QuickCast: Fast and Efficient Inter-Datacenter Transfers using Forwarding Tree Cohorts

Mohammad Noormohammadpour[1], Cauligi S. Raghavendra[1], Srikanth Kandula[2], Sriram Rao[2]
[1]Ming Hsieh Department of Electrical Engineering, University of Southern California
[2]Microsoft

***Abstract*—Large inter-datacenter transfers are crucial for cloud service efficiency and are increasingly used by organizations that have dedicated wide area networks between datacenters. A recent work uses multicast forwarding trees to reduce the bandwidth needs and improve completion times of point-to-multipoint transfers. Using a single forwarding tree per transfer, however, leads to poor performance because the slowest receiver dictates the completion time for all receivers. Using multiple forwarding trees per transfer alleviates this concern–the average receiver could finish early; however, if done naively, bandwidth usage would also increase and it is apriori unclear how best to partition receivers, how to construct the multiple trees and how to determine the rate and schedule of flows on these trees. This paper presents QuickCast, a first solution to these problems. Using simulations on real-world network topologies, we see that QuickCast can speed up the average receiver's completion time by as much as $10\times$ while only using $1.04\times$ more bandwidth; further, the completion time for all receivers also improves by as much as $1.6\times$ faster at high loads.**

***Index Terms*—Software Defined WAN; Datacenter; Scheduling; Completion Times; Replication**

I. Introduction

Software Defined Networking (SDN) is increasingly adopted across Wide Area Networks (WANs) [1]. SDN allows careful monitoring, management and control of status and behavior of networks offering improved performance, agility and ease of management. Consequently, large cloud providers, such as Microsoft [2] and Google [3], have built dedicated large scale WAN networks that can be operated using SDN which we refer to as SD-WAN. These networks connect dozens of datacenters for increased reliability and availability as well as improved utilization of network bandwidth reducing communication costs [4], [5].

Employing geographically distributed datacenters has many benefits in supporting users and applications. Replicating objects across multiple datacenters improves user-access latency, availability and fault tolerance. For example, Content Delivery Networks (CDNs) replicate objects (e.g. multimedia files) across many cache locations, search engines distribute large index updates across many locations regularly, and VMs are replicated across multiple locations for scale out of applications. In this context, *Point to Multipoint (P2MP)* transfers (also known as One-to-Many transfers) are necessary. A P2MP transfer is a special case of multicasting with a single sender and a fixed set of receivers that are known apriori. These properties together provide an opportunity for network optimizations, such as sizable reductions in bandwidth usage and faster completion times by using carefully selected forwarding trees.

We review several approaches for performing P2MP transfers. One can perform P2MP transfers as many independent point-to-point transfers [4]–[7] which can lead to wasted bandwidth and increased completion times. Internet multicasting approaches [8] build multicast trees gradually as new receivers join the multicast sessions, and do not consider the distribution of load across network links while connecting new receivers to the tree. This can lead to far from optimal multicast trees which are larger than necessary and poor load balancing. Application layer multicasting, such as [9], focuses on use of overlay networks for building virtual multicast trees. This may lead to poor performance due to limited visibility into network link level status and lack of control over how traffic is directed in the network. Peer-to-peer file distribution techniques [10], [11] try to maximize throughput per receiver locally and greedily which can be far from a globally optimal solution. Centralized multicast tree selection approaches have been proposed [12], [13] that operate on regular and structured topologies of networks inside datacenters which cannot be directly applied to inter-DC networks. Other related research either does not consider elasticity of inter-DC transfers which allows them to change their transmission rate according to available bandwidth [14], [15] or inter-play among may inter-DC transfers for global network-wide optimization [16], [17].

We recently presented a solution called DCCast [18] that reduces tail completion times for P2MP transfers. DCCast employs a central traffic engineering server with a global view of network status, topology and resources to select forwarding trees over which traffic flows from senders to all receivers. This approach combines good visibility into and control over networks offered by SDN with bandwidth savings achieved by reducing the number of links used to make data transfers. OpenFlow [19], a dominant SDN protocol, has supported forwarding to multiple outgoing ports since v1.1 [20]; for example, by using Group Tables and the `OFPGT_ALL` flag. Several recent SDN switches support this feature [21]–[24].

In this paper, we propose a new rate-allocation and tree selection technique called **QuickCast** with the aim of *minimizing average completion times* of inter-DC transfers. QuickCast reduces completion times by replacing a large forwarding tree with multiple smaller trees each connected to a subset of

This is an extended version of a paper accepted for publication in IEEE INFOCOM 2018, Honolulu, HI, USA

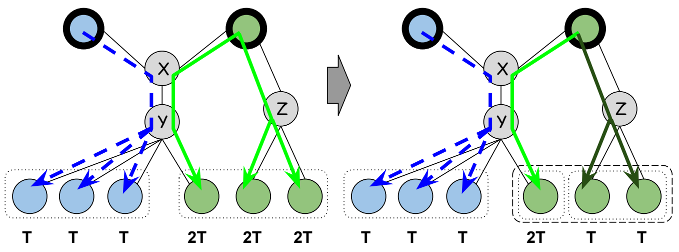


Fig. 1. Partitioning receivers into subsets can improve mean completion times (this figure assumes FCFS rate-allocation policy, dashed blue transfer arrives slightly earlier, all links have equal capacity of 1)

receivers which we refer to as a cohort of forwarding trees. Next, QuickCast applies the Fair Sharing scheduling policy which we show through simulations and examples minimizes contention for available bandwidth across P2MP transfers compared to other scheduling policies.

Despite offering bandwidth savings and reduced tail completion times, DCCast can suffer from significant increase in completion times when P2MP transfers have many receivers. As the number of receivers increases, forwarding trees grow large creating many overlapping edges across transfers, increasing contention and completion times. Since the edge with minimal bandwidth determines the overall throughput of a forwarding tree, we refer to this as **"Weakest Link"** problem. We demonstrate weakest link problem using a simple example. Figure 1 shows a scenario where two senders (top two nodes) are transmitting over forwarding trees and they share a link ($x \rightarrow y$). According to DCCast which uses the First Come First Serve (FCFS) policy, the dashed blue transfer (on left) which arrived just before the green (on right) is scheduled first delaying the beginning of green transfer until time T. As a result, all green receivers finish at time $2T$. By using multiple trees, in this case two for the green sender, each tree can be scheduled independently, and thereby reducing completion time of the two receivers on the right from $2T$ to T. That is, we create a new tree that does not have the *weakest* link, which in this scenario is $x \rightarrow y$ for both blue and green transfers.

To replace a large tree with a cohort of trees, we propose partitioning receiver sets of P2MP transfers into multiple subsets and using a separate forwarding tree per subset. This approach can significantly reduce completion times of P2MP transfers. We performed an experiment with DCCast over random topologies with 50 nodes to determine if there is benefit in partitioning all P2MP transfers. We simply grouped receivers into two subsets according to proximity, i.e., shortest path hop count between receiver pairs, and attached each partition with an independent forwarding tree (DCCast+2CL). As shown in Figure 2, this reduced completion times by 50% while increasing bandwidth usage by 6% (not shown).

To further reduce completion times and decrease additional bandwidth usage, partitioning should be only applied to P2MP transfers that benefit from it. For example, using more than one tree for the dashed blue transfer (on left) in Figure 1 will increase contention and will hurt completion times. By carefully selecting P2MP transfers that benefit from partitioning and using an independent forwarding tree per partition, we can considerably improve average completion times with small extra bandwidth usage. We refer to this approach as **selective partitioning**. We limit the number of partitions per transfer to two to minimize bandwidth overhead of additional edges, minimize contention due to overlapping trees and limit the number of network forwarding entries.

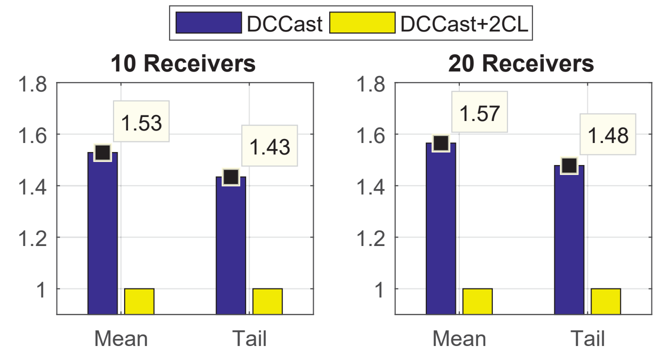


Fig. 2. Comparison of completion times (mean and tail) of DCCast vs. DCCast+2CL (two-clustering according to proximity) for P2MP transfers with 10 and 20 receivers per transfer (Normalized)

In addition to using multiple forwarding trees, we investigate various scheduling policies namely FCFS used in DCCast, Shortest Remaining Processing Time (SRPT) and Fair Sharing based on Max-Min Fairness (MMF) [25]. Although SRPT is optimal for minimizing mean completion times while scheduling traffic on a single link, we find that MMF beats SRPT by a large factor as forwarding trees grow (which causes many trees to share every link) and as offered load increases (which increases contention for using available bandwidth).

