Deadline-Aware Multicast Transfers in Software-Defined Optical Wide-Area Networks

Long Luo, *Member, IEEE,* Klaus-Tycho Foerster,
Stefan Schmid, *Member, IEEE,* Hongfang Yu, *Member, IEEE.*

***Abstract*—The increasing amount of data replication across datacenters introduces a need for efficient bulk data transfer protocols which provide certain guarantees, most notably timely transfer completion. We present *DaRTree* which leverages emerging optical reconfiguration technologies, to jointly optimize topology and multicast transfers in software-defined optical Wide-Area Networks (WANs), and thereby maximize throughput and acceptance ratio of transfer requests subject to transfer deadlines. *DaRTree* is based on a novel integer linear program relaxation and deterministic rounding scheme. To this end, *DaRTree* uses Steiner trees for forwarding and adaptive routing based on the current network load. *DaRTree* provides transfer completion guarantees without the need for rescheduling or preemption. Our evaluations show that *DaRTree* increases the network throughput and the number of accepted requests by up to $1.7\times$, especially for larger WANs. Moreover, *DaRTree* even outperforms state-of-the-art solutions when the traffic demands are only unicast transfers or when the WAN topology cannot be reconfigured. While *DaRTree* determines the rate and route to serve a request at the time of (online) admission control, we show that the acceptance ratio and throughput can be improved by up to $1.3\times$ even further when *DaRTree* updates the rate and route of admitted transfers also at runtime.**

I. Introduction

With the increasing popularity of online services on many fronts (health, business, streaming, or social networking), datacenters will continue to grow explosively in the coming years, both in size and numbers [2]. Datacenters hence become a critical infrastructure of our digital society. This also introduces increasingly stringent availability and dependability requirements, which in turn require large-scale data replication across multiple datacenters. Such replication can result in bulk transfers ranging from terabytes to petabytes [3]–[7].

These bulk transfers of replication applications are often *one-to-many*. For example, for availability, many cloud services typically require data or content (e.g., search indices, video files, and backups) to be dynamically copied from the datacenter hosting the data to many destination datacenters that rely on such replica to run services. Indeed, one-to-many transfers can dominate the inter-datacenter traffic of large-scale service companies [8]. Another key characteristic of such one-to-many transfers is that they require to be completed timely. A majority of such transfers have a hard deadline for completion time, while the transfers are not very sensitive to delay and rate [7, 9]. Other transfers require quick synchronization, e.g., of an index of a search service, to provide users with high search quality [9]–[11]. A recent survey of Wide-Area Network (WAN) customers at Microsoft showed that 88% of them incur penalties on missed deadlines [9]. Network operators hence need to carefully manage these large one-to-many transfers to make sure they meet their deadlines.

This paper is motivated by two technological opportunities to improve the efficiency of one-to-many transfers. The first opportunity is related to emerging innovative traffic engineering mechanisms, as enabled by Software-Defined Networks (SDNs), which allow to improve large data transfers. An SDN does not only support more flexible changes of routes and rates, which can be exploited to admit more traffic while ensuring that deadlines are met [3]–[5, 7, 10], but it also allows to go beyond today's *unicast* approach (e.g., *Amoeba* [7]) to one-to-many bulk data transfers: SDNs (e.g., using OpenFlow group tables [12] or P4 [13]) support efficient communication primitives such as *anycast* [14] or *multicast* [15]–[19]. Communication along a multicast trees is particularly interesting for bulk-data transfers as it can save bandwidth by avoiding redundant transmissions.

The second opportunity is related to emerging optical technologies, which allow to optimize also the *physical layer*, through reconfigurations at runtime [20]–[23]: recent optical WAN technology allows to update the network topology by flexibly and rapidly *shifting* wavelengths to neighboring fibers. Wavelength assignments, the vehicle to send data across fibers, hence become *reconfigurable*. In turn, this enables *demand-aware* network topologies, which adjust the network's capacity to current traffic demands [20, 21]. Notwithstanding, SDN is the practical enabler of these dynamic technologies [20].

However, today we do not have a good understanding of how to exploit such technologies toward efficient multicast transfers. While recent work highlights the potential of reconfigurations, these solutions are still limited to unicast transfers [20, 21], and hence are not well suited for multicast transfers.

Contributions. In this paper, we initiate the study of how to jointly optimize bulk multicast transfers subject to strict deadlines, leveraging both SDN-enabled forwarding trees and

Long Luo and Hongfang Yu are with the University of Electronic Science and Technology, P.R. China (e-mail: longluo.uestc@gmail.com, yuhf@uestc.edu.cn). Klaus-Tycho Foerster and Stefan Schmid are with Faculty of Computer Science, University of Vienna, Austria (e-mail: klaus-tycho.foerster@univie.ac.at, stefan_schmid@univie.ac.at). Work performed while visiting at University of Vienna. A preliminary version of this article appears in the proceedings of the 27th IEEE/ACM International Symposium on Quality of Service [1].

This work was partially supported by the National Key Research and Development Program of China(2019YFB1802803); the PCL Future Greater-Bay Area Network Facilities for Large-scale Experiments and Applications (PCL2018KP001). This project has also received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement No. 864228, AdjustNet: Self-Adjusting Networks).

reconfigurable topologies in our *DaRTree*[1] approach. *DaRTree* is based on a deterministic MILP rounding scheme and comes with several attractive properties. In particular, we show that while *DaRTree* combines multicast transfer and topology reconfiguration optimizations, *DaRTree* outperforms state-of-the-art of approaches already for *just one* of these optimizations:

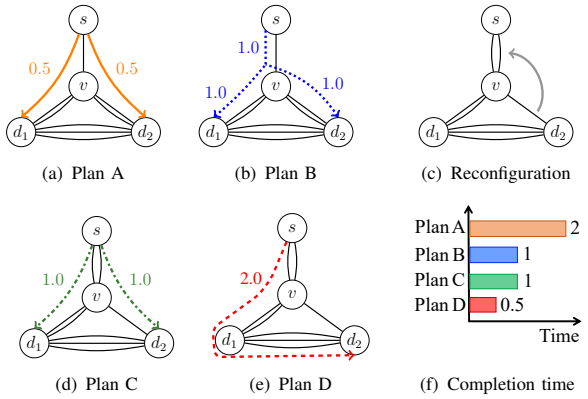
- Even under a workload which consists only of unicast transfers, *DaRTree* outperforms prior work such as *Owan* [20] (which is based on local search heuristics to reconfigure the WAN), by efficiently relaxing and rounding an integer program formulation.
- Even if the WAN topology is static, i.e., wavelengths cannot be reconfigured, *DaRTree* outperforms prior multicast approaches like *MTree* [17] as well. *DaRTree* generates multicast Steiner trees with the current network load in mind, i.e., performs adaptive routing.
- Our extensive simulations on real-world topologies show that the joint optimization of *DaRTree* greatly improves on the state of the art. We can increase the network throughput and the number of accepted requests by up to 1.7×, in particular for larger real-world topologies.
- Moreover, *DaRTree* can also adapt to different transfer request utility functions, maintaining efficient computation times and improving the weighted acceptance ratio.
- *DaRTree* does not rely on rescheduling or preemption, and always guarantees deadlines, by being reservation-based. In case rates and routes may be adapted over time, we utilize *DaRTreeJoint*, which optimizes the network and transfers in every timeslot, while maintaining deadline completion for all admitted transfers. Our simulations show that *DaRTreeJoint* improves the acceptance ratio and throughput by another factor of up to 1.3×, depending on the scenario.

Example. Consider the four node WAN in Fig. 1, where each node can use up to five unit-capacity wavelengths in total to connect to its neighbors[2] over fiber. Initially, we have one wavelength (black edge) connecting s and v, two wavelengths between v and d_1, d_2, respectively, and three between d_1, d_2. The objective is to improve the network throughput and accept more requests with tight deadlines.

Assume that there is a data transfer request from s to two receivers d_1, d_2. As there is a bottleneck between s and v, a unicast transfer as in Plan A in Fig. 1(a), using e.g. *Amoeba* [7], takes twice as long as a multicast transfer as in Plan B in Fig. 1(b), using e.g. *MTree* [17]. Both methods can be sped up by reconfiguring the wavelengths across this WAN, as shown in Fig. 1(c). Now, unicast transfers finish in half the time using Plan C (see Fig. 1(d)) that may be found by *Owan* proposed in [20]. Our approach *DaRTree*, combining both multicasting and reconfiguration, completes in 0.5 time units as shown in Plan D in Fig. 1(e). As seen in Fig. 1(f), only *DaRTree* can accept the request if its deadline is 0.5 time units, and all other approaches have to reject it or miss its deadline, they only meet deadlines of 1 to 2 time units.

[1] *DaRTree* stands for **D**eadline-**a**ware **R**econfigurable **Tree**s.

[2] E.g., the node v can connect to all other nodes, but s only to v.



