make the corresponding design runnable at line rate and implementable on emerged P4-based programmable dataplane.

In the following, we first introduce the protocol details in §III then present how the switch logic can be implemented with the P4-based switch dataplane in §IV.

III. Protocol Design

As Figure 1 sketches, PAM is a rate/congestion control protocol for data center multicast. During past decades, numerous multicast protocols such as PGM [29], NORM [30], have been proposed for reliable multicast. Similar to them, PAM employs SYN packets for multicast transfer initializations, FIN packets for terminations, and receiver-triggered NACKs for lost packet retransmission. Moreover, PAM senders and receivers also maintain standard data structures for reliable transmission, such as sequence numbers, estimated round-trip times and states (e.g., timer) for in-flight packets. In the rest, we focus on the design of congestion control algorithm and describe the corresponding designs implemented at the PAM sender (§III-A), receiver (§III-B), and switch (§III-C) in detail.

A. PAM Sender

For each multicast transfer f_i, the PAM sender maintains several state variables for rate control: the transfer's current sending rate (r_i, initialized to zero), remaining size (s_i), flow deadline (d_i, which is optional), desired priority (p_i), inter-probing time multiplexer (I_i), the set of receiver IDs (D_i), the current measured RTT of each receiver ($RTT_i^x, x \in D_i$) along with their maximum value ($RTT_i = \max_{x \in D_i} RTT_i^x$), and the sending rate for next epoch reported by each receiver ($b_i^x, x \in D_i$, initialized to zero).

For active dissemination tasks, the PAM sender sends transfer f_i's data packets at rate $r_i = \min_{x \in D_i} b_i^x$. If r_i is zero, it sends a probe containing only the scheduling header without data content every I_i RTTs to get the available bandwidth of the multicast tree. To reduce the bandwidth overhead of probing, I_i is set based on the transfer's remaining size. When packet departures, the sender attaches a scheduling header containing the transfer's desired priority value p_i, current sending rate r_i, and next sending rate value b_i to the packet header. b_i is initialized as the maximal sending rate the sender can allocate to this multicast transfer, which is the NIC rate for simplification.[2]

Once the packet is sent out successfully, the PAM sender updates the transfer's remaining size and corresponding desired priority, respectively. Thanks to the power of priority-based scheduling [7], [8], by generating multicast transfers' desired priorities according to different formulates (see Table I), PAM is able to support various scheduling goals such as *minimizing the average completion time*, *minimizing the number of missed deadlines*, and *minimizing the maximum lateness*, for the schedule of file disseminations.

[2]Indeed, to handle the case of edge bottleneck, each sender's NIC could runs the same rate control logic to allocate the bandwidth on edge links.

Algorithm 1 PAM receiver operation

Inputs: $\langle p_i, r_i, b_i \rangle$: the received packet's schedule header; b^*: the maximal rate that the receiver can process and receive; b_i^{last}: this receiver's available mulitcast rate for this packet's transfer measured in last turn.
Outputs: updated b_i^{last}; send a rate feedback or not

1: $b_i \leftarrow \min(b_i, b^*)$
2: **if** there is a NACK to send **then**
3: $\quad b_i^{last} \leftarrow b_i$
4: $\quad$ attach b_i to the NACK, and send it back
5: **else if** $|b_i - b_i^{last}| \geq \gamma \cdot b_i^{last}$ **or** $rand() < \tau_i$ **then**
6: $\quad b_i^{last} \leftarrow b_i$
7: $\quad$ generate a rate feedback packet for b_i and sent it back after a random delay

On getting a rate feedback from receiver x, the PAM sender updates b_i^x, the available bandwidth of the multicast path to this receiver, with the echoed value, and updates the multicast rate r_i with $\min_{x \in D_i} b_i^x$. At the same time, the cached RTT to this receiver x (i.e., RTT_i^x), and the current maximum measured RTT among all receivers (i.e., RTT_i), would get updated based on the packet's arrival time as well.

When a multicast transfer completes,[3] or it turns out that its deadline can not be met even if data packets are sent at NIC rate, the PAM sender would terminate it and multicast a FIN to its receivers. On receiving this FIN, switches along the multicast tree release the bandwidth allocated to this transfer immediately. As we will show latter, PAM switches can tolerate the loss of FIN packets by design (see §IV-A).

B. PAM Receiver

The role of PAM receiver in congestion control is to echo the newest bandwidth that this transfer could get along the path back to its sender with feedback messages. Recall that a multicast transfer generally involves more than one receiver. The sender, as well as the links in the reversed path, would get overloaded if all the receiver send feedback for every received packet (like TCP). To avoid this, PAM receivers generate feedback messages smartly following Algorithm 1. Basically, if there happens to be a NACK to send, the PAM receiver directly piggybacks the available bandwidth value b_i on it; otherwise, the PAM receiver generates a rate feedback packet i) definitely when the change of available bandwidth is greater than the tunable threshold γ, or ii) with the probability of τ_i to tolerate the loss of feedback. Thus, the expected feedback overhead is significantly reduced by the factor of τ_i. To make the feedback overheads of large scale multicast groups controllable, in practice, the τ_i of transfer i is recommended be to set according to their group sizes: $\frac{\tau_b}{|Di|}$. As well, to prevent the sender from being flooded by the new rate messages triggered by the same bottleneck link, multicast receivers

[3]Similar to PDQ's *Early Start* [7], if the transfer will complete in the next RTT, the sender would emit a special FIN early and tag remaining data packets with the FIN_STAGE flag, such that other transfers could take its released bandwidth to achieve seamless flow switching. To highlight the performance of PAM we do not implement this optimization in ns-3 simulation.

TABLE I: The policy of how desired priorities are generated defines the scheduling goal

#	Scheduling goal	Policy of bandwidth allocation	Value of desired priority
1	Minimize average completion times	Smallest remaining size first (SRSF)	Transfer's remaining size × receiver number
2	Minimize missed deadlines	Earliest deadline first (EDF)	Transfer's remaining time to deadline
3	Minimize maximum lateness	First come first reserve (FIFO)	Radix complement of the transfer's started time
4	Fair sharing	Fair sharing on bottleneck links (FS)	The value of the lowest priority
5	Predefined priories	In customized priority order	Predefined priority value

would add random delays within $[0, RTT_i]$ before sending the feedback. In our simulation, γ and τ_b are set to 0.05 and 0.01 by default.

Besides the notification of the corresponding path's available bandwidth, new rate messages in PAM could also work for *flow controls*. In case the rate of multicasting is too fast for the receiver to receive and process reliably, it could generate a conservative new rate to slow the sender down.

C. PAM Switch

PAM switches achieve priority-based bandwidth allocation with explicit rate controls. However, today's switch dataplane only provides a limited set of primitive operations without the support of complex computations such as *sorting* and *list traversal*. To be implementable on current programmable switches, we employ a simplified yet efficient design: for each link, PAM switch heuristically i) lets the most critical multicast transfer (i.e., the one with the lowest desired priority value) occupy the available bandwidth as much as possible, and then ii) allocates the remaining bandwidth to all other active multicast transfers fairly. Since the most critical transfer generally occupies almost the entire bandwidth of bottleneck links, for the remaining link capacity, there is little difference between prioritized allocation and fair sharing. Indeed, for the schedule of average transfer completion time minimization and missed deadline minimization, our extensive simulations imply that their performance gaps are negligible (§V).

To achieve the above design, PAM switch maintains four types of states for each egress port/link: i) the most critical transfer through it, ii) the available capacity for multicast transfer, iii) the total traffic load, and iv) the number of currently active multicast transfers. Based on these states, it allocates bandwidth to multicast transfers according to their priorities following Algorithm 2.

Let F and N be the set and the number of currently active multicast transfers on this link, k, p_k, and r_k be the transfer ID, priority value, and current sending rate of the most critical transfer through this link, respectively. Then, on getting a FIN packet (i.e., T_i is FIN), which means this transfer has completed, the PAM switch would directly remove it from the active transfer set F and update N (Line 3). Moreover, provided that the transfer to be removed happens to be the most critical one, PAM switch would eliminate its information cached on the switch at the same time (Line 5). On getting a normal multicast packet (i.e., T_i is MULTICAST_DATA), PAM switch first adds it to the active transfer set and updates the corresponding N if it is not in it (Line 8). If this packet's transfer is the most critical one (Line 10), b_i, the transfer's new sending rate after a RTT, is updated according to the prior bottleneck and the available bandwidth on this link (Line 11). Also, the most critical (active) transfer's states cached in the switch would get updated correspondingly (Line 13). Otherwise, the transfer would share the remaining bandwidth (i.e., $C(t) - r_k$) with all other $N-1$ transfers fairly (Line 15).