Using a cohort of forwarding trees per P2MP transfer can also increase reliability in two ways. First, by mitigating the effect of weak links, a number of receivers will complete reception earlier which reduces the probability of data loss due to failures. Second, in case of a link failure, using more than one tree reduces the probability of all receivers being affected. In case subsequent trees are constructed in a link disjoint manner, no single link failure can affect all receivers. In summary, we make the following contributions:

1) We formulate a general optimization problem for minimizing mean completion times while scheduling P2MP transfers over a general network.
2) We propose QuickCast, which mitigates the "Weakest Link" problem by partitioning receivers into multiple subsets and attaching each partition with an independent forwarding tree. Using multiple forwarding trees can improve average completion time and reliability with only a little additional bandwidth usage. We show that partitioning should only be applied to transfers that benefit from it, i.e., partitioning some transfers leads to increased contention and worse completion times.
3) We explore well-known scheduling policies (FCFS, SRPT and Fair Sharing) for central rate-allocation over forwarding trees and find MMF to be the most effective in maximizing overall bandwidth utilization and reduc-

ing completion times.

4) We perform extensive simulations using both synthetic and real inter-datacenter traffic patterns over real WAN topologies. We find that performance gain of QuickCast depends on the offered load and show that under heavy loads, QuickCast can improve mean times by as much as $10\times$ and tail times by as much as $1.57\times$ while imposing very small increase in typical bandwidth usage (only 4%) compared to DCCast.

The rest of this paper is organized as follows. In Section II, we state the P2MP scheduling problem, explain the constraints and variables used in the paper and formulate the problem as an optimization scenario. In Section III, we present QuickCast and the two procedures it is based on. In Section IV, we perform abstract simulations to evaluate QuickCast and in Section V we discuss practical considerations and issues. At the end, in Section VI, we provide a comprehensive overview of related research efforts.

II. Assumptions and Problem Statement

Similar to prior work on inter-datacenter networks [4]–[7], [18], [26], [27], we assume a SD-WAN managed by a Traffic Engineering (TE) server which receives transfer requests from end-points, performs rate-allocations and manages the forwarding plane. Transfers arrive at the TE server in an online fashion and are serviced as they arrive. Requests are specified with four parameters of arrival time, source, set of receivers and size (volume in bytes). End-points apply rate-limiting to minimize congestion. We consider a slotted timeline to allow for flexible rate-allocation while limiting number of rate changes to allow time to converge to specified rates and minimize rate-allocation overhead [6], [26]. We focus on long running transfers that deliver large objects to many datacenters such as applications in [5]. For such transfers, small delays are usually acceptable, including overhead of centralized scheduling and network configurations. To reduce configuration overhead (e.g. latency and control plane failures [28]), we assume that a forwarding tree is not changed once configured on the forwarding plane.

To reduce complexity we assume that end-points can accurately rate-limit data flows and that they quickly converge to required rates; that there are no packet losses due to congestion, corruption or errors; and that scheduling is done for a specific class of traffic meaning all requests have the same priority. In Section V, we will discuss ways to deal with cases when some of these assumptions do not hold.

A. Problem Formulation

Earlier we showed that partitioning of receiver sets can improve completion times via decreasing network contention. However, to further improve completion times, partitioning should be done according to transfer properties, topology and network status. Optimal partitioning for minimization of completion times is an open problem and requires finding solution to a complex joint optimization model that takes into account forwarding tree selection and rate-allocation.

TABLE I
Definition of variables

Variable	Definition
t and t_{now}	Some timeslot and current timeslot
N	Total number of receivers per transfer
e	A directed edge
C_e	Capacity of e in bytes per second
(x, y)	A directed edge from x to y
G	A directed inter-datacenter network graph
T	Some directed tree connecting a sender to its receivers
$\mathbf{V_G}$ and $\mathbf{V_T}$	Set$\langle\rangle$ of vertices of G and T
$\mathbf{E_G}$ and $\mathbf{E_T}$	Set$\langle\rangle$ of edges of G and T
B_e	Current available bandwidth on edge e
B_T	Current available bandwidth over tree T
δ	Width of a timeslot in seconds
R_i	A transfer request where $i \in \mathbf{I} = \{1 \dots I\}$
O_i	Data object associated with R_i
S_{R_i}	Source datacenter of R_i
A_{R_i}	Arrival time of R_i
$\mathcal{V}_{R_i}$	Original volume of R_i in bytes
$\mathcal{V}^r_{R_i}$	Residual volume of R_i, ($\mathcal{V}^r_{R_i} = \mathcal{V}_{R_i}$ at $t = A_{R_i}$)
$\mathbf{D}_{R_i}$	Set$\langle\rangle$ of destinations of R_i
n	Maximum subsets (partitions) allowed per receiver set
$\mathbf{P}_i^j$	Set$\langle\rangle$ of receivers of R_i in partition $j \in \{1, \dots, n\}$
q_i	Total bytes used to deliver O_i to $\mathbf{D}_{R_i}$
T_i^j	Forwarding tree for partition $j \in \{1, \dots, n\}$ of R_i
L_e	e's total outstanding load, i.e., $L_e = \sum_{\substack{i,j \\ e \in T_i^j}} \mathcal{V}^r_{R_i}$
$f_i^j(t)$	Transmission rate of R_i on T_i^j at timeslot t
$\gamma_i^j(t)$	Whether R_i is transmitted over T_i^j at timeslot t
$\theta^j_{i,e}$	Whether edge $e \in \mathbf{E_G}$ is on T_i^j
$\nu_{i,j,v}$	Whether $v \in \mathbf{D}_{R_i}$ is in $\mathbf{P}_i^j$
$\mathbf{M}_i^j$	$\{\boldsymbol{\nabla} \mid \boldsymbol{\nabla} \subset \mathbf{V_G},\ \boldsymbol{\nabla} \cap \{\mathbf{P}_i^j \cup S_{R_i}\} \neq \emptyset, (\mathbf{V_G} - \boldsymbol{\nabla}) \cap \{\mathbf{P}_i^j \cup S_{R_i}\} \neq \emptyset\}$
$E(\boldsymbol{\nabla})$	$\{e = (x, y) \mid x \in \boldsymbol{\nabla}, y \in (\mathbf{V_G} - \boldsymbol{\nabla})\}$

We first point out the most basic constraint out of using forwarding trees. Table I provides the list of variables we will use. For any tree T, packets flow from the source to receivers with same rate r_T that satisfies:

$$r_T \leq B_T = \min_{e \in \mathbf{E_T}} (B_e) \qquad (1)$$

We formulate the problem as an online optimization scenario. Assuming a set of requests $R_i,\ i \in \{1 \dots I\}$ already in the system, upon arrival of new request R_{I+1}, an optimization problem needs to be formulated and solved to find rate-allocations, partitions and forwarding trees. Figure 3 shows the overall optimization problem. This model considers up to $n \geq 1$ partitions per transfer. Demands of existing requests are updated to their residuals upon arrival of new requests.

The objective is formulated in a hierarchical fashion giving higher priority to minimizing mean completion times and then reducing bandwidth usage. The purpose of $\gamma_i^j(t)$ indicator variables is to calculate the mean times: the latest timeslot over which we have $f_i^j(t) > 0$ determines the completion time of partition j of request i. These completion times are

$$\text{minimize} \quad \sum_{i\in \mathbf{I}} \Big(\sum_{j\in\{1,\dots,n\}} |\mathbf{P}_i^j| \ \big(\sum_{t > A_{R_{I+1}}} (t - A_{R_{I+1}}) \gamma_i^j(t) \prod_{t'>t} (1 - \gamma_i^j(t')) \big) \Big) + \Big(\frac{\sum_{i\in\mathbf{I}} q_i}{n \ |\mathbf{E_G}| \ \sum_{i\in\mathbf{I}} \mathcal{V}_{R_i}} \Big)$$

subject to

Calculate total bandwidth usage:

$$(1) \quad q_i = \sum_{e\in\mathbf{E_G}} \big(\sum_{j\in\{1,\dots,n\}} \theta_{i,e}^j \big) \mathcal{V}_{R_i}^r \qquad \forall i$$

Demand satisfaction constraints:

$$(2) \quad \sum_t \gamma_i^j(t) f_i^j(t) = \frac{\mathcal{V}_{R_i}^r}{\delta} \qquad \forall i, j$$

Capacity constraints:

$$(3) \quad \sum_{i\in\mathbf{I}} \sum_{j\in\{1,\dots,n\}} \theta_{i,e}^j f_i^j(t) \le C_e \qquad \forall j, t, e$$

Steiner tree constraints [29]:

$$(4) \quad \sum_{e\in E(\nabla)} \theta_{i,e}^j \ge 1 \qquad \forall i, j, \nabla \in \mathbf{M}_i^j$$

Basic range constraints:

$$(5) \quad \gamma_i^j(t) = 0, \ f_i^j(t) = 0 \qquad \forall i, j, t < A_{R_{I+1}}$$

$$(6) \quad f_i^j(t) \ge 0 \qquad \forall i, j, t$$

$$(7) \quad \theta_{i,e}^j \in \{0,1\} \qquad \forall i, j, e$$

$$(8) \quad \gamma_i^j(t) \in \{0,1\} \qquad \forall i, j, t$$

$$(9) \quad \nu_{i,j,v} \in \{0,1\} \qquad \forall i, j, v \in \mathbf{D}_{R_i}$$

$$(10) \quad \gamma_i^j(t) = 0 \qquad \forall i, j, t < A_{R_i}$$

Fig. 3. Online optimization model, variables defined in Table I

then multiplied by partition size to create the total sum of completion times per receiver. Constraint 4 ascertains that there is a connected subgraph across sender and receivers per partition per request which is similar to constraints used to find minimal edge Steiner trees [29]. Since our objective is an increasing function of bandwidth, these constraints eventually lead to minimal trees connecting any sender to its receivers while not increasing mean completion times (the second part of objective that minimizes bandwidth is necessary to ensure no stray edges).