(a) Plan A (b) Plan B (c) Reconfiguration

(d) Plan C (e) Plan D (f) Completion time

Fig. 1. Example for the power of multicast transfers and topology reconfiguration. Initially, the wavelengths (black edges) are configured as shown in Fig. 1(a), where each wavelength connecting two nodes can send 1 unit of data per time unit. When the node s wants to replicate a volume of 1 data units to both d_1 and d_2, the transmission speed is limited to 1 unit at node v. As such, 2 seconds are needed according to Plan A using unicast transfers (Fig. 1(a)) and 1 second with Plan B using multicast transfers along Steiner trees (Fig. 1(b)). However, when the wavelengths are reconfigured as in Fig. 1(c), the transfer times are halved: 1 time unit with Plan C using unicast transfers (Fig. 1(d)) and just 0.5 time units according to Plan D with multicast transfers (Fig. 1(e).

Organization. We first review related work in §II, then provide some background on reconfigurable WANs and introduce our model in §III, followed by an overview of our approach in §IV, and an offline problem formalization in §V. We present *DaRTree* in details in §VI and then cover *DaRTreeJoint* in §VII, which may adapt transfer rates and routing. After reporting on simulation results in §VIII, we conclude in §IX.

II. Related Work

Following the emergence of software-defined networking technologies, many problems in the networking field have been rethought and reoptimized. Traffic engineering for inter-datacenter WANs, as a classic problem, has received increasing attention in networking research as the number of datacenters and the inter-datacenter traffic demands are growing at an unprecedented rate. In particular, many works have investigated how to improve the traffic engineering for wide-area networks under an SDN architecture [3]–[7, 10, 16, 17, 20, 21, 24]. Earlier work focused on network-wide objectives such as minimizing network utilization and maximizing network throughput [3, 4]. Recent work considers more fine-grained objectives, like meeting deadlines of bulk data transfers [5, 7, 10, 16, 17, 20, 24] and minimizing the completion time of data transfers [6, 21]. In this context, most work focuses on unicast transfers, with some more recent work also considering multicast transfers.

Unicast Transfers: Unicast transfers in inter-datacenter network have been the focus of much attention [5, 7, 10, 20, 21]. These works adopt k-shortest paths to deliver traffic and control the transmission rate along these paths to optimize the unicast transfers. Tempus [5] aims to allocate transfers fairly by delivering the maximum same deadline-met data fraction for all transfers. Amoeba [7] performs online admission control

and focuses on guaranteeing the deadlines for a maximum number of transfers. Luo *et at.* [10] propose a competitive online algorithm to maximize the system utility of delivering transfers with either hard or soft deadlines. All the above works do not take multicast transfers into consideration.

Multicast Transfers: With the exponential growth of geo-replication, there has also been a spike in interest to design algorithms explicitly for multicast data transfers in inter-datacenter WANs [6, 16, 17, 24]. DDCCast [16] proposes to satisfy the transfer deadlines by delivering data over a forwarding tree. QuickCast [6] considers multiple forwarding trees and focuses on reducing mean completion times of elastic multicast transfers by taking into account forwarding tree selection and rate-allocation. Ji *et al.* [17] focus on providing deadline promises to as many transfers as possible by controlling the transfer transmission rate over non-adaptive routing trees.

Luo *et al.* [24] aim to maximize the number of deadline-satisfied transfers by allowing the receivers that have already completed to send replica to other uncompleted receivers. There also exists a group of works [8, 25]–[29] that adopt a store-and-forward mechanism to optimize inter-datacenter bulk traffic by using additional storage capacities of servers in intermediate datacenters, which is not investigated by this work and many related works. All above proposals consider transfers over unreconfigurable networks.

Reconfigurable Networks: The power of dynamic inter-datacenter WAN reconfiguration was recently showcased by *Owan* [20, 21]. In order to satisfy deadlines for unicast transfers respectively reduce their completion time, *Owan* jointly reconfigures the network topology by a local search heuristic and controls the transmission rate along k-paths. Our approach on the other hand utilizes multicast routing along k-Steiner trees and leverages an efficiently relaxed optimization program to assign wavelengths. The also exists work on bandwidth-variable links [23, 30] and abstractions in reconfigurable WANs [31], and on multicast [32, 33] in reconfigurable datacenters [34], which are however all orthogonal to our setting.

III. Background and Preliminaries

We first give a technological background on reconfigurable WANs, which we integrate into our formal model, closely following the assumptions of prior work in this area [20, 21].

Background on Reconfigurable WANs. We focus on multicast bulk transfers in WANs connecting multiple datacenter networks (DCNs), empowered by SDN to centrally control the networking devices. However, it can also directly be applied by ISPs that offer bulk transfer services to clients [20].

A reconfigurable WAN consists of Reconfigurable Optical Add/Drop Multiplexers (ROADMs), which in turn are connected by optical fiber cables. The optical fibers are used to transmit wavelengths, whose number and capacity depends on the technology. For example, using Wavelength Division Multiplexing (WDM) and On-Off Keying (OOK), 40 wavelengths at 10 Gbps can be supported simultaneously [35]. Newer technologies can support e.g. 88 or more wavelengths using dense WDM, at higher data rates of 40/100 Gbps [35, 36]. While these WANs are manually configured by default (e.g., for initial setup), ROADMs also allow to dynamically reconfigure the wavelength allocations on the fly in the order of hundreds of milliseconds [20]. The number of deployed wavelengths per ROADM is limited by its number of transponders, where the receiving and sending parts are commonly bundled into bidirectional wavelengths, but may also be separated [37]. Previous work highlighted the potential of reconfigurable WANs, but so far focused on (single-hop [21, 38]) unicast transfers in inter-datacenter networks [20]. We go beyond these works by incorporating multicast transfers in multi-hop networks and providing an efficient algorithmic framework based on integer program relaxation and rounding.

Preliminaries. We model a reconfigurable WAN by an undirected graph $G = (V, E)$, where the nodes V represent ROADMs connected to DCNs and the edges E are the fibers connecting them. Each fiber $e \in E$ has a maximum number of wavelengths $C_e \in \mathbb{N}$ it can carry and each node $v \in V$ can send $C_v^s \in \mathbb{N}$ and receive $C_v^r \in \mathbb{N}$ wavelengths via its transponders in total, respectively. In order to model the proper wavelength assignment via transponders to the fibers, we introduce two directed (virtual) links L for each fiber $e \in E$, in opposing directions: a link $l \in L$ from u to v on e can be assigned at most $\min\{C_u^s, C_v^r, C_e\}$ wavelengths. We focus on an online system, where transfer requests arrive the network dynamically. Each transfer request R is specified by a source $s \in V$, a set of receivers $\boldsymbol{d} \subseteq V$, the volume (size) f of to-be-transferred data, the time t^{arr} to start and the deadline t^{dl} of completion time.

IV. Overview of *DaRTree*

Abstractly, *DaRTree* is an online scheduler for bulk multicast transfers in reconfigurable SDN-based wide-area networks (e.g., based on OpenFlow [12] or P4 [13]). It orchestrates the topology, routing and transmission rate for requested data transfers without prior knowledge of the future requests.

The controller maintains a global view of the network topology and all ongoing transfer requests. It operates in a discrete slotted time system, where each timeslot has a size of several minutes (e.g., 5 minutes). Transfer requests appear at the beginning of every timeslot in an online fashion. When new transfer requests arrive from clients, *DaRTree* performs admission control in order to determine which of them can be accepted, given the transfer deadline and network capacity constraints. Once a request is admitted, *DaRTree* guarantees the completion of a transfer request before its deadline, in order to avoid utility loss and further penalties. To this end, *DaRTree* utilizes efficient algorithms to orchestrate deadline-aware wavelength assignment, routing, and rate allocation for accepted transfers. Before each timeslot, the controller reconfigures the network-layer topology by enforcing ROADMs to change their wavelength allocation, configure routing trees by updating forwarding rules in switches (e.g., using network update mechanisms [39]), and informs clients of the sending rates of their data transfers.

As noted in this context by Jin et al., "*A time slot is much longer than the time to reconfigure the network and adjust sending rates, i.e., a few minutes vs. hundreds or thousands of milliseconds.*" [20] Moreover, as the volume of transfers in the context of inter-datacenters is often in the order of terabytes to petabytes [3]–[7], transfers usually last from minutes to hours. The reconfiguration delay imposed by *DaRTree* is hence negligible, analogously to the propagation delay.

Lastly, in order to not disturb the small fraction of interactive traffic, which is sensible to delays yet stable and predicable over short time periods [7], *DaRTree* can reserve a corresponding set of wavelengths and only reallocates the remaining wavelengths for optimizing large data transfers.