Recall that, if the sending rate of a multicast transfer is zero, the corresponding PAM sender would periodically send probe packets to i) release the allocated bandwidth on each link, and ii) estimate the available bandwidth that it can use (after a RTT). To release the allocated bandwidth of paused transfer, PAM switch first processes the probe packets as FINs (Line 2-5). As for the estimation of available bandwidth, the process of the probe is similar to that of normal multicast packets. However, since the probe packet's transfer is inactive and has

Algorithm 2 PAM switch operation

Inputs: i, $\langle p_i, r_i, b_i \rangle$, and T_i: the received packet's transfer ID, schedule header, and packet type;
k, p_k, and r_k: the transfer ID, priority value, and sending rate of the most critical transfer through this link;
F and N: the set and the number of currently active multicast transfers through this link/port;
$C(t)$: the current available bandwidth for multicast;
Output: updated scheduling header values $\langle p_i, r_i, b_i \rangle$

1: **if** T_i ∈{FIN, PROBE} **then**
2: **if** $i \in F$ **then**
3: $F \leftarrow F \setminus \{i\}$; $N \leftarrow N - 1$
4: **if** $k = i$ **then**
5: $k \leftarrow 0$; $r_k \leftarrow 0$; $p_k \leftarrow \inf$
6: **else if** T_i is MULTICAST_DATA **then**
7: **if** $i \notin F$ **then**
8: $F \leftarrow F \cup \{i\}$; $N \leftarrow N + 1$
9: **if** T_i ∈{PROBE, MULTICAST_DATA} **then**
10: **if** $p_i < p_k$ || $i = k$ **then**
11: $b_i \leftarrow \min(b_i, C(t))$
12: **if** T_i is MULTICAST_DATA **then**
13: $k \leftarrow i$; $r_k \leftarrow r_i$; $p_k \leftarrow p_i$
14: **else if** T_i is MULTICAST_DATA **then**
15: $b_i \leftarrow \min(b_i, \max(0, C(t) - r_k)/(N-1))$
16: **else if** T_i is PROBE **then**
17: $b_i \leftarrow \min(b_i, \max(0, C(t) - r_k)/N)$
18: $b_i \leftarrow \min(b_i, C)$ ▷ C is the link's capacity.
19: **return** $\langle p_i, r_i, b_i \rangle$

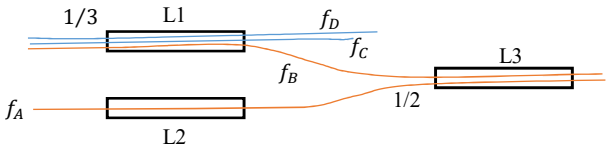
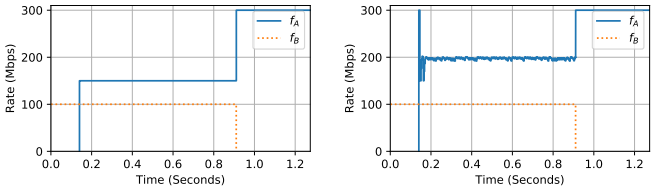


Fig. 2: Inconsistent bottleneck results in under-utilization



(a) $a = 1$ (b) α is updated via (3)

Fig. 3: PAM solves the problem of inconsistent bottleneck by updating α via (3), in which $\eta_o = 0.02$.

been removed from the active flow set F, probe packets would not cause cache updates (i.e., Line 13), even if it would be the most critical transfer. By the same reason, if it belongs to the case of fair sharing, the number of transfers sharing the remaining $C(t) - r_k$ with it is N rather than $N-1$ (Line 17).

Link capacity estimation. In a network, the sending rate of a transfer is determined by the available bandwidth of its bottleneck link, while different transfers generally involve various bottlenecks. On each link, the bandwidth left by other traffic and the most critical transfer is split "equivalently" to the remaining active transfers. When transfers involve different bottlenecks, such a design might cause a waste of link utilization. As an example, consider Figure 2, in which there are three links named L1, L2, and L3. After allocating to other traffic (e.g., TCP) and their most critical transfers (which are omitted for simplification), each link has 1 unit of bandwidth left. Then $\{f_A, f_B, f_C, f_D\}$ would use these left capacities to complete their tasks. Obviously, the bottleneck link of f_A and f_B are L3 and L1, respectively; f_A and f_B would get the sending rate of $\min(1, \frac{1}{2}) = \frac{1}{2}$ and $\min(\frac{1}{3}, \frac{1}{2}) = \frac{1}{3}$ according to Algorithm 2. Thus, the load on bottleneck link L3 is $\frac{1}{2} + \frac{1}{3} = \frac{5}{6}$, resulting in under-utilization. Such a problem is confirmed in ns-3 simulations as Figure 3a shows, in which each link has the remaining bandwidth of 300Mbps but f_A and f_B are failed to make efficient use of them all, resulting in a waste of 50Mbps. On the other hand, a burst of new transfers could cause the switch to temporarily allocate more bandwidth than its capacity, resulting in queuing buildup. Moreover, as background traffic, the bandwidth that multicast transfers can use would vary with the load of non-multicast traffic.

To account for all these cases, PAM switch borrows from [21] and adjusts a link's capacity for multicast based on the observed link utilization and queuing occupancy in every 2 RTTs as following:

$$C(t) \leftarrow \alpha \left(C - B(t)\right) - \beta \max(0, q(t) - q_0) \tag{2}$$

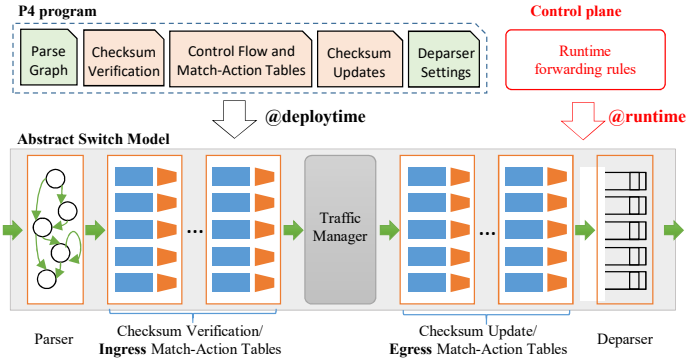


Fig. 4: V1Model architecture of P4 programmable switch

$$\alpha \leftarrow \begin{cases} \alpha + \eta & \text{if } \eta > \eta_o \\ 1 & \text{if } \eta < 0 \\ max(1, \alpha - \eta) & \text{otherwise} \end{cases} \tag{3}$$

$$\eta = 1 - \frac{B_m(t)}{C - B(t)} \tag{4}$$

Here, C is the link's capacity; $B(t)$ and $B_m(t)$ are the observed loads of non-multicast and multicast traffic respectively; $q(t)$ is the instantaneous queue size; q_0 is the desired maximum queue size; and η_o, β are chosen for stability and performance.[4] As the ns-3 simulation results in Figure 3b show, such a design enables PAM switches to make work-conversing bandwidth allocations gracefully.

FIN loss. In rare cases, a multicast transfer's FIN might get lost; or the routing of a multicast tree might get changed because of fail-over. To guarantee that the link capacities allocated to such terminated or rerouted transfers could get released correctly, PAM switch also maintains an idle timeout for each multicast transfer. Once a transfer has received no packets in the given number of seconds, the switch removes it from the active transfer set and updates the corresponding states (see §IV-A for data-plane implementation details).

IV. Prototype in P4

As the previous section has shown, with the cooperation of PAM sender, switch, and receivers, the network can perform priority-based explicit rate controls for multicast. In practice, the designs of PAM sender and receiver are easy to implement in software. However, PAM switch requires non-trivial modifications to the forwarding hardware. In this section, we show how to implement PAM switch's logic on emerged P4-based programmable switches with approximation designs.