B. Challenges

We focus on two objectives of first minimizing completion times of data transfers and minimizing total bandwidth usage. This is a complex optimization problem for a variety of reasons. First, breaking a receiver set to several partitions leads to exponential number of possibilities. Moreover, the optimization version of Steiner tree problem which aims to find minimal edge or minimal weight trees is a hard problem [29]. Completion times of transfers then depend on how partitions are formed and which trees are used to connect partitions to senders. In addition, the scenario is naturally an online problem which means even if we were able to compute an optimal solution for a given set of transfers in a short amount of time, we still would not be able to compute a solution that is optimal over longer periods of time due to incomplete knowledge of future arrivals.

III. QuickCast

We present our heuristic approach in Algorithm 1 called QuickCast with the objective of reducing mean completion times of elastic P2MP inter-DC transfers. We first review concepts behind design of this heuristic, namely rate-allocation, partitioning, forwarding tree selection and selective partitioning. Next, we discuss how Algorithm 1 realizes these concepts using two procedures one executed upon arrival of new transfers and the other per timeslot.

A. Rate-allocation

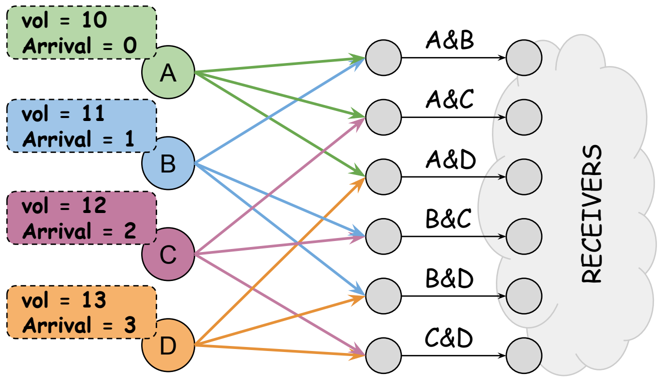
To compute rates per timeslot, we explore well-known scheduling policies: FCFS, SRPT and Fair Sharing. Although simple, FCFS can lead to increased mean times if large transfers block multiple edges by fully utilizing them. SRPT is known to offer optimal mean times over a single link but may lead to starvation of large transfers. QuickCast uses Fair Sharing based on MMF policy.

To understand effect of different scheduling policies, let us consider the following example. Figure 4 shows a scenario where multiple senders have initiated trees with multiple branches and they share links along the way to their receivers. SRPT gives a higher priority to the top transfer with size 10 and then to the next smallest transfer and so on. When the first transfer is being sent, all other transfers are blocked due to shared links. This occurs again when the next transfer begins. Scheduling according to FCFS leads to same result. In this example, mean completion times for both FCFS and SRPT is about $1.16\times$ larger than MMF. In Section IV, we perform simulation experiments that confirm the outcome in this example. We find that as trees grow larger and under high utilization, the benefit of using MMF over FCFS or SRPT becomes more significant due to increased contention. We also realize that tail times grow much faster for SRPT compared to both MMF and FCFS (since it also suffers from the starvation problem) while scheduling over forwarding trees with many receivers, and that increases SRPT's mean completion times.

B. Partitioning

There are two configuration parameters for grouping receivers into multiple partitions: *partitioning criteria* and *number of partitions*. In general, partitioning may lead to higher bandwidth usage. However, it may be the case that a small increase in bandwidth usage can considerably improve completion times. Efficient and effective partitioning to minimize completion times and bandwidth usage is a complex open problem. We discuss our approach in the following.

Partitioning Criteria: We focus on minimizing extra bandwidth usage while breaking large forwarding trees via partitioning. Generally, one could select partitions according to current network conditions, such as distribution of load



	Completion Times				
	A	B	C	D	Mean
FCFS / SRPT	10	21	33	46	27.5
MMF	19	23	26	27.5	23.87

Fig. 4. Fair Sharing can offer better mean times compared to both SRPT and FCFS while scheduling over forwarding trees, all links have capacity of 1

across edges. However, we notice that network conditions are continuously changing as current transfers finish and new transfers arrive. Minimizing bandwidth on the other hand appears as a globally desirable objective for partitioning and was hence chosen. To find partitions, QuickCast groups receivers according to proximity until we are left with desired number of groups each forming a partition. Distance between two receivers is computed as the number of hops on the shortest path between them. With this approach, a partition requires minimal number of edges to connect the nodes within. Reassigning any receiver to other partitions will increase the number of edges and thus consumed bandwidth.

Number of Partitions: The right number of partitions per transfer depends on factors such as topology, number of receivers, forwarding trees of other transfers, source and destinations of a transfer, and overall network load. In the extreme case of N partitions, a P2MP transfer is broken into N unicast transfers which significantly increases bandwidth usage. Partitioning is most effective if forwarding trees assigned to partitions do not increase overall contention for network bandwidth, i.e., the number of overlapping edges across new trees is minimal. Therefore, increasing number of partitions to more than connectivity degree of datacenters may offer minor gains or even loss of performance (e.g., in case of Google B4 [5], the minimum and maximum connectivity degrees are 2 and 4, respectively). From the practical aspect, number of partitions and hence forwarding trees determines the number of Group Table rules that need to be setup in network switches. Therefore, we focus on partitioning receivers into up to 2 groups each assigned an independent forwarding tree. Exploration of effects of more partitions is left for future work.

C. Forwarding Tree Selection

After computing two partitions, QuickCast assigns an independent forwarding tree per partition using tree selection approach presented in [18] which was shown to provide high bandwidth savings and improved completion times. It operates by giving a weight of $W_e = (L_e + \mathcal{V}_R)$ (see Table I for definition of variables) and then selecting the minimum weight forwarding tree. This technique allows load balancing of P2MP data transfers over existing trees according to total bandwidth scheduled on the edges. It also takes into account transfer volumes while selecting trees. Particularly, larger transfers are most likely assigned to smaller trees to minimize bandwidth usage while smaller transfers are assigned to least loaded trees (regardless of tree size). This approach becomes more effective when a small number of transfers are orders of magnitude larger than median [30], as number of larger forwarding trees is usually significantly larger than smaller trees on any graph.

D. Selective Partitioning

Partitioning is beneficial only if it decreases or minimally increases bandwidth usage and contention over resources which necessitates selectively partitioning the receivers. When we chose the two partitions by grouping receivers and after selecting a forwarding tree for every group, QuickCast calculates the total weight of each forwarding tree by summing up weights of their edges. We then compare sum of these two weights with no partitioning case where a single tree was used. If the total weight of two smaller trees is less than some *partitioning factor* (shown as p_f) of the single tree case, we accept to use two trees. If p_f is close to 1.0, partitioning occurs only if it incurs minimal extra weight, i.e., $(p_f - 1)$ times weight of the single forwarding tree that would have been chosen if we applied no partitioning. With this approach, we most likely avoid selection of subsequent trees that are either much larger or much more loaded than the initial tree in no partitioning case. Generally, an effective p_f is a function of traffic distribution and topology. According to our experiments with several traffic distributions and topologies, choosing it in the range of $1.0 \le p_f \le 1.1$ offers the best completion times and minimal bandwidth usage.

E. QuickCast Algorithm

A TE server is responsible for managing elastic transfers over inter-DC network. Each partition of a transfer is managed independently of other partitions. We refer to a transfer partition as **active** if it has not been completed yet. TE server keeps a list of active transfer partitions and tracks them at every timeslot. A TE server running QuickCast algorithm uses two procedures as shown in Algorithm 1.