Algorithm overview. We focus on multicast transfers that have strict deadlines on their completion times. In order to maximize the total system utility, we aim to admit the maximum number of deadline-meeting transfers by jointly optimizing the network topology together with the routing and rate allocation dynamically. We provide two versions of algorithms in *DaRTree*, depending on the requirements of the clients. In the first reservation-based version, *DaRTree* fixes the transfer rate and routes for each request in their first timeslot, i.e., the clients can already plan ahead for the whole lifetime of the transfer, as the resources are reserved. In the second version, coined *DaRTreeJoint* the deadline completion of each admitted transfer is guaranteed as well, but the rates and routes might change in each timeslot. As such, we can admit more requests, but the clients need to be more flexible. More precisely, *DaRTree* relies on the following ideas:

1) When a new batch of requests arrives, we compute a set of k Steiner trees for the routing for each transfer. This computation is separated from the wavelength and rate allocation part, to speed up the computation time of *DaRTree*. However, *DaRTree* is not oblivious to the network utilization in this step: the routing trees are created in a load-adaptive manner.
2) Next, we formulate the wavelength and rate allocation problem for transfers as a mixed integer linear program. In order to relax MILP constraints, we set a small amount of wavelengths aside, to obtain feasible solutions. These spare resources are optimized according to the chosen Steiner trees.
3) We then maximize the number of requests admitted in the current timeslot. In order to provision for future requests, we spread the resource usage over a longer time, instead of greedily filling the network for the next few timeslots.
4) Lastly, we admit the maximum number of requests possible for this timeslot and obtain a feasible wavelength allocation with the spare resources. We would like to emphasize that all allocation details made by current timeslot for these accepted transfer requests stay fixed: they may not be modified or preempted in future timeslots, and are in particular not impacted by topology reconfiguration.

Our second version, *DaRTreeJoint*, differs from *DaRTree* in the sense that the allocations are not fixed for the whole lifetime of the transfer request. Rather, in each timeslot, we recompute the routing and wavelength allocation, under the constraint that each admitted transfer can still meet its deadline. Herein we can leverage known network update techniques [20, 40] to perform consistent cross-layer updates during topology reconfiguration, e.g., via optimization formulations or dependency graphs.

TABLE I. Key notations used in the problem formulation

Network model	
V	the set of all datacenters (i.e., the nodes)
E	the set of all inter-datacenter fibers (i.e., the edges)
L	the set of all inter-datacenter directed link connections
$C^s_{v,t}$	the maximum number of wavelengths that node $v \in V$ can send at time t via fibers connecting to it
$C^r_{v,t}$	the maximum number of wavelengths that node $v \in V$ can receive at time t via fibers connecting to it
$C_{e,t}$	the maximum number of wavelengths that edge $e \in E$ can carry at time t
c	the capacity carried with per wavelength
α	the length of a timeslot
$\mathcal{R}^{\text{all}}$	the collection of all transfer requests from a global time view
$\mathcal{R}^{\text{cur}}$	the collection of newly incoming transfer requests at the beginning of timeslots t
$\mathcal{R}'$	the collection of all accepted but unfinished transfer requests at the end of timeslots t
Transfer request R	
s	the source datacenter
$\boldsymbol{d}$	the set of receivers: $\boldsymbol{d} \subseteq V \setminus \{s\}$
f	the volume of to-be-transferred data
t^{arr}	the arrival time of request R
t^{dl}	the deadline required to complete the data transfer R
$\mathcal{K}$	a set of k forwarding trees: $\mathcal{K} = \{\kappa_1, \cdots, \kappa_k\}$, each connecting the source s to all the receivers in $\boldsymbol{d}$
Internal and decision variables	
$g_{l,t}$	the number of wavelength assigned to link l in time t
$x_{R,\kappa}$	the transmission rate on forwarding tree κ of request R
$x_{R,\kappa,t}$	the transmission rate on forwarding tree κ of request R at time t
z_R	binary, whether request R can be completed before deadline
η	total weighted assigned number of wavelengths across links and time slots
ε	total number of deadline-satisfied data transfers from $\mathcal{R}^{\text{cur}}$

V. Offline Problem Formulation for *DaRTree*

Although we focus on the online multicast transfer problem in this work, we first introduce its offline version in order to 1) introduce key notation and 2) provide a mixed integer linear programming (MILP) formulation which we adapt in the later sections to efficient online algorithms. Note that in the offline case, all submitted transfer requests $\mathcal{R}^{\text{all}}$ are known a prior. The key notations are presented in Table I.

Maximizing the number of deadline-meeting transfers. The objective is to maximize the number of data transfers that can finish before their deadlines. Let the binary variable z_R denote whether a data transfer R can complete before its deadline, then the objective can be expressed by (1).

$$\max \sum_{R \in \mathcal{R}^{\text{all}}} z_R \tag{1}$$

Planning the topology configuration. Planning the network topology configuration is carried out by adjusting the wavelengths assignment among inter-datacenter links. Let integer variable $g_{l,t} \geq 0$ denote the number of wavelengths assigned to link l in timeslots t. When determining which directed link should carry how many wavelengths, the wavelength capacity of nodes and edges should be taken into account. Inequalities

(2)-(4) express the wavelength constraints on sender nodes, receiver nodes, and edges, respectively. Inequalities (5) enforce the valid values of variables $g_{l,t}, \forall(l,t)$.

$$\forall v,t: \sum_{l} I(l \in L_{v,\text{out}}) g_{l,t} \leq C^{s}_{v,t} \tag{2}$$

$$\forall v,t: \sum_{l} I(l \in L_{v,\text{in}}) g_{l,t} \leq C^{r}_{v,t} \tag{3}$$

$$\forall e,t: \sum_{l} I(l,e) g_{l,t} \leq C_{e,t} \tag{4}$$

$$\forall l,t: g_{l,t} \in \mathbb{N} \tag{5}$$

The indicator $I(l \in L_{v,\text{out}})$ denotes whether the directed link l is an outgoing link connection of node v, and $L_{v,\text{out}}$ denotes a set of the outgoing links that connect to node v. $I(l \in L_{v,\text{in}})$ denotes whether the directed link l is an incoming link of node v, and $L_{v,\text{in}}$ denotes a set of the incoming links that connect to node v. Lastly, $I(l,e)$ denotes whether the directed link l goes through edge (or fiber) e.

Allocating the transmission rate. We assume that each wavelength carries a capacity of c, hence the capacity of link l is $cg_{l,t}$ at time t. As this work considers multicast transfers, we use multiple forwarding trees for delivering the data. More specifically, we compute k Steiner trees for every transfer and plan at which rate each tree transmits data, we will describe the corresponding details later in §VI-A. In the following, let $\mathcal{K}_R$ denote a set of Steiner trees, each connecting the source and all the receivers of data transfer R. Let $x_{R,\kappa,t}$ denote the data transmission rate of a tree κ of request R at time t. (6) then enforces that all the data should be transferred before the deadline. Inequality (7) states that the traffic load on each link should not exceed the link capacity at any time, where $I(l \in \kappa)$ denotes whether a link l is traversed by tree κ, as the link load should not exceed the capacity. Lastly, Inequalities (8)-(10) enforce valid ranges for the variables $\boldsymbol{z}$ and $\boldsymbol{x}$.

Note that maximizing the number of admitted transfers under deadlines is NP-hard [41], already in fixed topologies. We can directly transfer hardness results from the fixed to the reconfigurable, by enforcing that only one meaningful reconfiguration exists, we briefly sketch a reduction: equip the original nodes with an infinite number of wavelengths, but at the same time, place new nodes on each edge (splitting them in two) that limit the connecting capacity to the one in the fixed setting.

$$\forall R: \sum_{t=t^{\text{arr}}_R}^{t^{\text{dl}}_R} \sum_{\kappa \in \mathcal{K}_R} \alpha x_{R,\kappa,t} = z_R f_R \tag{6}$$

$$\forall l,t: \sum_{R \in \mathcal{R}^{\text{all}}} \sum_{\kappa \in \mathcal{K}_R} x_{R,\kappa,t} I(l \in \kappa) \leq c g_{l,t} \tag{7}$$

$$\forall R: z_R \in \{0,1\} \tag{8}$$

$$\forall R,\kappa,t \notin [t^{\text{arr}}_R, t^{\text{dl}}_R]: x_{R,\kappa,t} = 0 \tag{9}$$

$$\forall R,\kappa,t \in [t^{\text{arr}}_R, t^{\text{dl}}_R]: x_{R,\kappa,t} \geq 0 \tag{10}$$

VI. Reservation-based Algorithm Details

We now present the details of the reservation-based transfer allocation and topology reconfiguration algorithm of *DaRTree*, including the adaptive routing component and the wavelength and rate allocation.

A. Load-Adaptive Multicast Routing

Previous work [17] that computed multiple multicast routing trees was load-*oblivious* manner, i.e., did not account for the current resource consumption. We improve this idea by weighing the links according to their leftover capacities and transfer load. In the following, we describe how we adapt link weights and then give details for the routing tree computation.