Basically, as Figure 4 illustrates, the P4 language [16] along with recent programmable chips [17], enables network engineers to write data-plane programs to specify how packets are parsed, matched, processed, and forwarded on switches, enabling customized protocols. To support PAM, we define a new type of header upon UDP for rate control and implement

[4]Indeed, q_0 and β control the target queue occupy and the speed PAM react to queue buildups, respectively. Motivated by [7], [8], [21], q_0 can be set to one BDP, η_o can be set to 0.02, and β can be set to $\frac{2}{RTT}$.

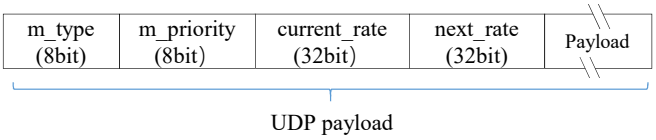


Fig. 5: PAM header example

the function of Algorithm 2 at the egress switch pipeline. Although most of them (e.g., selecting the most critical multicast transfer, updating each transfer's next sending rate based on the available bandwidth on this link) are simple and straightforward to implement in P4, there are still some types of data structures (such as floating value, collection) and arithmetic operators (e.g., multiplication, division) that can not be directly implemented in dataplane because of the line-rate processing requirements of switch dataplane [14], [15]. In the following, we first describe the novel designs and approximate techniques that we employ for Algorithm 2 (§IV-A), then analyze its overheads (§IV-B), and demonstrate its effectiveness via case studies (§IV-C).

A. Key Designs

Identification of multicast traffic. Figure 5 shows the customized header that PAM switch employs for bandwidth allocation.[5] In our prototype implementation, PAM header occupies 10 bytes, which, along with the multicast data, is encapsulated in UDP payload. To distinguish PAM traffic with the other, once a UDP's payload size is equal or larger than 10 bytes, the switch's parser tries to explain it as a PAM packet and this PAM header would take effect in egress processing only when it does match with some multicast rules at the ingress. Basically, each PAM transfer can be identified by the 4-tuple of its source IP address, source UDP port, target IP address, and target UDP port. With such a design, PAM switch can recognize concurrent transfers and control their sending rates separately.

Count of active transfer. As Algorithm 2 shows, for many multicast transfers, the bandwidth they can obtain on a link partly depends on the number of active multicast transfers through the same link. To precisely count the number of active transfers and react to transfer *join* (i.e., MULTICAST_DATA), *pause* (i.e., PROBE), and *leave* (e.g., FIN), PAM switch should maintain which transfers are active. In practice, P4 does not support the data structure of collection; thus, we employ bloom filters to approximate [32]. Basically, we use a bit array (i.e., P4 register) to store transfer's appearance and hash each multicast transfer to k array locations. On getting a multicast packet, if all the bits stored in its k locations are ones, it means this transfer has been counted already; otherwise not. By using bloom filter, PAM switch increases, keeps, or decreases the number of currently active transfers based on the received packet's type. Basically, each egress port should have a distinct

[5]According to the workflow of PAM protocol, there are five kinds of PAM packets: normal data packet (i.e., MULTICAST_DATA), the last few data packet (i.e., FIN_STAGE), PROBE, ACK, and FIN.

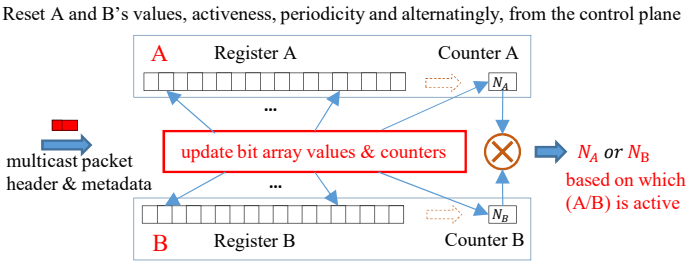


Fig. 6: PAM counts active transfers with a ping-pong design.

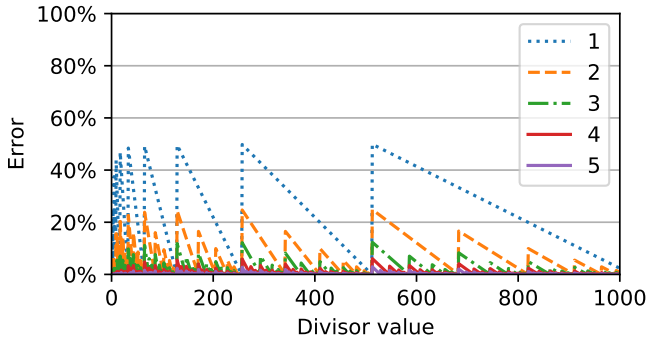


Fig. 7: The maximum truncation error of match-action approximated division rapidly decreases when more right shifts are used; 4 shifts are able to make the error under 6.1%.

counter. To make compact use of data-plane memories, we let egress ports residing in the same pipeline [33] share bloom filters by taking each packet's egress port index as inputs for index hashing as well.

Toleration of lost FIN. It should be noted that, in some cases, FIN packets might get lost (or absent in case of rerouting) or arrive at a switch prior to the last several data packets because of packet reordering. Then, the measured count of active transfer might be larger than the correct value, or even worse, the bandwidth allocated to the most critical transfer can not get reclaimed because the switch would not see its FIN, resulting in low bandwidth utilization. To address this, the PAM switch i) records the most critical transfer's last reference time in a register to detect then remove the expired critical transfer, and ii) measures the number of active multicast transfers with two suites of bit arrays and counters named A and B as Figure 6 illustrates. At any time, either A or B is active. Then, a control plane program periodically swaps the activeness of A and B every $T_{expired}$ seconds, and resets the counter and all bit array values to zeros, once their corresponding suite becomes inactive. On getting a PAM packet, A and B update their own bit arrays and counters independently; and the switch uses the active one's counter value for bandwidth allocation. Following this, PAM switch is robust to remove expired transfer's states.

Division in data plane. On each link, multicast transfers except the most critical one share the remaining bandwidth $C(t) - r_k$ equally. However, division is not supported by P4 data plane. It is oblivious that, the division of two positive integers can be transformed into the sum of several right

shifts. As an example, given a positive integer M, we have $\frac{M}{8} = (M >> 3)$, $\frac{M}{9} = (M >> 4) + (M >> 6) + (M >> 10) + \cdots$, where $>>$ is the operator of right shift supported by P4-compatible hardware. Motivated by this observation, PAM switch approximates division operations with *match-action* table lookups, in which the divisor acts as the matching key and the action is to sum the dividend's right shifts. To guarantee line-rate packet processing, the number of bit shifts the data-plane action can perform should be limited, which would impact the quotient's precision in turn. Figure 7 shows the maximal truncation error decreases greatly with the number of used right shifts. For example, the maximum errors of approximate division with 3, 4, and 5 right shifts are less than 12.4%, 6.1%, and 3%, respectively. Thus, the approximation of 4 right shifts is fair enough for many cases. As for the design of matching types, current hardware switches are easy to support a large number of exact matching rules. In consideration of there might be only tens to hundreds of active multicast transfers (unlike mice flows), we use an exact match-action table to approximate division precisely in our prototype implementation.[6]

Measurement of link load. Recall that (see Equation (2)) a link's capacity for multicast is based on the observed link utilization, which is determined by both the load of non-multicast traffic (i.e., $B(t)$) and multicast traffic (i.e., $B_m(t)$) in turn. To achieve these measurements, PAM switch directly employs the Discounting Rate Estimator (DRE) proposed by Conga [34]. Basically, DRE maintains a register, say X, for each type of traffic and increments X by the packet size it sent over the link/port. At the same time, a control plane program located at the switch decrements X with γX periodically (every T_{dre}): $X \leftarrow X - \gamma X$, where γ is a multiplicative factor between 0 and 1. Then, based on the measured $B(t)$, $B_m(t)$, and queue instance length $q(t)$, the local control program updates the link's $C(t)$ according to Equation (2).

B. Analysis of Hardware Overheads

PAM leverages a match-action table with exact matchings to achieve division computations for multicast packets, and a couple of registers to cache the information of the current most critical multicast transfer, and the show-up and the number of active multicast transfers for each egress.

Generally, the size of the division table should not be less than the maximum number of active multicast transfers going through the same egress. According to PAM's preemptive designs, such a value would be quite small as the most critical transfer would occupy almost whole the available bandwidth, making other multicast transfers paused (i.e., inactive). Moreover, PAM is designed to achieve efficient file dissemination among servers; it is easy to limit the maximum number of concurrent multicast transfers among switches from the application controller. In addition, recent advantages of reconfigurable switches have shown that modern hardware can support a quite large number of exact match-action entries in a memory-efficient way [35], [36]. Hence the hardware overhead introduced by PAM's division table is trivial and controllable.