Submit$(\mathrm{R}, \mathrm{n}, \mathrm{p_f})$**:** This procedure is executed upon arrival of a new P2MP transfer R. It performs partitioning and forwarding tree selection for the new transfer given its volume, source and destinations. We consider the general case where we may have up to n partitions per transfer. First, we compute edge weights based on current traffic matrix and volume of new transfer. We then build the agglomerative hierarchy of receivers using average linkage and considering proximity as clustering metric. Agglomerative clustering is a bottom up approach where at every level the two closest clusters are merged forming one cluster. The distance of any two clusters is computed using average linkage which is the average over pairwise distances of nodes in the two clusters. The distance between every pair of receivers is the number of edges on the shortest path from one to the other. It should be noted that although our networks are directed, all edges are considered to be bidirectional and so the distance in either direction between any two nodes should be the same. When the hierarchy is ready, we start from the level where there are n clusters (or at the bottom if total number of receivers is less than or equal to n) and compute the total weight of n forwarding trees (minimum weight Steiner trees) to these clusters. We move forward with this partitioning if the total weight is less than p_f times weight of the forwarding tree that would have been selected if we grouped all receivers into one partition. Otherwise, the same process is repeated while moving up one level in the clustering hierarchy (one less cluster). If we accept a partitioning, this procedure first assigns a forwarding tree to every partition while continuously updating edge weights. It then returns the partitions and their forwarding trees.

DispatchRates(): This procedure is executed at the beginning of every timeslot. It calculates rates per active transfer partition and according to MMF rate-allocation policy. New transfers arriving somewhere within a timeslot are allocated rates starting next timeslot. To calculate residual demands needed for rate calculations, senders report back the actual volume of data delivered during past timeslot per partition. This allows QuickCast to cope with inaccurate rate-limiting and packet losses which may prevent a transfer from fully utilizing its allotted share of bandwidth.

IV. Evaluations

We considered various topologies and transfer size distributions as in Tables II and III. For simplicity, we considered a uniform capacity of 1.0 for all edges, accurate rate-limiting at end-points, no dropped packets due to congestion or corruption, and no link failures. Transfer arrival followed a Poisson distribution with rate of λ. For all simulations, we considered a partitioning factor of $p_f = 1.1$ and timeslot length of $\delta = 1.0$. Unless otherwise stated, we assumed a fixed $\lambda = 1.0$. Also, for all traffic distributions, we considered an average demand equal to volume of 20 full timeslots per transfer. For heavy-tailed distribution that is based on Pareto distribution, we used a minimum transfer size equal to that of 2 full timeslots. Finally, to prevent generation of intractably large transfers, we limited maximum transfer volume to that of 2000 full timeslots. We focus on scenarios with no link failures to evaluate gains.

A. Comparison of Scheduling Policies over Forwarding Trees

We first compare the performance of three well-known scheduling policies of FCFS, SRPT and Fair Sharing (based on

Algorithm 1: QuickCast

```
Submit ($\mathrm{R}, \mathrm{n}, \mathrm{p_f}$)
    Input: $\mathrm{R}(\mathcal{V}_R, S_R, \mathbf{D}_R)$, $n$ (= 2 in this paper), $p_f$, $G$, $L_e$
           for $\forall e \in \mathbf{E_G}$ (Variables defined in Table I)
    Output: Pairs of (Partition, Forwarding Tree)
    $\forall \alpha, \beta \in \mathbf{D}_R,\ \alpha \neq \beta$, $\mathrm{DIST}_{\alpha,\beta} \leftarrow$ number of edges on
      the shortest path from $\alpha$ to $\beta$;
    To every edge $e \in \mathbf{E_G}$, assign weight $W_e = (L_e + \mathcal{V}_R)$;
    Find the minimum weight Steiner tree $T_R$ that connects
      $S_R \cup \mathbf{D}_R$ and its total weight $W_{T_R}$;
    for $k = n$ to $k = 2$ by $-1$ do
        Agglomeratively cluster $\mathbf{D}_R$ using average linkage
          and distance metric DIST calculated in previous
          line until only $k$ clusters left forming
          $\mathbf{P}^i_R,\ i \in \{1, \ldots, k\}$;
        foreach $i \in \{1, \ldots, k\}$ do
            Find $W_{T_{\mathbf{P}^i_R}}$, weight of minimum weight Steiner
              tree that connects $S_R \cup \mathbf{P}^i_R$;
        if $\sum_{i \in \{1,\ldots,k\}} W_{T_{\mathbf{P}^i_R}} \leq p_f \times W_{T_R}$ then
            foreach $i \in \{1, \ldots, k\}$ do
                Find the minimum weight Steiner tree $T_{\mathbf{P}^i_R}$
                  that connects $S_R \cup \mathbf{P}^i_R$;
                $L_e \leftarrow L_e + \mathcal{V}_R,\ \forall e \in T_{\mathbf{P}^i_R}$;
                Update $W_e = (L_e + \mathcal{V}_R)$ for all $e \in \mathbf{E_G}$;
            return $\mathbf{P}^i_R$ as well as $T_{\mathbf{P}^i_R},\ \forall i \in \{1, \ldots, k\}$;
    $L_e \leftarrow L_e + \mathcal{V}_R,\ \forall e \in T_R$;
    return $\mathbf{D}_R$ and $T_R$;

DispatchRates ()
    Input: Set of active request partitions $\mathbf{P}$, their current
           residual demands and forwarding trees $\mathcal{V}^r_P$ and
           $T_P,\ \forall P \in \mathbf{P}$, and timeslot width $\delta$
    Output: Rate per active request per partition for next
            timeslot
    $\mathrm{COUNT}_e \leftarrow$ number of forwarding trees $T_P,\ \forall P \in \mathbf{P}$
      sharing edge $e,\ \forall e \in \mathbf{E_G}$;
    $\mathbf{P}' \leftarrow \mathbf{P}$ and $\mathrm{CAP}_e \leftarrow 1,\ \forall e \in \mathbf{E_G}$;
    while $|\mathbf{P}'| > 0$ do
        foreach $P \in \mathbf{P}'$ do
            $\mathrm{SHARE}_P \leftarrow \min_{e \in T_P}(\frac{\mathrm{CAP}_e}{\mathrm{COUNT}_e})$;
        $P' \leftarrow$ a partition $P$ with minimum $\mathrm{SHARE}_P$ value;
        $\mathrm{RATE}_{P'} \leftarrow \min(\mathrm{SHARE}_{P'}, \frac{\mathcal{V}^r_{P'}}{\delta})$;
        $\mathbf{P}' \leftarrow \mathbf{P}' - \{P'\}$;
        $\mathrm{COUNT}_e \leftarrow \mathrm{COUNT}_e - 1,\ \forall e \in T_{P'}$;
        $\mathrm{CAP}_e \leftarrow \mathrm{CAP}_e - \mathrm{RATE}_{P'},\ \forall e \in T_{P'}$;
    return $\mathrm{RATE}_P,\ \forall P \in \mathbf{P}$
```

TABLE II
VARIOUS TOPOLOGIES USED IN EVALUATION

Name	**Description**
Random	Randomly generated and strongly connected with 50 nodes and 150 edges. Each node has a minimum connectivity of two.
GScale [5]	Connects Google datacenters across the globe with 12 nodes and 19 links.
Cogent [31]	A large backbone and transit network that spans across USA and Europe with 197 nodes and 243 links.

TABLE III
TRANSFER SIZE DISTRIBUTIONS USED IN EVALUATION

Name	**Description**
Light-tailed	According to Exponential distribution.
Heavy-tailed	According to Pareto distribution.
Facebook *Cache-Follower* [30]	Generated across Facebook inter-datacenter networks running cache applications.
Facebook *Hadoop* [30]	Generated across Facebook inter-datacenter networks running geo-distributed analytics.

MMF). We used the weight assignment in [18] for forwarding tree selection and considered Random topology in Table II. We considered both light-tailed and heavy-tailed distributions. All policies used almost identical amount of bandwidth (not shown). Under light load, we obtained results similar to scheduling traffic on a single link where SRPT performs better than Fair Sharing (not shown). Figure 5 shows the results of our experiment under heavy load. When the number of receivers is small, SRPT is the best policy to minimize mean times. However, as we increase the number of receivers (larger trees), Fair Sharing offers better mean and tail times. This simply occurs because the contention due to overlapping trees caused by prioritizing transfers over one another (either according to residual size in case of SRPT or arrival order in case of FCFS) increases as more transfers are submitted or as transfers grow in size.

B. Bandwidth usage of partitioning techniques

We considered three partitioning techniques and measured the average bandwidth usage over multiple runs and many timeslots. We used the topologies in Table II and traffic patterns of Table III. Figure 6 shows the results. We calculated the lower bound by considering a single minimal edge Steiner tree per transfer. Other schemes considered are: *Random(Uniform Dist)* breaks each set of receivers into two partitions by randomly assigning each receiver to one of the two partitions with equal probability, *Agg(proximity between receivers)* clusters receivers according to closeness to each other, and *Agg(closeness to source)* clusters receivers according to their distance from source (receivers closer to source are bundled together). As can be seen, *Agg(proximity between receivers)*, which is used by QuickCast, provides the least bandwidth overhead (up to 17% to lower bound). In general, breaking receivers into subsets that are attached to a sender with minimal bandwidth usage is an open problem.