Link weight adaption. We initialize the link weight to be the reciprocal of the leftover capacities. For every link $l \in L$, we set the initial link weight w_l to $\frac{1}{c_l}$, where c_l is the remaining amount of capacity that is not used by the admitted data transfers. The remaining capacity c_l of a link l consists of two parts. The first part is the residual capacity of the total capacities of the assigned wavelengths minus the capacities reserved for previously admitted transfers $\mathcal{R}'$. Let $c^{\text{res}}_{l,t}$ and $g'_{l,t}$ respectively denote such residual capacity and the number of assigned wavelengths on link l at time t. Then we can calculate $c^{\text{res}}_{l,t}$ by $cg'_{l,t} - \sum_{R \in \mathcal{R}',\kappa,t} x_{R,\kappa,t} I(l \in \kappa)$. The second part is the capacity potential of the yet unassigned wavelengths. If link l is from node u to v on edge e, we can calculate the maximum number of wavelengths that can be assigned to it by $\min(\overline{C}^{s}_{u,t}, \overline{C}^{r}_{v,t}, \overline{C}_{e,t})$, where $\overline{C}^{s}_{u,t}$, $\overline{C}^{r}_{v,t}$, $\overline{C}_{e,t}$ denote the number of unassigned wavelengths node u can send, node v can receive, and edge e can carry at time t, respectively. So, the total potential capacities $c^{\text{free}}_{l,t}$ of unsigned wavelengths is $c \times \min(\overline{C}^{s}_{u,t}, \overline{C}^{r}_{v,t}, \overline{C}_{e,t})$ for link l at time t. We thus calculate the total amount c_l of leftover capacities on link l by $\sum_t (c^{\text{res}}_{l,t} + c^{\text{free}}_{l,t})$.

Tree computation. We now describe our method to compute multiple Steiner trees in order to balance the traffic load across the network. We compute the trees on a request by request basis and the k minimum-weight Steiner trees for each transfer request on a tree by tree basis. To this end, we iteratively increase the weight of a link by one if it appears on newly computed trees. For this link weight update, we use w_l to denote the current weight of link l. Assume that we have found a new Steiner tree κ' using current link weight, we increase the weight w_l of link l (e.g., increase w_l to $w_l + 1$) if it is on this tree, namely $l \in \kappa'$. Then, we feed the updated link weights to the tree computation algorithm to find the next min-weight Steiner tree. We repeat this iterative computation process until we obtained k trees for each transfer request.

Alternatives. One could also consider using link-disjoint Steiner trees to balance the traffic of data transfers across network links. However, in experiments, the load-adaptive tree generation outperformed this approach. The reason is that link-disjointedness is oblivious to the remaining link capacity. As such, e.g. routing two trees over a link with a large capacity is preferable over two links with small remaining capacity.

B. Wavelength assignment and rate allocation

We now specify how to compute an efficient wavelength assignment and rate allocation, in order to guarantee deadline satisfaction for as many multicast data transfers as possible.

Adapting the objective function. In the offline case with prior knowledge of all future transfer requests, one can directly find the global optimal solution that completes the maximum number of transfers before their deadlines, by solving the offline formulation. For the online problem on the other hand, we only know the transfer requests that have been submitted so far, and not the future ones. In principle, we could adapt the offline formulation in a greedy fashion to the online case, by maximizing the number of requests that just arrived at this timeslot, aiming to finish them as quickly as possible. However, we observed in preliminary experiments that this approach is too greedy in realistic workloads. More specifically, it congests the network in the near future, leaving no space for upcoming requests. We provide some intuition next.

Don't be too greedy. We use the example in Fig. 2 to illustrate that being too greedy is not the best choice. Request R_1 appears at the beginning of the first timeslot (time 0) with a deadline and size of 6, whereas R_2 appears after the first timeslot, with a size and deadline of 3 and 4, respectively. Fig. 2(a) shows how to greedily allocate request R_1, minimizing its completion time by assigning it the complete 2 units of capacity for the next 3 timeslots. However, when request R_2 arrives, R_1 already blocks nearly all resources, allowing only a single timeslots with 2 units of capacity, not enough to satisfy R_2. When we spread out the resource usage of R_1 until its hard deadline, R_2 can still be admitted, see Fig. 2(b). Hence, by scaling back the greediness of the allocation algorithm, we can admit both requests, instead of just one. We thus choose to minimize the amount of resource usage in *DaRTree*, in order to be prepared for future transfers. Note that it is never useful to waste resources in the current timeslot: we therefore maximize the transfer rates for the newly admitted requests in their first timeslot, as shown in Fig. 2(c).

Algorithm 1 summarizes our algorithm: it performs the admission control together with the wavelength assignment and the rate allocation solutions for a batch of transfer requests (newly submitted to the system).

Minimizing resource consumption. Inspired by the above example, we propose to allocate each admitted transfer a minimum rate s.t. it still meets its deadline. We further extend this idea and keep the number of needed wavelengths small, to freely allocate them for future requests in the next timeslot.

We thus formulate the wavelength assignment and rate allocation problem as an optimization objective that minimizes the wavelengths needed to satisfy the requests. We formulate this transfer problem as a mixed integer linear program (MILP) $P(\eta,\varepsilon)$ = {(5), (8), (11)-(19)}, where the objective (11) is to minimize the total weighted assigned number η of wavelengths and to maximize the number ε of deadline-satisfied data transfers under the constraints (5), (8), (12)-(19).

Observe that $P(\eta,\varepsilon)$ has two different optimization objectives, minimizing the weighted assigned number η of wavelengths across links and time slots, and maximizing the number ε of deadline-satisfied data transfers from a set $\mathcal{R}^{\text{cur}}$ that includes all transfers arriving at the start of the timeslot. Moreover, when not all requests can be admitted, we prefer to use link resources in earlier timeslots. To this end, we introduce a weight $w_{l,t}$ for wavelengths of link l in timeslots t and set the value of $w_{l,t}$ to t^2. Lastly, in order to obtain tractable runtimes, we use an iterative solver to find optimized values of η and ε.

$$F(\eta,\varepsilon) = \{\min \eta, \max \varepsilon\} \tag{11}$$

$$\forall v,t: \sum_{l} I(l \in L_{v,\text{out}}) g_{l,t} \leq \overline{C}^{s}_{v,t} \tag{12}$$

$$\forall v,t: \sum_{l} I(l \in L_{v,\text{in}}) g_{l,t} \leq \overline{C}^{r}_{v,t} \tag{13}$$

$$\forall e,t: \sum_{l} I(l,e) g_{l,t} \leq \overline{C}_{e,t} \tag{14}$$

$$\sum_{l,t} w_{l,t} g_{l,t} \leq \eta \tag{15}$$

$$\sum_{R \in \mathcal{R}^{\text{cur}}} z_R \geq \varepsilon \tag{16}$$

$$\forall R: (t^{\text{dl}}_R - t^{\text{arr}}_R) \sum_{\kappa \in \mathcal{K}_R} \alpha x_{R,\kappa} = z_R f_R \tag{17}$$

$$\forall l,t: \sum_{R \in \mathcal{R}^{\text{cur}}} \sum_{\kappa \in \mathcal{K}_R} x_{R,\kappa} I(l \in \kappa) I'(t \in [t^{\text{arr}}_R, t^{\text{dl}}_R]) \leq c^{\text{res}}_{l,t} + c g_{l,t} \tag{18}$$

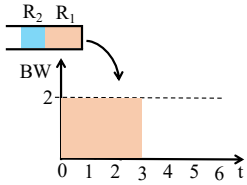
$$\forall R,\kappa: x_{R,\kappa} \geq 0 \tag{19}$$

Iterative solver. In this context, an iterative optimization solver fixes one of the two η,ε values and optimizes the other one. Hence, we start with $\varepsilon = m$, the total number of data transfers submitted in current time, and conduct a search to find the smallest η for which $P(\eta,m)$ is feasible (Line 2-Line 14, Algorithm 1). Ideally, we want to complete all requests before their deadline—however, the optimization problem may have no feasible solution if the remaining link capacity is insufficient. We then decrease the value of ε and recall $P(\eta,\varepsilon)$ (ε is a constant here). We repeat the above procedure until we find the minimum number of wavelengths to satisfy ε transfers.

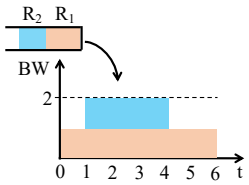
Solving $P(\eta,\varepsilon)$ by deterministic rounding. As the controller needs to decide the (new) wavelength assignment on every network link in every timeslot, the problem complexity naturally scales with deadline length and network size. More specifically, the optimization model contains integer variables which increase quadratically with transfer deadline and network scale, which makes it difficult to solve in real time for the transfers with far deadlines in networks with many links. We thus resort to a LP relaxation (Line 4, Algorithm 1)) and (deterministic) rounding algorithm to obtain solutions quickly. More specifically, we first relax the integer variables $g_{l,t}$ to be continuous and then solve the program $P(\eta,\varepsilon)$, i.e., we set

$$g_{l,t} \geq 0 \tag{20}$$

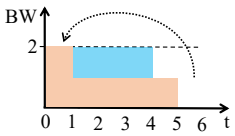
Given a fractional solution $\boldsymbol{g}^*$, we could obtain the integer wavelength solution $\hat{\boldsymbol{g}}$ by setting $\hat{g}_{l,t} = \lceil g^*_{l,t} \rceil, \forall l,t$.



(a) The greedy allocation of R_1 blocks the completion of R_2 until t=4. Only a transfer of size 2 is possible.



(b) A better allocation: allocating R_1 by minimizing resource usage allows R_2 to complete.



(c) Even more efficient than in Fig. 2(b), by filling up the capacity of the current timeslot.

Fig. 2. A greedy allocation can easily block future transfers, both requests could be admitted online with resource usage minimization.