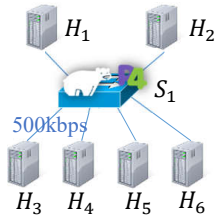
Regarding the occupancy of register, for each egress port, PAM switch needs i) four register cells to record the ID, priority, last packet time, and current sending rate of the selected most critical multicast transfer, ii) two register cells to measure the load of non-multicast and multicast traffic, and iii) two suites of registers, each of which contains a transfer counter and a bloom filter, to count the number of active multicast transfer in a ping-pong manner. A straightforward design is to let each egress use distinct registers for bloom filter. Suppose that the switch consists c egress ports, the average number of active multicast transfer is n, and the bloom filter involves k hash functions and a bit array with m cells. Then, there are $2c$ bloom filters, taking $2mc$ bits of registers in total, and the probability of a false positive is $(1-e^{-\frac{kn}{m}})^k$ [32]. To make more efficient use of register memories for bloom filter, we let all egress port belonging to the same pipeline share the same bit array by taking the egress port's index into account when computing locations. Say that the average fanout of multicast is c^*, where $c^* \leq c$. Then, with the same size of registers, the probability of a false positive is improved to $(1 - e^{-\frac{kn}{m}\frac{c^*}{c}})^k$ now, which is always less than $(1 - e^{-\frac{kn}{m}})^k$. Recall that, the number of concurrent multicast transfers are limited in practice, say 1000 for instance; then, just with 2MB of on-chip memories and 3 distinct hash functions, PAM could make the false positive probability of multicast transfers who have the average fanout of 20 outputs on the PAM switch, less than 4.2×10^{-7}.

C. Case Study

To evaluate the effectiveness of PAM, we feed the bmv2 simple switch [16], [37] with our P4 program and conduct a toy example consisting of three file dissemination tasks on Mininet as Figure 8a shows. Different from production-grade software switch like Open vSwitch, the bmv2 simple switch is designed as a software tool for P4 development and has very poor performance in terms of throughput and latency [37]. Moreover, in the Mininet environment, all virtual hosts and the bmv2 switch use the shared CPU for simulation. Thus, to avoid resource competition between them and to highlight the results, we scale each task's file sizes down to 120KB, 200KB, 50KB, respectively, and set each link's bandwidth to 500 Kbps. Limited by the performance of bmv2 simple switch, the One-Way Delay (OWD) between hosts is not very consistent, which is about several microseconds. We let paused multicast transfers send probes every 0.1 seconds. These three multicast tasks, f_A, f_B, and f_C, start at time 1s, 0s, and 2s, and would expire at 4s, 6s, and 5s, respectively. To highlight the comparison, auto-termination of expired transfer is disabled. PAM senders generate desired priorities according to the policies listed in Table I. On receiving multicast packets, the PAM switch allocates bandwidth following Algorithm 2.

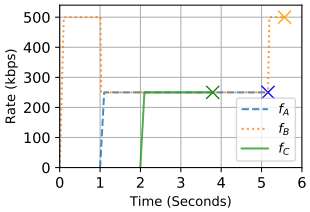
Figure 8b to 8e illustrate the corresponding results of how their sending rates change. It is obvious that, PAM is flexible to achieve various scheduling goals. During the scheduling, once the number of active requests changes, PAM transfers can converge to the new allocations immediately (in one RTT).

[6]In practice, exact match uses much fewer memories than LPM and ternary. If the number of active multicasts is quite huge, one can replace this exact matching-based division with ternary-based matchings on a log table [14].

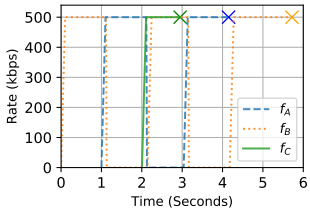


FID	Src	Dst	Size	t_{start}	$t_{expired}$
f_A	H_1	H_3, H_4	120KB	1 s	4 s
f_B	H_1	H_3, H_5	200KB	0 s	6 s
f_C	H_2	H_4, H_6	50KB	2 s	5 s

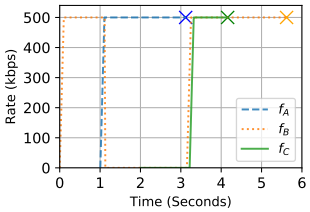
(a) The toy example



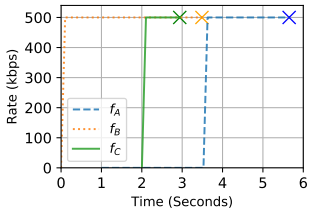
(b) Fair sharing



(c) Smallest remaining size first



(d) Earliest deadline first



(e) First come first reserve

Fig. 8: PAM is flexible to schedule multicast transfers respecting various policies; note that, auto-termination of expired flow is disabled here.

For example, Figure 8b shows that the completion times of these three transfers under the setting of fair sharing are 1.79s, 4.16s, and 5.57s, respectively, resulting in the average of 3.84s. As *fair sharing* is the fundamental bandwidth allocation principle of TCP, thus, such a result yields a lower bound of all the widely existing single-rate multicast schemes that adopt slowly TCP-alike congestion controls (e.g., PGMCC [10], ERMCC [11], MCTCP [12]). As a comparison, when generating desired priorities with the SRSF policy, multicast transfers occupy the available bandwidth on each link respecting to their remaining sizes, resulting in an optimized average completion time. At the very beginning, f_B takes all the available bandwidth for multicast; then, f_A, who has the smaller remaining size, seizes the bandwidth of link $S_1 \rightarrow H_3$, thus pausing f_B, at time 1s. Again, f_C starts at time 2s and takes over the bandwidth of link $S_1 \rightarrow H_4$; f_A hangs and the released bandwidth enables f_B to continue its multicast, immediately. Once f_C completes, f_A becomes the one with the smaller remaining size on all links; it occupies the bandwidth on link $S_1 \rightarrow H_3$ and $S_1 \rightarrow H_4$ and resumes its multicast. Finally, all these transfers get finished at 0.95s, 3.15s, 5.72s, respectively; their average value is 3.27s, yielding the improvement of 15% over the default—fair sharing.

As another example, when deadlines are enabled, these transfers would expire at $4s$, $6s$, and $5s$. Only the earliest deadline first scheduling (EDF) ensures that they all meet their deadlines: f_A misses its deadline under all other schedulings. Besides fair sharing, SRSF, and EDF, we also change the policy to FIFO for minimizing the transfers' maximum lateness and rerun the tests. Results in Figure 8 confirm that the corresponding scheduling goal is achieved: transfers' maximum lateness is reduced to 4.65s, better than these (5.57s, 5.72s, 5.62s)[7] achieved by all other scheduling schemes.

V. Performance Evaluation

In this section, we use ns-3 simulations to evaluate the performance of PAM and find that:

- PAM is near-optimal; on minimizing the average transfer completions and missed deadlines, the performance gaps between PAM and the optimal schedule that make preemptive priority-based bandwidth allocation for all transfers, are less than 3.7% and 2.1%, outperforming the default fair sharing up to 5.7× and 75%, respectively.
- PAM has low overhead and is TCP-friendly. The optimization design PAM employs could eliminate most of the rate feedbacks. And with proper configurations, PAM makes efficient use of the remaining bandwidth left by TCP while letting coexist TCP flows feel as PAM transfers did not exist.

Simulation setup. To evaluate PAM's performance, we consider two typical multicast file dissemination scenarios namely *Data Replication* and *Service Deployment* in simulation. These two types of multicast tasks abstract out the common communication pattern of the data replication process of today's distributed file systems [6], and the file dissemination process of service deployment (and upgrade), respectively.