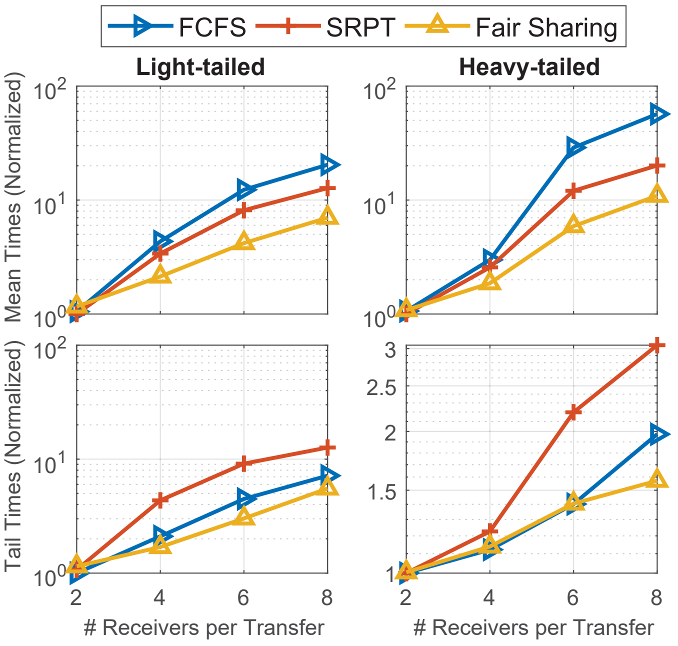


Fig. 5. Performance of three well-known scheduling policies under heavy load (forwarding tree selection according to DCCast)

TABLE IV
EVALUATION OF PARTITIONING TECHNIQUES FOR P2MP TRANSFERS

Scheme	**Details**
QuickCast	Algorithm 1 (*Selective Partitioning*).
QuickCast(NP)	QuickCast with no partitioning applied.
QuickCast(TWO)	QuickCast with two partitions always.

C. QuickCast with different partitioning approaches

We compare three partitioning approaches shown in Table IV. We considered receiver sets of 5 and 10 with both light-tailed and heavy-tailed distributions. We show both mean (top row) and tail (bottom row) completion times in the form of a CDF in Figure 7. As expected, when there is no partitioning, all receivers complete at the same time (vertical line in CDF). When partitioning is always applied, completion times can jump far beyond the no partitioning case due to unnecessary creation of additional weak links. The benefit of QuickCast is that it applies partitioning selectively. The amount of benefit obtained is a function of partitioning factor p_f (which for the topologies and traffic patterns considered here was found to be most effective between 1.0 and 1.1 according to our experiments, we used 1.1). With QuickCast, the fastest receiver can complete up to 41% faster than the slowest receiver and even the slowest receiver completes up to 10% faster than when no partitioning is applied.

D. QuickCast vs. DCCast

We now compare QuickCast with DCCast using real topologies and inter-datacenter transfer size distributions, namely **GScale(Hadoop)** and **Cogent(Cache-Follower)** shown in Tables II and III. Figure 8 shows the results. We considered 10 receivers per P2MP transfer. In all cases, QuickCast uses up

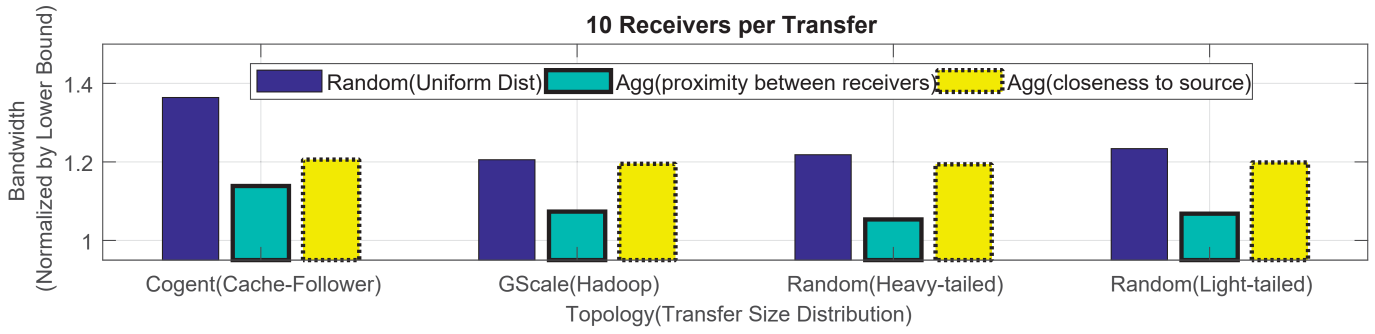


Fig. 6. Bandwidth usage of various partitioning techniques (lower is better)

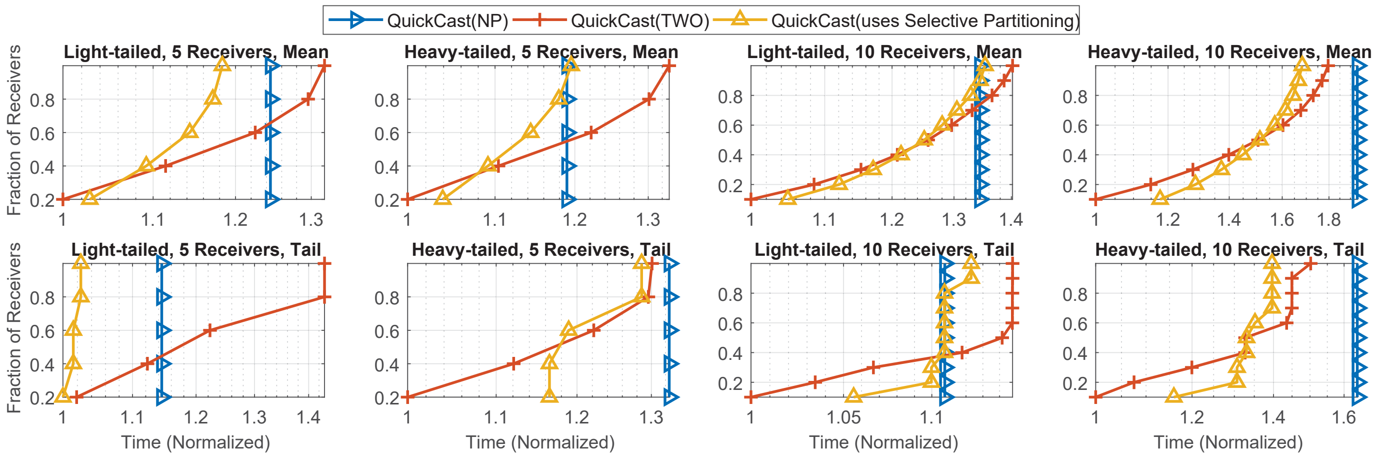


Fig. 7. Comparison of partitioning approaches in Table IV

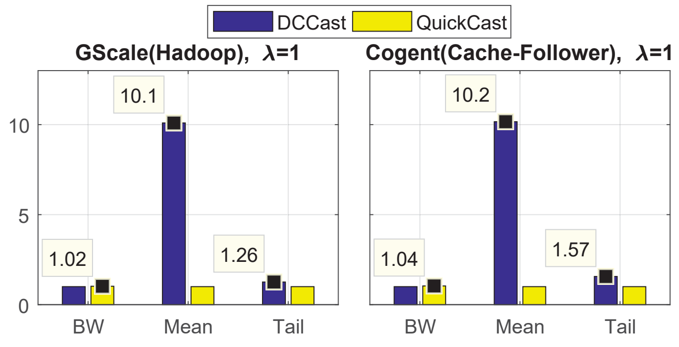
to 4% more bandwidth. For lightly loaded scenarios where $\lambda = 0.01$, QuickCast performs up to 78% better in mean times, but about 35% worse in tail times. The loss in tail times is a result of rate-allocation policy: FCFS performs better in tail times compared to Fair Sharing under light loads where contention due to overlapping trees is negligible (similar to single link case when all transfers compete for one resource). For heavily loaded scenarios where $\lambda = 1$, network contention due to overlapping trees is considerable and therefore QuickCast has been able to reduce mean times by about 10× and tail times by about 57%. This performance gap continues to increase in favor of QuickCast as offered load grows further. In general, operators aim to maximize network utilization over dedicated WANs [4], [5] which could lead to heavily loaded time periods. Such scenarios may also appear as a result of bursty transfer arrivals.