Algorithm 1 Fast and efficient transfer allocation and topology reconfiguration algorithm

Input: A batch of m new transfer requests $\mathcal{R}^{\text{cur}} = \{R_1, R_2, \cdots, R_m\}$, a set of Steiner trees computed for the routing of these transfers, residual link capacities $\boldsymbol{c}^{\text{res}} = \{c^{\text{res}}_{l,t}, \forall l \in L, t\}$, unassigned wavelengths $\overline{C}^r_{i,t}$, $\overline{C}^s_{i,t}$, $\overline{C}_{e,t}$ $\forall i \in V, e \in E, t$.
Output: Admitted transfers, associated wavelength assignment, rate allocation that satisfies their deadlines.
1: Set aside wavelengths according to the routing trees;
2: Initialize $\varepsilon = m$;
3: **while** $\varepsilon > 0$ **do**
4: Build and solve optimization program with constraints from (8), (12)-(20) and an objective of (11) with given ε;
5: **if** feasible solution exists **then**
6: Admit ε new transfers;
7: Obtain the wavelength assignment fractional solution $\boldsymbol{g} = \{g_{l,t}, \forall l, t\}$, the admission decision $\boldsymbol{z} = \{z_R, \forall R\}$, and the rate allocation $\boldsymbol{x} = \{x_{R,\kappa}, \forall R, \kappa\}$
8: Round the fractional wavelength assignment to integral ones: $\overline{\boldsymbol{g}} = \{\overline{g}_{l,t} = \lceil g_{l,t} \rceil, \forall l, t\}$;
9: $\boldsymbol{c}^{\text{res}} \leftarrow$ UPDATERESIDUALCAPACITY($\boldsymbol{c}^{\text{res}}, \overline{\boldsymbol{g}}, \boldsymbol{z}, \boldsymbol{x}$);
10: **return** Admission decisions of transfer requests ($\boldsymbol{z}$) and rate allocation $\boldsymbol{x}$ of admitted transfers.
11: **else**
12: Decrease ε and set it to be $\varepsilon - 1$;
13: **end if**
14: **end while**
15: **return** Reject current submitted transfers $\mathcal{R}^{\text{cur}}$.

16: **function** UPDATERESIDUALCAPACITY($\boldsymbol{c}^{\text{res}}, \overline{\boldsymbol{g}}, \boldsymbol{z}, \boldsymbol{x}$)
17: **for** $(l \in L, t \in [\min_{R \in \mathcal{R}^{\text{cur}}}\{t^{\text{arr}}_R\}, \max_{R \in \mathcal{R}^{\text{cur}}}\{t^{\text{dl}}_R\}])$ **do**
18: $c^{\text{res}}_{l,t} = c^{\text{res}}_{l,t} + c\overline{g}_{l,t} - \sum_{R \in \mathcal{R}^{\text{cur}}} \sum_{\kappa \in \mathcal{K}_R} z_R x_{R,\kappa} I(l \in \kappa) I'(t \in [t^{\text{arr}}_R, t^{\text{dl}}_R)$;
19: **end for**
20: Fill up the current timeslot with the traffic of current requests $\mathcal{R}^{\text{cur}}$ allocated in future timeslots;
21: **end function**

However, directly rounding up the fractional solution $\boldsymbol{g}$ may violate the wavelength constraints (12)-(14) of the integer program. To obtain a feasible solution that satisfies the wavelength capacity constraints, we thus set aside a small amount of wavelengths ahead of time. Observe that if we were to reserve a wavelength for every link and reduce the maximum amount of wavelengths per fiber, we could always round up—but at the cost of efficiency. We improve this idea by only reserving wavelengths for links that are traversed by the forwarding trees of requests that arrived in the current timeslot.

Let $\mathcal{K}$ denote the set of routing trees computed for all the current requests, then we set aside $\sum_{\kappa \in \mathcal{K}} \sum_{l \in \kappa} I(l \in L_{v,\text{out}})$ and $\sum_{\kappa \in \mathcal{K}} \sum_{l \in \kappa} I(l \in L_{v,\text{in}})$ wavelengths for a node v to send and receive, respectively. In addition, we set aside $\sum_{\kappa \in \mathcal{K}} \sum_{l \in \kappa} I(l, e)$ wavelengths to not violate (14).

Updating the residual link capacity. After obtaining the solution of the wavelength assignment $\{g_{l,t} \forall l, t\}$ of links across time slots, the request admission decision $\{z_R, \forall R \in \mathcal{R}^{\text{cur}}\}$, and the rate $\{x_{R,\kappa}, \forall R \in \mathcal{R}^{\text{cur}}, \kappa \in \mathcal{K}_R\}$ allocated to accepted requests, it is easy to update the residual capacity $\{c^{\text{res}}_{l,t}, \forall l, t\}$. For every link l, its new residual capacity is calculated by adding all capacities carried by newly assigned wavelengths, and deducting all capacities reserved for the admitted transfers (Line 18, Algorithm 1).

C. Transfer requests with non-uniform utility

Up until now, we assumed that every transfer request brings the system the same or an equal amount of utility when it finishes before deadline. In other words, by maximizing the number of deadline-meeting transfers, we can maximize the total system benefit. However, the real system may gain different amount of utilities by data transfers generated by applications or clients with different-level of priority and importance. In such scenarios, the transfer scheduler should have the ability to perform admission control with preference in order to maximize the total system utility by admitting more transfer requests with larger utility. For simplicity, we use the abstract weight to denote the potential utility of each transfer request.

We can also employ Algorithm 1 to schedule weighted transfers requests just with some minor modifications. Let $w_R \in \mathbb{N}^+$ denote the weight of transfer request R, we can then adapt Algorithm 1 to weighted requests by replacing the constraint $\sum_R z_R \geq \varepsilon$ with (21) and initializing ε to be $\sum_R w_R$. However, due to the possible large value of the request weight, it may only find the optimal solution after a tedious amount of iterations with iterative searching (Line 3–Line 14, Algorithm 1), thus being too time-consuming.

$$\sum_{R \in \mathcal{R}^{\text{cur}}} w_R z_R \geq \varepsilon \tag{21}$$

We thus improve the algorithm's time efficiency via a two-phase computation. As we will show later, such an approach improves even upon a standard binary search in the application setting. In the first phase, we find the maximum number of deadline-meeting weighted requests by solving an optimization programming with objective of $\max \varepsilon$ and constraints of (12)-(14), (17)-(19), (20) and (21). Then, given the optimal objective value of ε found in this first phase, we compute the minimum number of wavelengths needed to achieve this objective, as well as the fitting rate allocation by solving an optimization program (similar to Line 4 in Algorithm 1).

Moreover, the two-phase computation can also be applied to speed up the computation of finding solutions for uniformly weighted requests. For example, in cases of heavy transfer

Algorithm 2 Joint allocation algorithm

Input: A set of newly incoming transfer requests $\mathcal{R}^{\text{cur}}$ at the beginning of current timeslot and a set of unfinished accepted transfers $\mathcal{R}^{'}$ till the end of last timeslots.
Output: The admission decisions of $\mathcal{R}^{\text{cur}}$, the wavelength assignment and the rate allocation of requests in $\mathcal{R}'$ and newly accepted requests in $\mathcal{R}^{\text{cur}}$ in timeslots t.
1: Compute load adaptive routing trees for requests in $\mathcal{R}^{\text{cur}}$ and $\mathcal{R}'$;
2: Set aside wavelengths according to the routing trees;
3: $\mathcal{R}^{\text{accept}} \leftarrow \varnothing$;
4: **if** $\mathcal{R}^{\text{cur}} \neq \varnothing$ **then**
5: $\quad \mathcal{R}^{\text{accept}} \leftarrow$ AdmissionControl$(\mathcal{R}^{\text{cur}}, \mathcal{R}')$.
6: **end if**
7: $\boldsymbol{x}, \boldsymbol{g} \leftarrow$ DeadlineGuranteeAllocation$(\mathcal{R}^{\text{accept}} \cup \mathcal{R}^{'})$;
8: Round fractional wavelength assignment to be integral: $\overline{\boldsymbol{g}} = \{\overline{g}_l = \lceil g_l \rceil, \forall l\}$;
9: Compute the residual capacity of the current timeslot;
10: Fill up the current timeslot with residual capacity to send as much data as possible.

11: **function** AdmissionControl$(\mathcal{R}^{\text{cur}}, \mathcal{R}^{'})$
12: $\quad$ Solve optimization program with the objective of (22) and constraints of (2)-(4), (8), (19), (20), and (23)-(25);
13: $\quad$ Obtain the newly accepted requests $\mathcal{R}^{\text{accept}} = \{R | z_R = 1, \forall R \in \mathcal{R}^{\text{cur}}\}$.
14: **end function**

15: **function** DeadlineGuranteeAllocation$(\mathcal{R}^{''})$
16: $\quad$ Solve linear optimization program with the objective of (26) and constraints of (19), (27)-(33);
17: $\quad$ Obtain the wavelength assignment $\boldsymbol{g} = \{g_l, \forall l\}$ and the rate allocation $\boldsymbol{x} = \{x_{R,\kappa}, \forall (R \in \mathcal{R}^{''}, \kappa)\}$ of all accepted requests $\mathcal{R}^{''}$.
18: **end function**

load (e.g., many requests arrive in the same timeslot), we might find the optimal admission rate only after many iterations when the admission rate is low (i.e., the network capacities are insufficient to admit most of these requests). Thus, iteratively searching from the total number of requests is not time-efficient. For these cases, we can also apply the two-phase computation to quickly find the optimal admission rate by solving a relaxed optimization program. However, in cases where *DaRTree* can admit nearly all requests or the number of arrival is low, the overhead of a two-phase computation is non-negligible in comparison to an iterative approach. The reason is that the iterative search only takes few rounds to check the feasibility of the optimization program, which is much easier and faster than the two-phase computation that solves a maximization optimization programming.