Basically, in *Data Replication*, each server in the cluster are continuing to receive some data chunks/files; to improve reliability, these files are replicated to R randomly selected servers with multicast. Here, we denote the receiver number of a multicast (i.e., R) as its *fanout* and set the typical value to 3. As for *Service Deployment*, a fixed subset of hosts in the cluster would work as the software repository; they need to efficiently disseminate the same software files (e.g., docker images) to a random group of nodes to deploy or upgrade distributed services. Similar to *Data Replication*, the number of receiver nodes are defined as the multicast fanout; we assume it follows an exponential distribution with the mean of 3 by default. Regarding the file/data sizes, they are synthesized based on the transfer size distribution measured from a (web search) data center [8]. Since, PAM is not designed for the dissemination of tiny file, for a network whose edge link has the capacity of $c \times$ 1Gbps, we scale the file sizes to $c \times [1, 100]$MB, accordingly. In tests, we vary c from 0.5 to 16 to study the robustness of PAM. For a network involving N hosts, we synthesize $7N$ multicast tasks for tests. In line with prior study [8], [13], these tasks are assumed to arrive in Poisson and the arrival rate is varied to obtain a desired level of network load. We use 0.9 as the default level of network load. For each parameter setting, we run 10 instances.

To highlight the schedule of multicast rates and eliminate the effects of routing, we abstract the cluster network as a big

[7]Ideally, these three values should be equal; however, they are not because the performance of bmv2 is not very stable.

switch following prior study [8], [38]. Indeed, this is a reasonable abstraction of today's production data center network architectures since non-blocking Clos typologies are widely adopted and congestion generally occurs at the edge [8], [13]. By default, the cluster involves 48 servers and the capacity and latency of each edge link are 2Gbps and 10us, respectively.

In the following, we first investigate the performance of PAM under various network loads, multicast fanouts, cluster scales, and link capacities in §V-A, then examine its feedback overheads, impacts on coexisting TCP traffic, and convergence speed in §V-B. As for the performance metrics, similar to prior work [7], [8], we mainly consider i) the Average of their normalized Transfer Completion Times (ATCT)[8] for the minimization of average completion time, and ii) the proportion of transfers that miss their deadlines for the minimization of missed deadlines. Let X_A and X_B be the average of normalized transfer completion times under the schedule of scheme A and B, respectively; then in this paper, we define the performance gap between scheme A and B by $\frac{|X_A - X_B|}{\max(X_A, X_B)}$ and the performance gain of A over B by $\frac{X_B}{X_A}$. As for the schedule of deadline-constrained transfers, let Y_A and Y_B be the ratio of missed deadlines under the schedule of scheme A and B; we define the performance gap between A and B by $|Y_A - Y_B|$ and the performance gain of A over B by $Y_B - Y_A$.

A. PAM Performance

Essentially, the core of PAM is to approximate prioritized bandwidth allocation for multicast transfers by making preemptive allocations only for the most critical transfer. This design simplifies PAM switch and makes it ready-deployable. To check how far this approximation draws us away from the optimal, we modify PAM switch to cache the detailed states of all active transfers and perform precisely priority-based bandwidth allocations for all active transfers like PDQ [7].

1) Deadline-unconstrained tasks: Figure 9 shows a case study of the completion times of multicast transfers under PAM, the ideal/optimal, and the default fair sharing scheduling schemes. As their absolute average completion times in Figure 9a and 9c show, in this case, the average TCTs achieved by PAM for data replication and service replications, i.e., 0.2603s and 0.2816s, are quite close to or even slightly better than those of the optimal, i.e., 0.2614s and 0.2785s, yielding significant improvements of 0.1229s and $0.1558s$ over those of the fair sharing. Such performance improvements are confirmed by the distributions of their normalized completion times shown in Figure 9b and 9d. Basically, the distribution of normalized completion times scheduled by PAM is almost overlapped with that of the optimal. We find that they would achieve exactly the same average completion completions in most of the tests. And on terms of the average value their normalized completion times, PAM achieves more than $2.6\times$ and $3.1\times$ improvements over those of the fairness scheduling.

To check whether PAM achieves consistent performances over various network loads, multicast fanouts, cluster scales, and link capacities, we override the default parameter settings and rerun the tests. Results in Figure 10 and 11 confirm the effectiveness of PAM.

Impact of network load. Figure 10a and 11a show the average completion times of the two types of transfers as we vary the network load from 0.5 to 0.99. The results show that with the increase of network load, the average transfer completion times scheduled by all these schemes would increase slightly, and the increments made by fairness scheduling are worse than these of PAM and the optimal. There is no surprise: a heavier loaded network would deteriorate the transfer completion times, and in that case, priority-based bandwidth allocations are more crucial for the optimization of average completion times. Indeed, such findings are consistent with those of the optimized schedule of unicast transfers [8].

Impact of multicast fanout. Quite similar to the change of network load, as results in Figure 10b and 11b imply, for both data replication and service deployment, the average transfer completion times would increase if transfers involve more receivers. This is reasonable: with larger fanouts, more multicast transfers would interleave with each other, in which condition, more bandwidth competitions occur and transfers need more times to complete then. Furthermore, the interaction of multicast transfers also increases the room of rate scheduling. As a result, prioritized-based scheduling would obtain increased performers gains in such a case as is shown.

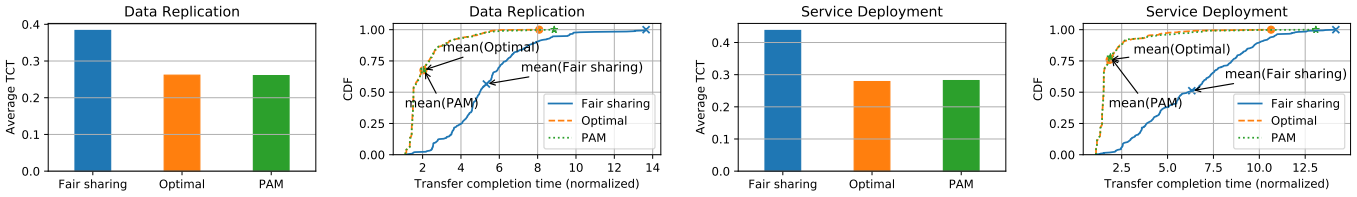
Impact of network scale. Next, we increase the network scale from 32 nodes to 96 and rerun the tests. Basically, as Figure 10c and 11c exhibit, given a consistent multicast fanout and network load, Scheduling schemes would achieve consistent performance under various network scales. However, fair sharing involves an exception on scheduling service deployment transfers as Figure 11c shows. This might be caused by the traffic pattern of server deployment. In our simulation, all service deployment transfers are assumed to be served by three fixed registries. With the network scaling up, each of them has to server more transfers. However, fair sharing could not make very efficient use of the network bandwidth, Thus, transfer completion times would be enlarged since there would be more transfers for each sender.

Impact of link capacity. In our default settings, the capacity of each link is set to 2Gbps. To check whether PAM is able to work well in various network environment, we vary each link's capacity from 0.5Gbps to 16Gbps. Results in Figure 10d and 11d imply a consistent performance of PAM. This gives us the insight that, PAM is likely to work well on future 100Gbps and 400 Gbps networks as well.

All these tests confirm the consistent and stable performance of PAM on optimizing the average transfer completion times for multicasts. Among all these tests, the performance gaps between PAM and the optimal are always less than 3.7%, outperforming the fair-sharing scheme up to $5.7\times$.

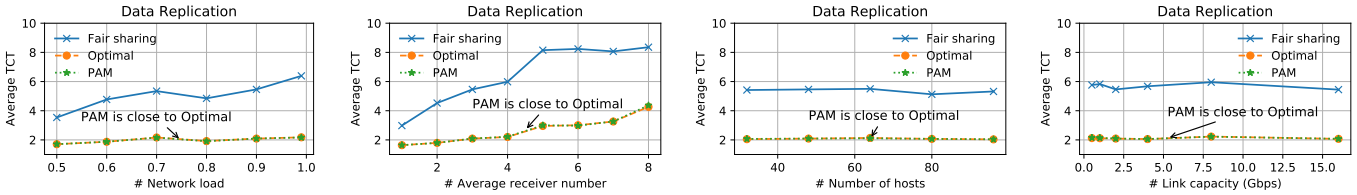
2) Deadline-constrained tasks: To check the performance of PAM on scheduling deadline-constrained multicast transfers, we configure both PAM and its optimal variant to schedule transfers with the policy of EDF. Figure 13 shows the proportion of transfers that do not complete within their

[8]I.e., TCT, the completion time of each transfer, is normalized to the ideal value this transfer could achieve.