In a different experiment, we studied the effect of number of replicas of data objects on performance of DCCast and QuickCast as shown in Figure 9 (please notice the difference in vertical scale of different charts). Bandwidth usage of both schemes were almost identical (QuickCast used less than 4% extra bandwidth in the worst case). We considered two operating modes of lightly to moderately loaded ($\lambda = 0.01$) and moderately to heavily loaded ($\lambda = 0.1$). QuickCast offers most benefit when number of copies grows. When the number of copies is small, breaking receivers into multiple sets may provide limited benefit or even degrade performance as resulting partitions will be too small. This is why mean and tail times degrade by up to 5% and 40% across both topologies and traffic patterns when network is lightly loaded, respectively. Under increasing load and with more copies, it can be seen that QuickCast can reduce mean times significantly, i.e., by as much as 6× for **Cogent(Cache-Follower)** and as much as 16× for **GScale(Hadoop)**, respectively. The sudden increase in tail times for GScale topology is because this network has only 12 nodes which means partitioning while making 10 copies may most likely lead to overlapping edges across partitioned trees and increase completion times. To address this problem, one could reduce p_f to minimize unnecessary partitioning.

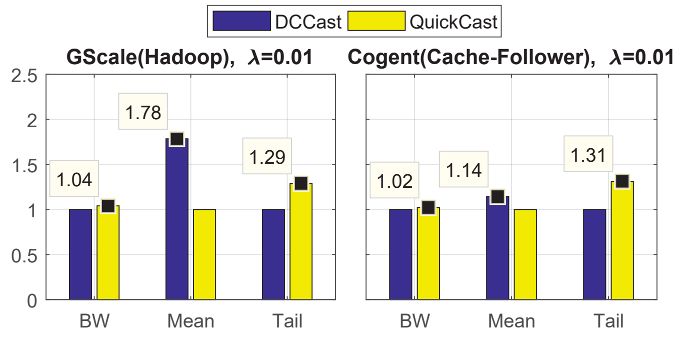
E. Complexity

We discuss two complexity metrics of run-time and number of Group Table entries needed to realize QuickCast.

Computational Complexity: We computed run-time on a machine with a Core-i7 6700 CPU and 24GBs of memory using JRE 8. We used the same transfer size properties mentioned at the beginning of this section. With **GScale(Hadoop)** setting and $\lambda = 0.01$, run-time of procedure **Submit** increased from $1.44ms$ to $2.37ms$ on average while increasing copies from 2 to 10. With the same settings, runtime of procedure **DispatchRates** stayed below $2\mu s$ for varying number of copies. Next, we increased both load and network size by



(a) Heavy Load

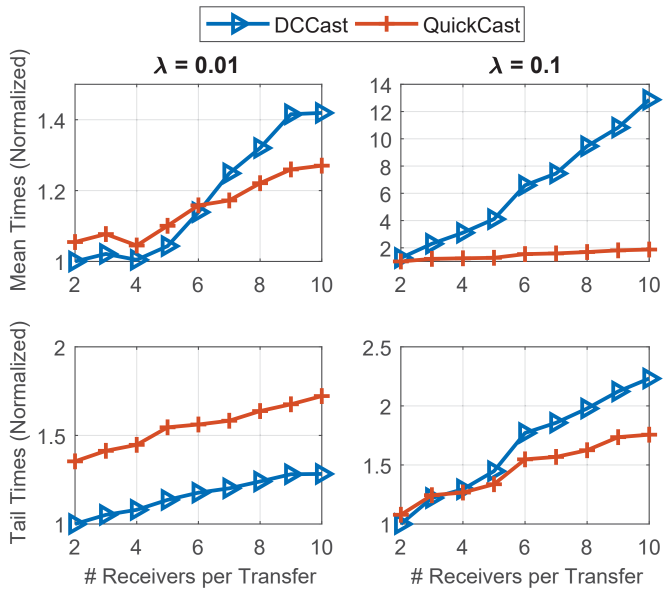


(b) Light Load

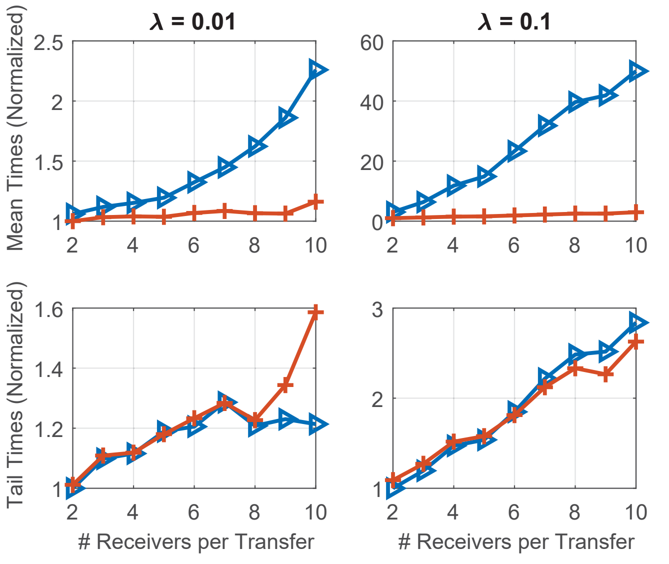
Fig. 8. Comparison of completion times of and bandwidth used by QuickCast vs. DCCast (Normalized by minimum in each category)

switching to **Cogent(Cache-Follower)** setting and $\lambda = 1.0$. This time, run-time of procedure **Submit** increased from $3.5ms$ to $35ms$ and procedure **DispatchRates** increased from $0.75ms$ to $1.6ms$ on average while increasing copies from 2 to 10. Although these run-times are significantly smaller than timeslot widths used in prior works which are in the range of minutes [7], [26], more efficient implementation of proposed techniques may result in even further reduction of run-time. Finally, with our implementation, memory usage of QuickCast algorithm is in the order of 100's of megabytes on average.

Group Table Entries: We performed simulations over 2000 timeslots with $\lambda = 1.0$ (heavily loaded scenario) and number of copies set to 10. With **GScale(Hadoop)** setting, the maximum number of required Group Table entries was 455 and the average of maximum rules for all nodes observed over all timeslots was 166. With **Cogent(Cache-Follower)** setting, which is more than 10 times larger than GScale, we observed a maximum of 165 and an average of 9 Group Table entries for the maximum observed by all nodes over all timeslots. We considered the highly loaded case as it leads to higher number of concurrent forwarding trees. Currently, most switches that support Group Tables offer a maximum of 512 or 1024 entries in total. In this experiment, a maximum of one Group Table entry per transfer per node was enough as we considered switches that support up to 32 action buckets per entry [32] which is more then total number of receivers we chose per transfer. In general, we may need more than one entry per node per transfer or we may have to limit the branching factor of selected forwarding trees, for example when Group Table



(a) Cogent(Cache-Follower)



(b) GScale(Hadoop)

Fig. 9. Comparison of completion times of QuickCast and DCCast (Normalized by minimum in each category) by number of object copies

entries support up to 8 action buckets [33].

V. Discussion

The focus of this paper is on algorithm design and abstract evaluations. In this section, we dive a bit further into practical details and issues that may arise in a real setting.

Timeslot duration: One configuration factor is timeslot length δ. In general, smaller timeslots allow for faster response to changes and arrival of new requests, but add the overhead of rate computations. Minimum possible timeslot length depends on how long it takes for senders to converge to centrally allocated rates.

Handling rate-limiting inaccuracies: Rate-limiting is generally not very accurate, especially if done in software [34]. To deal with inaccuracies and errors, every sender has to report back to the TE server at the end of every timeslot and specify how much traffic it was able to deliver. TE server will deduct these from the residual demand of requests to get the new residual demands. Rate-allocation continues until a request is completely satisfied.

Receiver feedback to sender: Forwarding trees allow flow of data from sender to receivers but receivers also need to communicate with senders. Forwarding rules can be installed so that it supports receivers sending feedback back to sender over the same tree but in reverse direction. There will not be need to use Group Tables on the reverse direction since no replication is performed. One can use a simple point to point scheme for receiver feedback. Since we propose applying forwarding trees over wired networks with rate-limiting, dropped packets due to congestion and corruptions are expected to be low. This means if a scheme such as Negative Acknowledgement (NAK) is used, receiver feedback should be tiny and can be easily handled by leaving small spare capacity over edges.

Handling network capacity loss: Link/switch failures may occur in a real setting. In case of a failure, the TE server can be notified by a network element that detects the failure. The TE server can then exclude the faulty link/switch from topology and re-allocate all requests routed on that link using their residual demands. After new rates are allocated and forwarding trees recomputed, forwarding plane can be updated and new rates can be given to end-points for rate-limiting.

TE server failure: Another failure scenario is when TE server stops working. It is helpful if end-points are equipped with some distributed congestion control mechanism. In case TE server fails, end-points can roll back to the distributed mode and determine their rates according to network feedback.

VI. Related Work

IP multicasting [8], CM [35], TCP-SMO [36] and NORM [37] are instances of approaches where receivers can join groups anytime to receive required data and multicast trees are updated as nodes join or leave. This may lead to trees far from optimal. Also, since load distribution is not taken into account, network capacity may be poorly utilized.