VII. Reallocation for Improved Efficiency

As described before, *DaRTree* allocates and reserves wavelengths and link capacities for the admitted requests upon arrival. This resource reservation can be considered as a calendar, which is simple for the system to operate and may also be beneficial for the served clients. However, *DaRTree* is also constrained by these reserved resources, reallocation in future timeslots could yield better system performance.

The efficiency can be improved by jointly scheduling the already accepted requests together with the newly revealed ones. To this end, we propose a new version of *DaRTree*, *DaRTreeJoint*, that does not explicitly reserve future resources for accepted transfer requests. Instead, before the beginning of every timeslot, it plans wavelength assignment, routing and rate allocation only for this current timeslot. According to the allocation results, the controller reconfigures the topology, the routing, and informs the senders of the transmission rates of the transfers originating from them. However, we still keep in mind that all admitted requests have to complete until their deadline. More specifically, our joint allocation algorithm *DaRTreeJoint* consists of the following three parts.

Step 1. We first perform admission control on newly incoming transfer requests of the current timeslot. Our objective is to admit the maximum number of newly transfer requests that can be finished until their deadline. Note that we must be careful to not violate such a promise to unfinished transfers accepted in previous timeslots. To this end, we employ an optimization program, with the objective of (22) and constraints of (2)-(4), (8), (19), (20), and (23)-(25) in Line 12 of Algorithm 2, to find the maximum number of acceptable new requests.

Step 2. Then, we determine the wavelength assignment and rate allocation in the current timeslot for both the newly and previously admitted transfers. For the accepted transfers, we allocate them at least at a minimum rate that guarantees their deadlines. Meanwhile, to fully use the network capacity, we maximize the throughput of the current timeslot by solving a linear optimization program with an objective of (26) and constraints of (19), (27)-(33).

Step 3. Finally, we completely utilize the current timeslot by using yet unallocated (wavelength) capacities.

$$\max \sum_{R \in \mathcal{R}^{\text{cur}}} z_R \tag{22}$$

$$\forall R \in \mathcal{R}^{\text{cur}} : (t_R^{\text{dl}} - t_R^{\text{arr}}) \sum_{\kappa \in \mathcal{K}_R} \alpha_R x_{R,\kappa} = z_R f_R \tag{23}$$

$$\forall R \in \mathcal{R}^{'} : (t_R^{\text{dl}} - \max(t_R^{\text{arr}}, t_{\text{cur}})) \sum_{\kappa \in \mathcal{K}_R} \alpha_R x_{R,\kappa} = f_R^{\text{remain}} \tag{24}$$

$$\forall l, t : \sum_{R \in \mathcal{R}^{\text{cur}} \cup \mathcal{R}'} \sum_{\kappa \in \mathcal{K}_R} x_{R,\kappa} I(l \in \kappa) I'(t \in [t_R^{\text{arr}}, t_R^{\text{dl}}]) \leq c g_{l,t} \tag{25}$$

$$\max \sum_{R \in \mathcal{R}^{''}} \sum_{\kappa \in \mathcal{K}_R} x_{R,\kappa} \tag{26}$$

$$\forall v : \sum_{l} I(l \in L_{v,\text{out}}) g_l \leq C_v^s \tag{27}$$

$$\forall v : \sum_{l} I(l \in L_{v,\text{in}}) g_l \leq C_v^r \tag{28}$$

$$\forall e : \sum_{l} I(l, e) g_l \leq C_e \tag{29}$$

$$\forall l : g_l \geq 0 \tag{30}$$

$$\forall R \in \mathcal{R}^{''} : \sum_{\kappa \in \mathcal{K}_R} \alpha x_{R,\kappa} \leq f_R^{\text{remain}} \tag{31}$$

$$\forall R \in \mathcal{R}^{''} : \sum_{\kappa \in \mathcal{K}_R} x_{R,\kappa} \geq \frac{f_R^{\text{remain}}}{\alpha(t_R^{\text{dl}} - \max(t_R^{\text{arr}}, t_{\text{cur}}))} \quad (32)$$

$$\forall l : \sum_{R \in \mathcal{R}^{''}} \sum_{\kappa \in \mathcal{K}_R} x_{R,\kappa} I(l \in \kappa) \leq c g_l \quad (33)$$

VIII. Evaluation

In this section, we present our evaluation results of *DaRTree* by comparing it to several state-of-the-art approaches in extensive simulations. We study multiple different scenarios and consider different real-world inter-datacenter networks. We next describe the simulation setup in §VIII-A. We show in §VIII-B that *DaRTree* outperforms prior work already for unicast transfers or for WAN topologies without reconfiguration. We also show that the integer-relaxation used by *DaRTree* yields efficient runtimes. A comprehensive general evaluation is then performed in §VIII-C and §VIII-D.

A. Simulation Setup

Network Topologies. We run simulations over four real-world inter-datacenter networks of large cloud service providers. Table II shows the details about these network topologies. Following the assumptions in [7], we assign an initial uniform capacity of 160 Gbps to every link, representing the static topology configuration. Analogously to the evaluations in [20], in our experiments, each wavelength can carry 10 Gbps. We place sender and receiver hardware accordingly. In order to facilitate meaningful and realistic reconfiguration scenarios, we allow a link to carry up to 50% more wavelengths than initially assigned (at the cost of borrowing adjacent wavelengths).

Transfer workloads. We use synthetic models to generate multicast transfer requests similar to related work [5, 7, 10, 20]. We assume a slotted timeline, where time is measured in the number of timeslots, where each slot has a length of 5 minutes. Transfer requests arrive at the system at the beginning of each timeslot. To generate transfer requests, we model the request arrival time as a Poisson process, where the *arrival rate factor* per timeslot is λ. For an experiment that simulates a time span of T_{span} timeslots, we generate λT_{span} transfer requests on average. For each transfer request, we randomly choose a datacenter as the source and $\gamma(N-1)$ other datacenters as the receivers, where the *receiver factor* $\gamma \in [10\%, 100\%]$ and N is the total number of datacenters in a network. We choose the deadline for each data transfer from a uniform distribution between [T, δT], where T is the timeslot length and δ is a factor used to change the tightness of deadlines. We will refer to this factor as the *deadline factor* in the following. To generate the data size for each transfer, we integrate the average transfer throughput under an Exponential distribution with a mean of 20 Gbps. Then, we calculate the data size by multiplying the throughput by the transfer lifespan, e.g., the data size would be 9TB on average for a transfer with an one hour deadline.

Simulation Environment. We performed all simulations using a Python script that employs MOSEK [43] as our backend solver to find the solutions to the optimization models.

Comparison with the state of the art. We compare *DaRTree* with the following related works:

Name	Description
Internet2 [20]	ISP network with 9 datacenters and 18 inter-DC links.
GScale [4]	Google's inter-DC WAN which has 12 datacenters and 19 inter-datacenter links.
Equnix [42]	An inter-DC WAN from Equnix, which connects 20 datacenters using 141 inter-datacenter links.
IDN [3]	Microsoft's inter-datacenter WAN with 40 datacenters, each connected to 2-16 other datacenters.

TABLE II. Topologies used in our simulations

- *MTree* [17] adopts k trees to optimize multicast transfers in static topologies.
- *Amoeba* [7] allocates rates over k-paths for each admitted data transfer and aims to guarantee the deadline for as many unicast transfers as possible in static topologies.
- *Owan* [20] also adopts k-paths to deliver data and jointly controls network topology and transmission rates of paths to reduce the completion time or satisfy deadlines for inter-datacenter unicast transfers. Specifically, we use the algorithm proposed by *Owan* that optimizes data transfers with deadlines.

All the compared approaches, excluding *Owan*, use admission control for multicast transfers submitted to the system. *Owan* on the other hands accepts every incoming transfer and aims to complete transfers as quickly as possible in order to meet the deadline. As such, *Owan* is particularly suited for environments where a majority of requests can be completed under deadlines, most of our scenarios fulfil this assumption.

To simulate *Owan* and *Amoeba* in our setting, we split each multicast transfer into multiple unicast transfers. We count the multicast transfers that are delivered completely to all receivers in *Owan* and *Amoeba*, and denote them as *Owan*- and *Amoeba*-Multicast, respectively. For further comparisons in §VIII-C, we also allow fractional completion of multicast data transfers for *Owan* and *Amoeba* (e.g. to just 2 of 3 receivers), denoted as *Owan*- and *Amoeba*-Unicast, respectively.