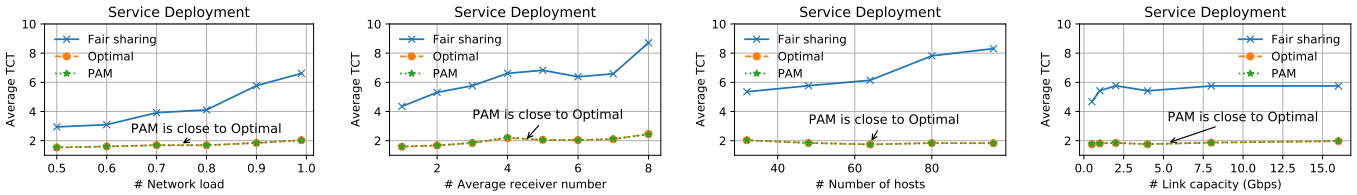
(a) Average TCTs (b) TCT distributions (c) Average TCTs (d) TCT distributions

Fig. 9: [ATCT minimization] PAM achieves near-optimal results on minimizing transfer completion times; it outperforms fair sharing up to 3.1× in terms of the average of their normalized completion times.



(a) Impact of network load (b) Impact of multicast fanout (c) Impact of network scale (d) Impact of link capacity

Fig. 10: [ATCT minimization, Data Replication] PAM achieves near-optimal performance under various settings on minimizing the average transfer completion time for data replication transfers: gap < 3.62%.



(a) Impact of network load (b) Impact of multicast fanout (c) Impact of network scale (d) Impact of link capacity

Fig. 11: [ATCT minimization, Service Deployment] PAM achieves near-optimal performance under various settings on minimizing the average transfer completion time for service deployment transfers: gap < 3%.

deadline under various settings. Basically, the deadline of transfer f_i is assumed to be $(2 + z) \times c_i$, where c_i is its minimum completion time in an empty network. Similar to prior work [7], [8], [22], z follows an exponential distribution whose mean value is 1.

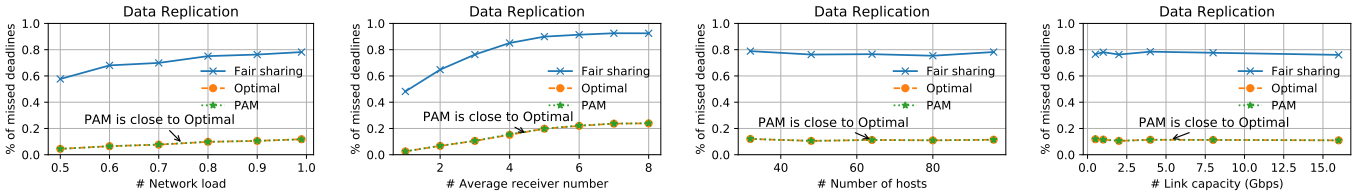
Consistent with the cases of average completion time minimization: these schemes achieve consistent ratios of missed deadlines for multicast transfers across various network scales (Figure 12c and 13c) and link capacities (Figure 12d and 13d); both their ratios of unfinished transfers and the performance gaps between PAM and fair sharing grow as we increase the network loads (Figure 12a and 13a); and again, larger multicast fanouts would let more transfers miss their deadlines (Figure 12b and 13b). Among all these tests, PAM achieves near-optimal results; its gap to the optimal is always less than 2.1%, outperforming the fair sharing scheme up to 75%.

B. PAM Property

Above tests show that PAM is near-optimal in items of the average completion times and the number of missed deadlines. In this part, we further investigate its scheduling overheads, impacts on TCP traffic, and convergence speed.

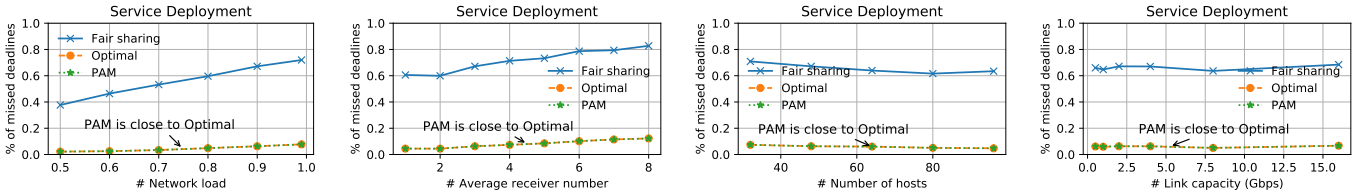
Scheduling overheads. PAM scheduling involves 3 types of traffic overheads: i) a schedule header that each PAM packet must carry with; ii) some probe packets that triggered by paused transfers; and iii) the packets for rate feedbacks. By design, the schedule header needed by PAM is quite small: according to our prototype implementation (Figure 5), 10 bytes are fairly enough. Also, PAM senders reduce the impact of probe by letting paused transfers with large remaining size sent probes less frequently. As for the cost of rate feedbacks, we measure their numbers; results indicate that the optimization designs of PAM reduce their loads greatly. Take the test instance shown in Figure 9a as an example, the optimization design of PAM (Algorithm 1) eliminates about 50% rate feedbacks for probe packets and 96.8% for normal data packets, resulting in a total reduction of 93%.

Impact on TCP. To check the impact of PAM over TCP traffic, we randomly replace half of data replication multicast transfers in the test instance shown in Figure 9a with TCP flows, then check whether those TCPs' completion times would be impacted by PAM traffic. Recall that the original multicast transfer involves more than one receivers; here, each TCP only sends data to the one with the lowest node label. As



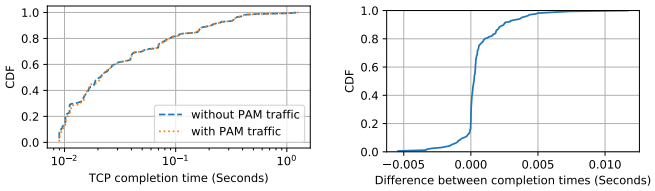
(a) Impact of network load (b) Impact of multicast fanout (c) Impact of network scale (d) Impact of link capacity

Fig. 12: [Missed-deadline minimization, Data Replication] PAM achieves near-optimal performance under various settings on minimizing missed deadlines for data replication transfers: gap < 2.1%.



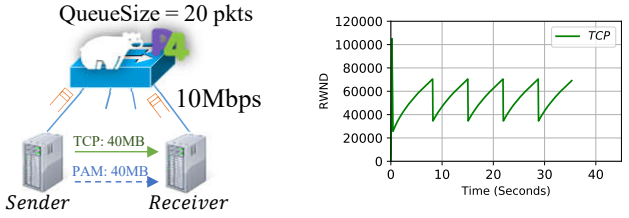
(a) Impact of network load (b) Impact of multicast fanout (c) Impact of network scale (d) Impact of link capacity

Fig. 13: [Missed-deadline minimization, Service Deployment] PAM achieves near-optimal performance under various settings on minimizing missed deadlines for service deployment transfers: gap < 1%.

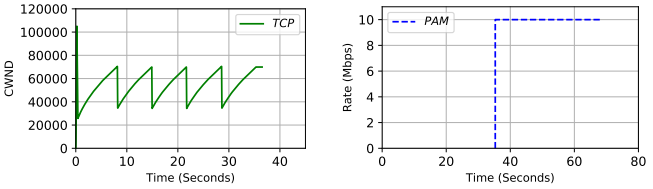


(a) CDF of TCTs (b) CDF of TCT increments

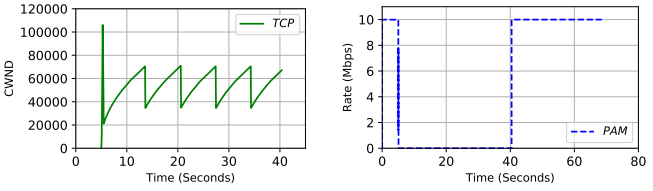
Fig. 14: The impacts of PAM traffic on the completions of TCP flows are trivial: the average TCP completion times with- and without- PAM traffic are 0.0772s and 0.0778s, receptively, yielding a gap within 0.8%.



(a) The toy case: a TCP flow share links with a PAM flow. (b) The curve of TCP's CWND when no other flow exists



(c) The curves when TCP starts before the PAM transfer



(d) The curves when PAM starts before TCP

Fig. 15: PAM has negligible impact on coexisting TCP flow.