Having knowledge of the topology, centralized management allows for more careful selection of multicast trees and improved utilization via rate-control and bandwidth reservation. CastFlow [38] precalculates multicast trees which can then be used at request arrival time for rule installation. ODPA [39] presents algorithms for dynamic adjustment of multicast spanning trees according to specific metrics. These approaches however do not apply rate-control. MPMC [16], [17] proposes use of multiple multicast trees for faster delivery of files to many receivers then applies coding for improved performance. MPMC does not consider the inter-play between transfers when many P2MP requests are initiated from different source datacenters. In addition, MPMC requires continues changes to the multicast tree which incurs significant control plane overhead as number of chunks and transfers increases. [40] focuses on rate and buffer size calculation for senders. This work does not propose any solution for tree calculations. RAERA [14] is an approximation algorithm to find Steiner trees that minimize data recovery costs for multicasting given a set of recovery nodes. We do not have recovery nodes in our scenario. MTRSA [15] is an approximation algorithm for multicast trees that satisfy a minimum available rate over a general network given available bandwidth over all edges. This work assumes constant rate requirement for transfers and focuses on minimizing bandwidth usage rather than completion times.

For some regular and structured topologies, such as FatTree intra-datacenter networks, it is possible to find optimal (or close to optimal) multicast trees efficiently. Datacast [13] sends data over edge-disjoint Steiner trees found by pruning spanning trees over various topologies of FatTree, BCube and Torus. AvRA [12] focuses on Tree and FatTree topologies and builds minimal edge Steiner trees that connect the sender to all receivers as they join. MCTCP [41] reactively schedules flows according to link utilization.

As an alternative to in-network multicasting, one can use overlay networks where hosts perform forwarding. RDCM [42] populates backup overlay networks as nodes join and transmits lost packets in a peer-to-peer fashion over them. NICE [9] creates hierarchical clusters of multicast peers and aims to minimize control traffic overhead. AMMO [43] allows applications to specify performance constraints for selection of multi-metric overlay trees. DC2 [44] is a hierarchy-aware group communication technique to minimize cross-hierarchy communication. SplitStream [45] builds forests of multicast trees to distribute load across many machines. Due to lack of complete knowledge of underlying network topology and status (e.g. link utilizations, congestion or even failures), overlay systems are limited in reducing bandwidth usage and managing distribution of traffic.

Alternatives to multicasting for bulk data distribution include peer-to-peer [10], [11], [46] and store-and-forward [47]–[50] approaches. Peer-to-peer approaches do not consider careful link level traffic allocation and scheduling and do not focus on minimizing bandwidth usage. The main focus of peer-to-peer schemes is to locally and greedily optimize completion times rather than global optimization over many transfers across a network. Store-and-forward approaches focus on minimizing costs by utilizing diurnal traffic patterns while delivering bulk objects and incur additional bandwidth and storage costs on intermediate datacenters. Coflows are another related concept where flows with a collective objective are jointly scheduled for improved performance [51]. Coflows however do not aim at bandwidth savings.

There are recent solutions for management of P2MP transfers with deadlines. DDCCast [52] uses a single forwarding tree and the As Late As Possible (ALAP) scheduling policy for admission control of multicast deadline traffic. In [53], authors propose use of few parallel forwarding trees from source to all receivers to increase throughput and meet more deadlines considering transfer priorities and volumes. The techniques we proposed in QuickCast for receiver set partitioning can be applied to these prior work for further performance gains.

In design of QuickCast, we did not focus on throughput-optimality, which guarantees network stability for any offered load in capacity region. We believe our tree selection approach, which balances load across many existing forwarding trees, aids in moving towards throughput-optimality. One could consider developing a P2MP joint scheduling and routing scheme with throughput-optimality as the objective. However, in general, throughput-optimality does not necessarily lead to highest performance (e.g., lowest latency) [54], [55].

Reliability: Various techniques have been proposed to make multicasting reliable including use of coding and receiver (negative or positive) acknowledgements or a combination of them [56]. In-network caching has also been used to reduce recovery delay, network bandwidth usage and to address the ACK/NAK implosion problem [13], [57]. Using positive ACKs does not lead to ACK implosion for medium scale (sub-thousand) receiver groups [36]. TCP-XM [58] allows reliable delivery by using a combination of IP multicast and unicast for data delivery and re-transmissions. MCTCP [41] applies standard TCP mechanisms for reliability. Receivers may also send NAKs upon expiration of some inactivity timer [37]. NAK suppression to address implosion can be done by routers [57]. Forward Error Correction (FEC) can be used for reliable delivery, using which a sender encodes k pieces of an object into n pieces ($k \leq n$) where any k out of n pieces allow for recovery [59], [60]. FEC has been used to reduce re-transmissions [37] and improve the completion times [61]. Some popular FEC codes are Raptor Codes [62] and Tornado Codes [63].

Congestion Control: PGMCC [64], MCTCP [41] and TCP-SMO [36] use window-based TCP like congestion control to compete fairly with other flows. NORM [37] uses an equation-based rate control scheme. Datacast [13] determines the rate according to duplicate interest requests for data packets. All of these approaches track the slowest receiver.

VII. Conclusions and Future Work

In this paper, we presented QuickCast algorithm to reduce completion times of P2MP transfers across datacenters. We showed that by breaking receiver sets of P2MP transfers with many receivers into smaller subsets and using a separate tree per subset, we can reduce completion times. We proposed partitioning according to proximity as an effective approach for finding such receiver subsets, and showed that partitioning need be applied to transfers selectively. To do so, we proposed a partitioning factor that can be tuned according to topology and traffic distribution. Further investigation is required on finding metrics to selectively apply partitioning per transfer. Also, investigation of partitioning techniques that optimize network performance metrics as well as study of optimality bounds of such techniques are left as part of future work.

Next, we discovered that while managing P2MP transfers with many receivers, FCFS policy used in DCCast creates many contentions as a result of overlapping forwarding trees significantly reducing utilization. We applied Fair Sharing policy reducing network contention and improving completion times. Finally, we performed experiments with well-known rate-allocation policies and realized that Max-Min Fairness provides much lower completion times compared to SRPT while scheduling over large forwarding trees. More research is needed on best rate-allocation policy for P2MP transfers. Alternatively, one may drive a more effective joint partitioning, rate-allocation and forwarding tree selection algorithm by approximating a solution to optimization model we proposed.

References

[1] "Predicting sd-wan adoption," http://bit.ly/gartner-sdwan, visited on July 21, 2017.

[2] "Microsoft azure: Cloud computing platform & services," https://azure.microsoft.com/.

[3] "Compute engine - iaas - google cloud platform," https://cloud.google.com/compute/.

[4] C.-Y. Hong, S. Kandula, R. Mahajan *et al.*, "Achieving high utilization with software-driven wan," in *SIGCOMM*. ACM, 2013, pp. 15–26.

[5] S. Jain, A. Kumar *et al.*, "B4: Experience with a globally-deployed software defined wan," *SIGCOMM*, vol. 43, no. 4, pp. 3–14, 2013.

[6] S. Kandula, I. Menache, R. Schwartz, and S. R. Babbula, "Calendaring for wide area networks," *SIGCOMM*, vol. 44, no. 4, pp. 515–526, 2015.

[7] X. Jin, Y. Li, D. Wei, S. Li, J. Gao, L. Xu, G. Li, W. Xu, and J. Rexford, "Optimizing bulk transfers with software-defined optical wan," in *SIGCOMM*. ACM, 2016, pp. 87–100.

[8] S. Deering, "Host Extensions for IP Multicasting," Internet Requests for Comments, pp. 1–16, 1989. [Online]. Available: https://tools.ietf.org/html/rfc1112

[9] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast," in *SIGCOMM*. ACM, 2002, pp. 205–217.

[10] R. Sherwood, R. Braud, and B. Bhattacharjee, "Slurpie: a cooperative bulk data transfer protocol," in *INFOCOM*, vol. 2, 2004, pp. 941–951.

[11] J. Pouwelse, P. Garbacki, D. Epema *et al.*, *The Bittorrent P2P File-Sharing System: Measurements and Analysis*. Springer Berlin Heidelberg, 2005.

[12] A. Iyer, P. Kumar, and V. Mann, "Avalanche: Data center multicast using software defined networking," in *COMSNETS*. IEEE, 2014, pp. 1–8.

[13] J. Cao, C. Guo, G. Lu, Y. Xiong, Y. Zheng, Y. Zhang, Y. Zhu, C. Chen, and Y. Tian, "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, 2013.

[14] S. H. Shen, L. H. Huang, D. N. Yang, and W. T. Chen, "Reliable multicast routing for software-defined networks," in *INFOCOM*, April 2015, pp. 181–189.

[15] L. H. Huang, H. C. Hsu, S. H. Shen, D. N. Yang, and W. T. Chen, "Multicast traffic engineering for software-defined networks," in *INFOCOM*. IEEE, 2016, pp. 1–9.

[16] A. Nagata, Y. Tsukiji, and M. Tsuru, "Delivering a file by multipath-multicast on openflow networks," in *International Conference on Intelligent Networking and Collaborative Systems*, 2013, pp. 835–840.