Performance metrics. We evaluate the approaches in a wide spectrum of performance metrics, such as deadline-met ratio of multicast (unicast) transfers, throughput of multicast (unicast) transfers, and runtime.

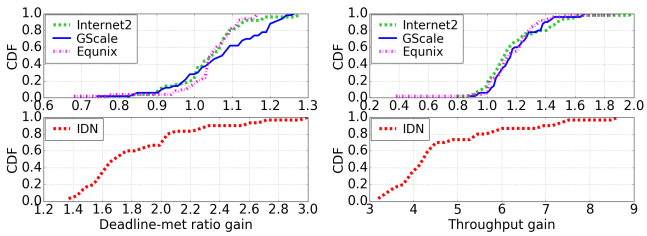
In §VIII-B and §VIII-C, we consider requests with uniform weight in comparison of Algorithm 1 (*DaRTree* for brevity) to other related approaches. We then conduct further simulations in §VIII-D to show how *DaRTree* adapts to the non-uniform weight (*DaRTreeWeight*) and what further improvements can be obtained by reallocating (not aborting) already admitted requests (Algorithm 2, coined *DaRTreeJoint*). Table III summarizes the differences among the three *DaRTree* variants.

B. Unicast, Static WAN, and Runtime Experiments

Unicast transfers: *DaRTree* vs. *Owan*. We first evaluate how *DaRTree* compares against *Owan* even if there are just unicast requests (i.e., all the arriving transfer requests only have one sender/receiver). To simulate unicast transfers, we randomly select one datacenter as the receiver for each transfer. We set the deadline factor δ to six (0.5 hours), which will generate transfers with a size following an exponential distribution with

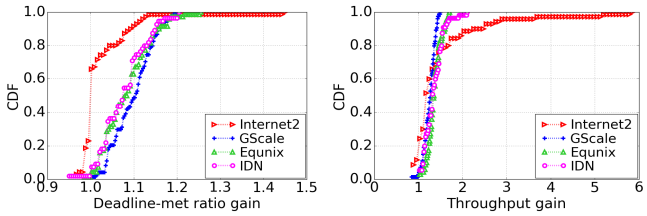
Property	DaRTree	DaRTreeWeight	DaRTreeJoint
Accepted requests always complete before deadline?	**Yes**	**Yes**	**Yes**
Requests with non-uniform weights?	**No** (uniform weights)	**Yes**	**No** (uniform weights)
Reschedule running requests?	**No** (resource reservation)	**No** (resource reservation)	**Yes**

TABLE III. Comparison between the different versions of DaRTree



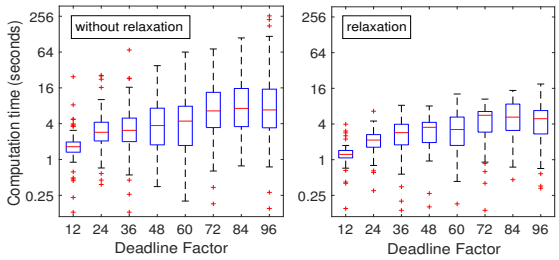
(a) Deadline-met ratio comparison (b) Throughput comparison

Fig. 3. *DaRTree* outperforms *Owan* even for unicast transfers.

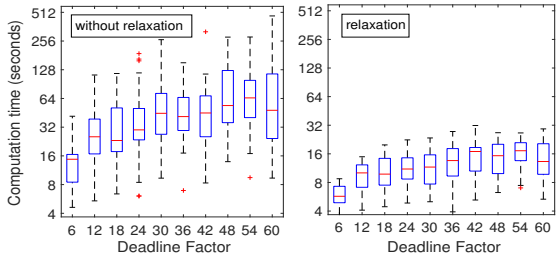


(a) Deadline-met ratio comparison (b) Throughput comparison

Fig. 4. *DaRTree* outperforms *MTree* even in static WANs, i.e., without topology reconfiguration.



(a) Internet2



(b) IDN

Fig. 5. Computation time comparison of *DaRTree* with and without relaxation. Please note that the y-axis is *logarithmic*.

a mean of 4.5TB. Each run simulates 2.5 hours (30× 5-minute timeslots), with a request arrival rate λ randomly chosen from $\{1, 2, 3\}$. Hence, we will on average generate 30 to 90 transfer requests for each run. We collect the percentage of successfully admitted requests and the average network throughput. To evaluate how *DaRTree* compares to *Owan*, we compute the performance gain by dividing the transfer deadline-met ratio and the network throughput of *DaRTree* by those of *Owan*, respectively.

Fig. 3 reports the CDF of the performance gain in the transfer deadline-met ratio and the average network throughput over 50 experiments. Compared to *Owan*, *DaRTree* achieves a higher deadline-met ratio in about 75% experiments in the Internet2, GScale and Equnix topologies. In the IDN topology, *DaRTree* consistently outperforms *Owan* for every experiment and improves the deadline-met ratio by 0.3× to 2×. In addition, the throughput results in Fig. 3(b) show that *DaRTree* outperforms *Owan* on network throughput in around 80% experiments over Internet2, GScale and Equnix topologies and improves the average network throughput by at least 2× and up to 7.8× in the IDN topology. Hence, we can conclude that the relaxed optimization-based allocation in *DaRTree* outperforms the simulated annealing algorithm of *Owan* in most unicast experiments.

Static WANs: *DaRTree* vs. *MTree*. We next evaluate how *DaRTree* compares against *MTree* if the topology is not reconfigurable, i.e., in static WANs. To this end, we turn off all functions relating to the reconfiguration part in *DaRTree*. Fig. 4(a) plots the CDF of the deadline-met ratio gain of *DaRTree*. Compared to the CDFs in Fig. 3, the advantage of *DaRTree* versus *MTree* is less prevalent. However, *DaRTree* can still admit around 10% to 40% more transfers than *MTree* in 25%-55% of the experiments, with the remaining ones being very close. Moreover, Fig. 4(b) shows that *DaRTree* achieves a higher network throughput for more than 99% of the experiments, compared to *MTree*. The point $(5.8, 1)$ in particular highlights that *DaRTree* can obtain up to 5.8× higher throughput than *MTree* in the IDN network: the reason is that *DaRTree* uses load-adaptive routing trees, which distribute traffic more evenly across the network, even without reconfiguration.

Runtime improvement due to relaxation. We now evaluate the time efficiency of *DaRTree* by comparing it with a version that omits our rounding method and uses integer variables $\boldsymbol{g}$ to find the solution. Since the deadline is key to determining the number of variables $\boldsymbol{g}$, we generate transfers with varying deadline factors, in the smallest and largest real-world topology.

Fig. 5 plots the computation time for the Internet2 and the IDN topologies, with 40-80 allocations per algorithm and deadline factor. Figs. 5(a) and 5(b) show that the computation time is up to 250 seconds longer for the Internet2 network and up to 400 seconds longer for the IDN network, respectively.

In contrast, *DaRTree* maintains a relatively small computation time, no more than 15 seconds for the Internet2 network and 30 seconds for the IDN network. In addition, we can also see that the computation time of both approaches increases as the deadline factor grows and as the network size scales up. We thus conclude that the integer relaxation technique in *DaRTree*

significantly reduces computation times.

C. General Performance Evaluation of DaRTree

We now evaluate the impact of the different parameters used to generate data transfers on the performance of the different approaches. We parametrize: 1) the request arrival rate factor λ, 2) the deadline factor δ, and 3) the receiver fraction factor γ. We conduct five runs for each setting per parameter, in every topology, and evaluate all approaches in each run. We report on the average number of data transfers that meet their deadlines and the average network throughput of these experiments.

Impact of the request arrival rate factor. We now evaluate the impact of the request arrival rate. We simulate a timespan of 60 timeslots for each run, fix the deadline factor to 6, and randomly choose $\{20\%, 30\%, 50\%\}$ of the datacenters as receivers. We vary the request arrival rate λ from 1 to 5.

Fig. 6 plots the average percentage of data transfers that can meet their deadlines and the average network throughput obtained under different request arrival rates over the four network topologies. Figs. 6(a)-6(d) show that the percentage of deadline-met data transfers decreases as the request arrival rate increases for all four approaches. These results are as expected since both the number of data transfers submitted to the system and the network traffic load increases as the request arrival rate grows. However, we can see that *DaRTree* always maintains a deadline-met ratio at a high level of 80% to 100% and outperforms all other approaches. Moreover, *DaRTree* can satisfy the deadlines for up to 30% more multicast transfers, compared to *Owan*, *Amoeba* and *MTree*. We also see that although *Owan* and *Amoeba* achieve a high deadline-met ratio for unicast transfers, they obtain relatively low deadline-met ratios for multicast transfers. The reason is that they only focus on guaranteeing the deadline for each individual unicast transfer and may fail to satisfy the deadline for all the receivers of the multicast transfer. Regarding *MTree*, its transfer deadline-met ratio drops dramatically as the request arrival rate increases. It outperforms *Owan* and *Amoeba* for deadline satisfaction of multicast transfers on the Internet2 and the GScale topologies, but falls behind when the request arrival rate is $\lambda \geq 2$ on both Equnix and IDN topologies[3].