Figure 14 shows, even though some TCPs' completion times do get affected, the impacts are negligible: the average TCP completion times with- and without- PAM traffic are 0.0772s and 0.0778s, receptively, yielding a gap within 0.8%. To look into the detail of how TCP would get affected, as Figure 15a shows, we let a PAM transfer and a TCP flow go through the same link whose bandwidth and latency are 10Mbps and 1ms, respectively. The data sizes of both transfers are 40MB. We measure how the TCP's congestion window (CWND) and the PAM transfer's sending rate change with time. As Figure 15 shows, with a large β (i.e., $\frac{2}{RTT}$), the PAM transfer has negligible impact on the TCP flow as if no PAM traffic coexists, no matter in what order they appear. By design, PAM switch estimates the link capacity that PAM transfers can use, based on the observed link utilization and queuing occupancy (see Equation (2)). When the TCP starts first (Figure 15c), the PAM switch knows there is no bandwidth left for multicast; thus, it would keep this PAM transfer paused until the TCP completes. In case the PAM transfer starts first (Figure 15d), the new incoming TCP would cause queue build-ups on the PAM switch. Then, the PAM switch gets known immediately and reduces the PAM transfer's sending rate accordingly. In a very short time, the multicast transfer's sending rate decreases to zero and the TCP flow occupies the bandwidth of the entire link/path. We try various TCP congestion control algorithms like New Reno, Vegas, and Bic in simulations, and obverse

the consistent results.

Convergence speed. Among all simulations, we also observe that, similar to other priority-based explicit rate control protocols [7], [31], PAM converges to equilibrium quickly within one or two RTTs for stable workloads and makes nearly perfect utilization of bottleneck links. Thus, PAM would be a perfect multicast protocol for file dissemination tasks widely existing in today's high-speed data center networks.

VI. Related Work

Since its first appearance in the 1980s, network layer multicast has been studied for decades and a large number of enhancement proposals and applications have been proposed [4], [10], [39]. In this section, we briefly review its recent advancements in the context of data center networks.

Multicast routing technique. Because of the high costs on forwarding entries and control messages, legacy distributed multicast routing protocols designed for wide-area networks (e.g., IGMP and PIM) fall short of supporting the dynamic, large-scale, and huge-number multicast groups required by modern data center networks [2], [3], [40]. To overcome this, many SDN-based solutions have been proposed by leveraging the unique topological properties of modern data center network architectures [3], [4]. For example, D. Li et al investigated the use of bloom filters to compress the multicast forwarding states [1], [40]. X. Li and M. Freedman scaled data center multicast to support thousands of multicast groups through controller-assisted address partitions and local multicast forwarding rule aggregations [2]. And more recently, M. Shahbaz at el presented Elmo, which takes advantage of emerging programmable switches and the unique characteristics of data center networks to entirely encode multicast group information inside the packet header [3], sharing a similar design with XCast [41]. To deal with link and switch failures, rerouting mechanisms are designed along with these proposals [2], [3], [42], [43]. Besides the construction of multicast IP routing, several recent proposals also design algorithms to manage the optical network topology to optimize intra- or inter- datacenter multicast transfers [44]–[46]. While orthogonal to them, PAM aims at controlling the rate of multicast. The scheduling header adopted by PAM is encapsulated as UDP payloads; thus, PAM can be integrated with all these routing techniques by design.

Multicast congestion control. Basically, existing congestion control mechanisms proposed for multicast can be divided into two categories. In the first one, multicast source senders are considered to transmit data at fixed rates. Then, the problem of congestion control in such cases is reformulated as the problem of optimizing the construction and embedding of multicast trees to avoid congestions or make the congestion bounded [47], [48]. This type of design works well for streaming tasks and virtual network requests, as one can reserve link bandwidth for each multicast tree and control each tree's load via rate limiting. However, they are with low bandwidth efficiency and limited in use because they lack the ability to react to dynamic link congestions. Thus, they can not be used for the schedule of file disseminations as there are many other types of coexisting transfers whose sending rate might vary with time (e.g., latency-sensitive mice TCP flows [8], [23]). In the second category, the source of each multicast tree would perform TCP-alike congestion control algorithms based on the (dynamic selected) slowest receiver's feedbacks [10]–[12], [49]. These schemes enable multicast transfers to remove dynamic congestions. However, TCP-based scheduling is slow to coverage, and moreover, the *fair sharing* they are pursuing is proved to be far from optimal for file dissemination tasks [7], [13]. As a comparison, by taking advantages of emerged programmable switches, PAM designs explicit rate controls to achieve fast and customizable bandwidth allocations for multicast. Such a design is partly inspired by the recent advances on the optimized scheduling of unicast transfers in data center [7], [9], [21], [31]. PAM enriches this research topic by making priority-based rate scheduling work for data center multicast and implementable on P4-based programmable switches.

Multicast in Layer 7. Besides the implementation over IP layer, many application-layer (i.e., L7) multicast protocols [50] are also proposed to achieve data streaming [51], file dissemination [52], [53], and distributed messaging [54] for datacenter applications. In L7-based approaches, there is no need for switch supports and reliability is easily achieved by directly using TCP. However, since L7-based multicast is built on top of unicast, each packet would be transmitted multiple times, resulting in low bandwidth efficiency. Other solutions like Datacast [6] are able to avoid duplicated transmissions by letting intermediate switches cache the transmitted packets. However, they rely on clean-slate network architectures like content-centric networking (CCN), which are quite different from the data center networks we have today.

VII. Conclusion

This paper presents the design, analysis, implementation, and evaluation of PAM (Priority-based Adaptive Multicast), a preemptive, decentralized, and ready-deployable rate control protocol we designed for data center multicast. PAM provides a distributed mechanism to approximate a range of scheduling disciplines. By generating multicast transfers' priorities with appropriate strategies like SRSF and EDF, it is able to minimize the average multicast completion time and the number of deadline-missed transfers for intra-datacenter file dissemination. We prototype PAM in both P4 and ns-3. Extensive packet-level simulations indicate that PAM converges fast, has negligible impact on TCP traffic, and always performs near-optimal priority-based schedules. For example, in simulations, it outperforms the default fair sharing widely adopted by today's multicast protocols, up to 5.7× and 75%, respectively, on the two aforementioned scheduling goals.

Although there are abundant proposals for the schedule of data center unicast [7]–[9], [15], [18], [20]–[23], [34], the optimization scheduling of data center multicast has been neglected for a long time. We identify this important topic and propose a case design of PAM. Currently, PAM focuses on optimizing intra-datacenter multicast transfers in clouds owned

by a single tenant. As the future work, we would extend PAM to support priority-based yet performance-isolated schedule for multicast transfers belonging to multiple tenants, in which the transfer of one tenant would like to not be impacted by those of another [55]. Moreover, to achieve low latency and high bandwidth data delivery for time-sensitive application, more and more data center networks are employing lossless network infrastructure for remote direct memory access (RDMA) [56], [57]. How PAM would work in these environments is still unknown, and we leave it as future work.