[17] K. Ogawa, T. Iwamoto, and M. Tsuru, "One-to-many file transfers using multipath-multicast with coding at source," in *IEEE International Conference on High Performance Computing and Communications*, 2016, pp. 687–694.

[18] M. Noormohammadpour, C. S. Raghavendra, S. Rao, and S. Kandula, "Dccast: Efficient point to multipoint transfers across datacenters," in *HotCloud*. USENIX Association, 2017.

[19] N. McKeown, T. Anderson *et al.*, "Openflow: Enabling innovation in campus networks," *SIGCOMM*, vol. 38, no. 2, pp. 69–74, 2008.

[20] B. Pfaff, B. Lantz, B. Heller *et al.*, "Openflow switch specification, version 1.1.0 implemented (wire protocol 0x02)," http://archive.openflow.org/documents/openflow-spec-v1.1.0.pdf, 2011.

[21] "Omniswitch aos release 8 switch management guide," http://bit.ly/sdn-lucent, visited on July 21, 2017.

[22] "Openflow v1.3.1 compliance matrix for devices running junos os," http://bit.ly/sdn-juniper, visited on July 21, 2017.

[23] "Hp openflow 1.3 administrator guide," http://bit.ly/sdn-hp, visited on July 21, 2017.

[24] "Network os software defined networking (sdn) configuration guide," http://bit.ly/sdn-brocade, visited on July 21, 2017.

[25] D. Bertsekas and R. Gallager, "Data networks," 1987.

[26] H. Zhang, K. Chen, W. Bai *et al.*, "Guaranteeing deadlines for inter-datacenter transfers," in *EuroSys*. ACM, 2015, p. 20.

[27] M. Noormohammadpour, C. S. Raghavendra, and S. Rao, "Dcroute: Speeding up inter-datacenter traffic allocation while guaranteeing deadlines," in *High Performance Computing, Data, and Analytics (HiPC)*. IEEE, 2016.

[28] H. H. Liu, S. Kandula, R. Mahajan, M. Zhang, and D. Gelernter, "Traffic engineering with forward fault correction," in *SIGCOMM*. ACM, 2014, pp. 527–538.

[29] M. Stanojevic and M. Vujoevic, "An exact algorithm for steiner tree problem on graphs," *International Journal of Computers Communications & Control*, vol. 1, no. 1, pp. 41–46, 2006.

[30] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *SIGCOMM*. ACM, 2015, pp. 123–137.

[31] "The internet topology zoo (cogent)," http://www.topology-zoo.org/files/Cogentco.gml, visited on July 19, 2017.

[32] "Understanding how the openflow group action works," https://www.juniper.net/documentation/en_US/junos/topics/concept/junos-sdn-openflow-groups.html, visited on March 14, 2017.

[33] "Hp 5130 ei switch series openflow configuration guide," https://support.hpe.com/hpsc/doc/public/display?docId=c04217797&lang=en-us&cc=us, visited on Dec 8, 2017.

[34] M. Noormohammadpour and C. S. Raghavendra, "Datacenter Traffic Control: Understanding Techniques and Trade-offs," *arXiv preprint arXiv:1712.03530*, 2017. [Online]. Available: https://arxiv.org/abs/1712.03530

[35] S. Keshav and S. Paul, "Centralized multicast," in *International Conference on Network Protocols*. IEEE, 1999, pp. 59–68.

[36] S. Liang and D. Cheriton, "Tcp-smo: extending tcp to support medium-scale multicast applications," in *Proceedings.Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 3, 2002, pp. 1356–1365.

[37] B. Adamson, C. Bormann, M. Handley, and J. Macker, "Nack-oriented reliable multicast (norm) transport protocol," 2009.

[38] C. A. C. Marcondes, T. P. C. Santos, A. P. Godoy, C. C. Viel, and C. A. C. Teixeira, "Castflow: Clean-slate multicast approach using in-advance path processing in programmable networks," in *IEEE Symposium on Computers and Communications*, 2012, pp. 94–101.

[39] J. Ge, H. Shen, E. Yuepeng, Y. Wu, and J. You, "An openflow-based dynamic path adjustment algorithm for multicast spanning trees," in *IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, 2013, pp. 1478–1483.

[40] X. Ji, Y. Liang, M. Veeraraghavan, and S. Emmerson, "File-stream distribution application on software-defined networks (sdn)," in *IEEE Annual Computer Software and Applications Conference*, vol. 2, July 2015, pp. 377–386.

[41] T. Zhu, F. Wang, Y. Hua, D. Feng *et al.*, "Mctcp: Congestion-aware and robust multicast tcp in software-defined networks," in *International Symposium on Quality of Service*, June 2016, pp. 1–10.

[42] D. Li, M. Xu, M. c. Zhao, C. Guo, Y. Zhang, and M. y. Wu, "Rdcm: Reliable data center multicast," in *2011 Proceedings IEEE INFOCOM*, 2011, pp. 56–60.

[43] A. Rodriguez, D. Kostic, and A. Vahdat, "Scalability in adaptive multi-metric overlays," in *International Conference on Distributed Computing Systems*, 2004, pp. 112–121.

[44] K. Nagaraj, H. Khandelwal, C. Killian, and R. R. Kompella, "Hierarchy-aware distributed overlays in data centers using dc2," in *COMSNETS*. IEEE, 2012, pp. 1–10.

[45] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "Splitstream: High-bandwidth multicast in cooperative environments," in *SOSP*. ACM, 2003, pp. 298–313.

[46] M. Hefeeda, A. Habib, B. Botev *et al.*, "Promise: Peer-to-peer media streaming using collectcast," in *MULTIMEDIA*. ACM, 2003, pp. 45–54.

[47] N. Laoutaris, M. Sirivianos, X. Yang, and P. Rodriguez, "Inter-datacenter bulk transfers with netstitcher," in *SIGCOMM*. ACM, 2011, pp. 74–85.

[48] S. Su, Y. Wang, S. Jiang, K. Shuang, and P. Xu, "Efficient algorithms for scheduling multiple bulk data transfers in inter-datacenter networks," *International Journal of Communication Systems*, vol. 27, no. 12, 2014.

[49] N. Laoutaris, G. Smaragdakis, R. Stanojevic, P. Rodriguez, and R. Sundaram, "Delay-tolerant bulk data transfers on the internet," *IEEE/ACM TON*, vol. 21, no. 6, 2013.

[50] Y. Wang, S. Su *et al.*, "Multiple bulk data transfers scheduling among datacenters," *Computer Networks*, vol. 68, pp. 123–137, 2014.

[51] M. Chowdhury and I. Stoica, "Coflow: A networking abstraction for cluster applications," in *Workshop on Hot Topics in Networks*. ACM, 2012, pp. 31–36.

[52] M. Noormohammadpour and C. S. Raghavendra, "DDCCast: Meeting Point to Multipoint Transfer Deadlines Across Datacenters using ALAP Scheduling Policy," Department of Computer Science, University of Southern California, Tech. Rep. 17-972, 2017.

[53] S. Ji, "Resource optimization across geographically distributed datacenters," Master's thesis, University of Toronto, 2017.

[54] A. Yekkehkhany, "Near Data Scheduling for Data Centers with Multi Levels of Data Locality," *arXiv preprint arXiv:1702.07802*, 2017.

[55] A. Yekkehkhany, A. Hojjati, and M. H. Hajiesmaili, "Gb-pandas: Throughput and heavy-traffic optimality analysis for affinity scheduling," *arXiv preprint arXiv:1709.08115*, 2017.

[56] M. Handley, L. Vicisano, M. Luby, B. Whetten, and R. Kermode, "The reliable multicast design space for bulk data transfer," Internet Requests for Comments, pp. 1–22, 2000. [Online]. Available: https://tools.ietf.org/html/rfc2887

[57] L. H. Lehman, S. J. Garland, and D. L. Tennenhouse, "Active reliable multicast," in *INFOCOM*, vol. 2, Mar 1998, pp. 581–589 vol.2.

[58] K. Jeacle and J. Crowcroft, "Tcp-xm: unicast-enabled reliable multicast," in *ICCCN*, 2005, pp. 145–150.

[59] M. Luby, L. Vicisano, J. Gemmell *et al.*, "The use of forward error correction (fec) in reliable multicast," 2002.

[60] M. Luby, J. Gemmell, L. Vicisano, L. Rizzo, and J. Crowcroft, "Asynchronous layered coding (alc) protocol instantiation," 2002.

[61] C. Gkantsidis, J. Miller, and P. Rodriguez, "Comprehensive view of a live network coding p2p system," in *IMC*. ACM, 2006, pp. 177–188.

[62] A. Shokrollahi, "Raptor codes," *IEEE Transactions on Information Theory*, vol. 52, no. 6, pp. 2551–2567, 2006.

[63] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege, "A digital fountain approach to reliable distribution of bulk data," in *SIGCOMM*. ACM, 1998, pp. 56–67.

[64] L. Rizzo, "Pgmcc: A tcp-friendly single-rate multicast congestion control scheme," in *SIGCOMM*, 2000.