REFERENCES

[1] D. Li, H. Cui *et al.*, "Scalable data center multicast using multi-class bloom filter," in *ICNP*, 2011, pp. 266–275.
[2] X. Li and M. J. Freedman, "Scaling ip multicast on datacenter topologies," in *CoNEXT*, 2013, pp. 61–72.
[3] M. Shahbaz, L. Suresh *et al.*, "Elmo: Source routed multicast for public clouds," in *SIGCOMM*, 2019, pp. 458–471.
[4] S. Islam, N. Muslim, and J. W. Atwood, "A survey on multicasting in software-defined networking," *IEEE Communications Surveys Tutorials*, vol. 20, no. 1, pp. 355–387, Firstquarter 2018.
[5] B. Burns, B. Grant *et al.*, "Borg, omega, and kubernetes," *ACM Queue*, vol. 14, pp. 70–93, 2016.
[6] J. Cao, C. Guo *et al.*, "Datacast: A scalable and efficient reliable group data delivery service for data centers," *IEEE Journal on Selected Areas in Communications*, vol. 31, no. 12, pp. 2632–2645, Dec 2013.
[7] C.-Y. Hong, M. Caesar, and P. B. Godfrey, "Finishing flows quickly with preemptive scheduling," in *SIGCOMM*, 2012, pp. 127–138.
[8] M. Alizadeh, S. Yang *et al.*, "pfabric: Minimal near-optimal datacenter transport," in *SIGCOMM*, 2013, pp. 435–446.
[9] Y. Lu, G. Chen *et al.*, "One more queue is enough: Minimizing flow completion time with explicit priority notification," in *IEEE INFOCOM*, May 2017, pp. 1–9.
[10] L. Rizzo, "Pgmcc: A tcp-friendly single-rate multicast congestion control scheme," in *SIGCOMM*, 2000, pp. 17–28.
[11] J. Li, M. Yuksel, and S. Kalyanaraman, "Explicit rate multicast congestion control," *Comput. Netw.*, vol. 50, no. 15, pp. 2614–2640, 2006.
[12] T. Zhu, F. Wang *et al.*, "Mctcp: Congestion-aware and robust multicast tcp in software-defined networks," in *24th IWQoS*, June 2016, pp. 1–10.
[13] S. Luo, H. Yu *et al.*, "Towards practical and near-optimal coflow scheduling for data center networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 11, pp. 3366–3380, Nov 2016.
[14] N. K. Sharma, A. Kaufmann *et al.*, "Evaluating the power of flexible packet processing for network resource allocation," in *NSDI*, 2017.
[15] A. Sivaraman, A. Cheung *et al.*, "Packet transactions: High-level programming for line-rate switches," in *SIGCOMM*, 2016, pp. 15–28.
[16] "P4 language and related specifications," https://p4.org/specs/.
[17] "Barefoot tofino," https://barefootnetworks.com/products/brief-tofino/.
[18] M. Alizadeh, A. Greenberg *et al.*, "Data center tcp (dctcp)," in *SIGCOMM*, 2010, pp. 63–74.
[19] W. Bai, L. Chen *et al.*, "Information-agnostic flow scheduling for commodity data centers," in *NSDI*, 2015, pp. 455–468.
[20] K. Nagaraj, D. Bharadia *et al.*, "Numfabric: Fast and flexible bandwidth allocation in datacenters," in *SIGCOMM*, 2016, pp. 188–201.
[21] C. Wilson, H. Ballani *et al.*, "Better never than late: Meeting deadlines in datacenter networks," in *SIGCOMM*, 2011, pp. 50–61.
[22] B. Vamanan, J. Hasan, and T. Vijaykumar, "Deadline-aware datacenter tcp (d2tcp)," in *SIGCOMM*, 2012, pp. 115–126.
[23] L. Chen, K. Chen *et al.*, "Scheduling mix-flows in commodity datacenters with karuna," in *SIGCOMM*, 2016, pp. 174–187.
[24] S. Kandula, S. Sengupta *et al.*, "The nature of data center traffic: Measurements & analysis," in *ACM IMC*, 2009, pp. 202–208.
[25] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *ACM IMC*, 2010, pp. 267–280.
[26] A. Roy, H. Zeng *et al.*, "Inside the social network's (datacenter) network," in *SIGCOMM*, 2015, pp. 123–137.
[27] S. Luo, H. Yu *et al.*, "Minimizing average coflow completion time with decentralized scheduling," in *IEEE ICC*, June 2015, pp. 307–312.
[28] S. Luo, H. Yu, and L. Li, "Decentralized deadline-aware coflow scheduling for datacenter networks," in *IEEE ICC*, May 2016, pp. 1–6.
[29] T. Speakman, J. Crowcroft *et al.*, "Pgm reliable transport protocol specification," Internet Requests for Comments, RFC Editor, RFC 3208, December 2001, http://www.rfc-editor.org/rfc/rfc3208.txt.
[30] B. Adamson, C. Bormann *et al.*, "Nack-oriented reliable multicast (norm) transport protocol," Internet Requests for Comments, RFC Editor, RFC 5740, November 2009.
[31] F. R. Dogar, T. Karagiannis *et al.*, "Decentralized task-aware scheduling for data center networks," in *SIGCOMM*, 2014, pp. 431–442.
[32] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: A survey," *Internet Math.*, vol. 1, no. 4, pp. 485–509, 2003.
[33] X. Jin, X. Li *et al.*, "Netcache: Balancing key-value stores with fast in-network caching," in *SOSP*, 2017, pp. 121–136.
[34] M. Alizadeh, T. Edsall *et al.*, "Conga: Distributed congestion-aware load balancing for datacenters," in *SIGCOMM*, 2014, pp. 503–514.
[35] P. Bosshart, G. Gibb *et al.*, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn," in *SIGCOMM*, 2013, pp. 99–110.
[36] L. Jose, L. Yan *et al.*, "Compiling packet programs to reconfigurable switches," in *NSDI*, 2015, pp. 103–115.
[37] "Behavioral model (bmv2)," https://github.com/p4lang/behavioral-model.
[38] J. Perry, A. Ousterhout *et al.*, "Fastpass: A centralized "zero-queue" datacenter network," in *SIGCOMM*, 2014, pp. 307–318.
[39] D. R. Cheriton and S. E. Deering, "Host groups: A multicast extension for datagram internetworks," in *SIGCOMM*, 1985, pp. 172–179.
[40] D. Li, Y. Li *et al.*, "Esm: Efficient and scalable data center multicast routing," *IEEE/ACM Trans. Netw.*, vol. 20, no. 3, pp. 944–955, 2012.
[41] W. Jia, "A scalable multicast source routing architecture for data center networks," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 1, pp. 116–123, January 2014.
[42] D. Li, M. Xu *et al.*, "Reliable multicast in data center networks," *IEEE Transactions on Computers*, vol. 63, no. 8, pp. 2011–2024, Aug 2014.
[43] S. Luo, H. Xing, and K. Li, "Near-optimal multicast tree construction in leaf-spine data center networks," *IEEE Systems Journal*, pp. 1–4, 2019.
[44] L. Luo, K.-T. Foerster *et al.*, "Dartree: Deadline-aware multicast transfers in reconfigurable wide-area networks," in *27th IWQoS*, 2019.
[45] ——, "Deadline-aware multicast transfers in software-defined optical wide-area network," *IEEE Journal on Selected Areas in Communications*, 2020.
[46] ——, "Splitcast: Optimizing multicast flows in reconfigurable datacenter networks," in *IEEE INFOCOM*, 2020.
[47] Z. Guo, J. Duan, and Y. Yang, "On-line multicast scheduling with bounded congestion in fat-tree data center networks," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 1, pp. 102–115, 2014.
[48] Z. Guo and Y. Yang, "On nonblocking multicast fat-tree data center networks with server redundancy," *IEEE Transactions on Computers*, vol. 64, no. 4, pp. 1058–1073, April 2015.
[49] J. Widmer and M. Handley, "Extending equation-based congestion control to multicast applications," in *SIGCOMM*, 2001, pp. 275–285.
[50] M. Hosseini, D. T. Ahmed *et al.*, "A survey of application-layer multicast protocols," *Commun. Surveys Tuts.*, vol. 9, no. 3, pp. 58–74, Jul. 2007.
[51] J. Reich, O. Laadan *et al.*, "Vmtorrent: Scalable p2p virtual machine streaming," in *CoNEXT*, 2012, pp. 289–300.
[52] L. Luo, H. Yu, and Z. Ye, "Deadline-guaranteed point-to-multipoint bulk transfers in inter-datacenter networks," in *IEEE ICC*, 2018, pp. 1–6.
[53] L. Luo, Y. Kong *et al.*, "Deadline-aware fast one-to-many bulk transfers over inter-datacenter networks," *IEEE Transactions on Cloud Computing*, 2019.
[54] P. Dobbelaere and K. S. Esmaili, "Kafka versus rabbitmq: A comparative study of two industry reference publish/subscribe implementations: Industry paper," in *11th ACM International Conference on Distributed and Event-based Systems*, ser. DEBS '17, 2017, pp. 227–238.
[55] E. Zahavi, A. Shpiner *et al.*, "Links as a service (laas): Guaranteed tenant isolation in the shared cloud," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 5, pp. 1072–1084, May 2019.
[56] A. Shpiner, E. Zahavi, and O. Rottenstreich, "The buffer size vs link bandwidth tradeoff in lossless networks," in *IEEE 22nd Annual Symposium on High-Performance Interconnects*, Aug 2014, pp. 33–40.
[57] R. Mittal, A. Shpiner *et al.*, "Revisiting network support for rdma," in *SIGCOMM*, 2018, pp. 313–326.