Figs 6(e)-6(h) plot the network throughput of all compared approaches, normalized by that of *DaRTree*. We observe that *DaRTree* achieves 20%-70% and 40%-70% higher throughput than *MTree*, *Amoeba* and *Owan*, respectively. Even against fractional completion, the throughput is 20% to 40% higher.

Impact of the deadline factor. In this part, we evaluate how the tightness of the deadline impacts the performance of the approaches. We generate 7TB of data for each transfer and adjust the deadline factor δ from 5 to 25 to simulate different deadlines. Fig. 7 plots the percentage of transfers that meet their deadlines and the average throughput under different deadline factors. Naturally, more transfers will meet their deadlines as the deadline factor increases due to higher flexibility, as shown in Fig. 7(a)-7(d). *DaRTree* admits over 95% of the multicast transfers, roughly 10% to 30% more than the best of the other approaches. The results are similar for the throughput gain. Maybe interestingly, the performance of *MTree* degrades for larger topologies, unlike our *DaRTree* approach.

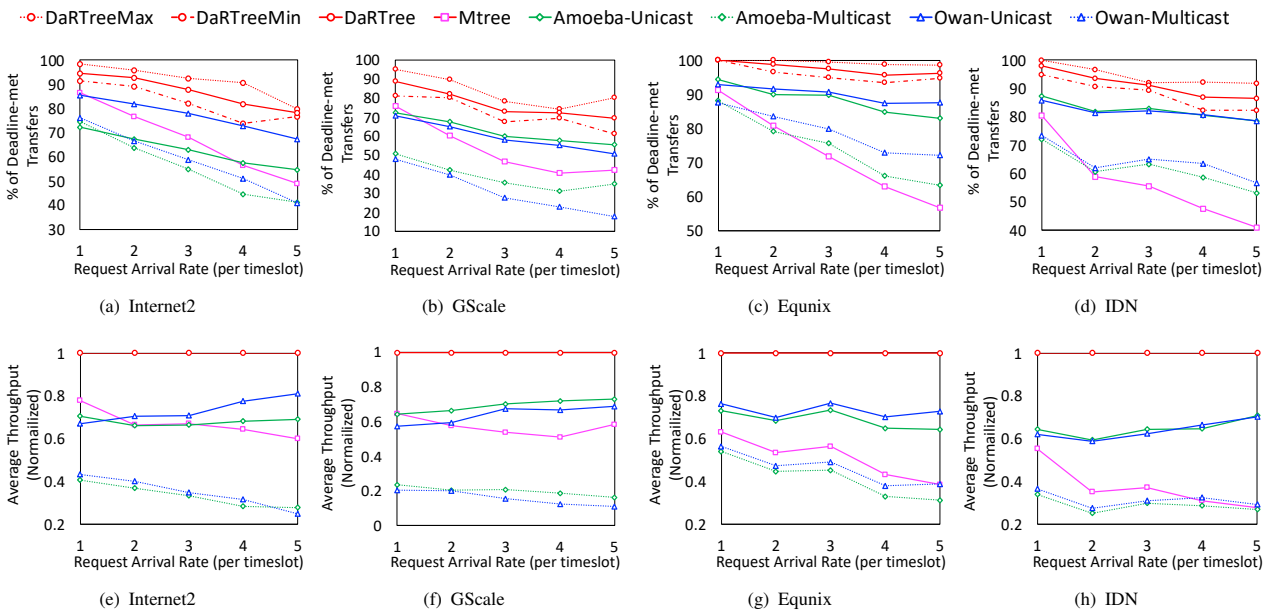
Receiver factor. In the last set of experiments, we evaluate the impact of the number of receivers for multicast transfers. To this end, we generate a constant data size for each transfer and set it to be 5TB (Internet2), 7TB (GScale), 10TB (Equinix), 14TB (IDN). Each transfer has a deadline of 6 timeslots (0.5 hours). To generate transfers with a varying number of receivers, we set the receiver factor of every multicast transfer to be different percentages of datacenters. Figs. 8(a)-8(d) show the factor of improvement on the percentages of transfers that meet their deadlines. Compared to *MTree*, *DaRTree* accepts around 10%-20% more multicast transfers in the four topologies. Against *Amoeba* and *Owan*, *DaRTree* satisfies at least 5% and up to 49% more multicast transfers. We can also observe that the improvement of deadline-met multicast transfers increases as the number of transfer receivers increases and as the network scales up. Figs. 8(e)-8(h) show the factor of improvement on the average network throughput. Compared to *MTree*, *DaRTree* improves the average network throughput by $1.15\times$ to $1.42\times$. In relation to *Amoeba*, *DaRTree* improves the average throughput of unicast transfers by up to $1.79\times$ and that of multicast transfers by up to $7.37\times$. Lastly, for *Owan*, *DaRTree* improves the average throughput of unicast transfers by $1.16\times$ to $2.24\times$ and that of multicast transfers by up to $8.63\times$. The trend of improvements on network throughput is similar to that of the deadline-satisfied transfers, remaining roughly identical for *MTree* and rising as the number of receivers increases for both *Amoeba* and *Owan*.

Summary. The results from §VIII-B indicate that the performance of *DaRTree* goes beyond simply combining multicast routing and reconfigurable WANs: in both scenarios, we improve upon prior work, in particular for larger networks. As we have seen in §VIII-C, leveraging load-adaptive Steiner trees and a rounding-based optimization significantly outperforms state of the art approaches in all four simulated real-world topologies. In particular, we improve the transfer admission rate and the throughput by up to $1.7\times$ in larger networks. We next discuss further extensions of *DaRTree* to more settings.

D. DaRTree for Non-Uniform Weights and Reallocation

Requests with non-uniform weight. To simulate weighted transfer requests, we assign each data transfer with a weight that corresponds to its size and deadline. The reasoning behind this idea is that larger requests should provide more utility, as well as more urgent requests. More precisely, the weight w_R of a request R is assigned by taking the quotient of the size $|\boldsymbol{d}_R| f_R$ of all receivers over its lifespan $t_R^{\text{dll}} - t_R^{\text{arr}}$, the duration from arrival t_R^{arr} until its deadline t_R^{dll}. We compare the normal version algorithm of *DaRTree* and the variant, *DaRTreeWeight*, that is designed for weighted requests (§VI-C). We evaluate them

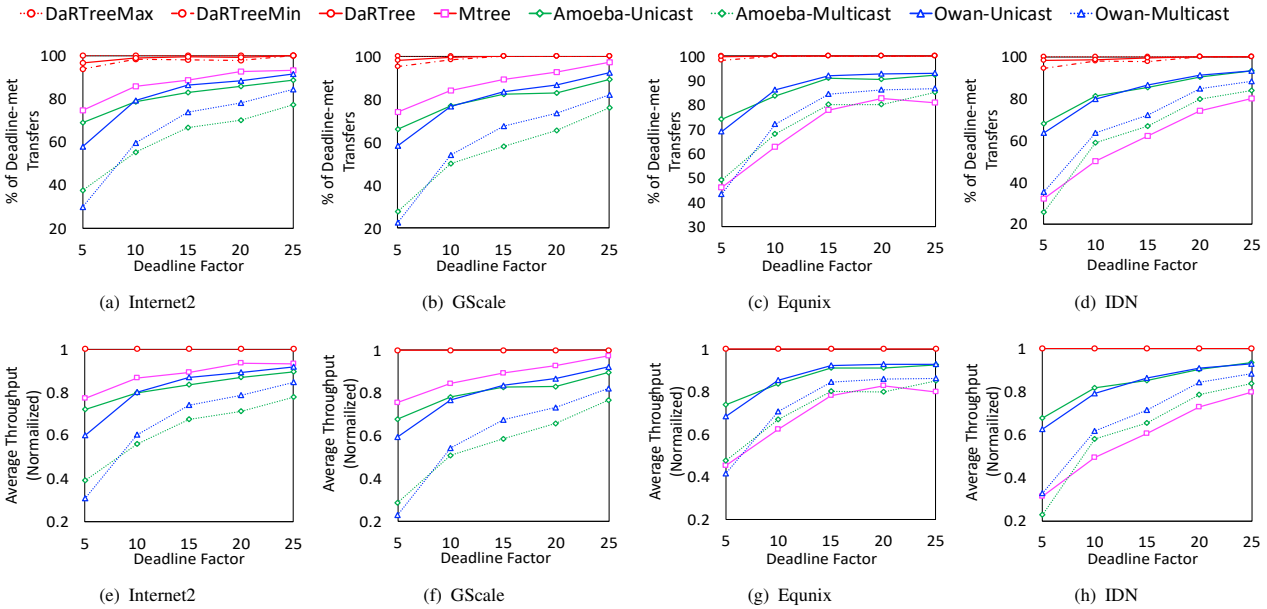
[3]We note that a similar performance of *Owan* and *Amoeba* was already visible in [20, Fig. 9], with slightly better results for *Owan*, which is consistent with our results in this and the following experiments.



(a) Internet2 (b) GScale (c) Equnix (d) IDN

(e) Internet2 (f) GScale (g) Equnix (h) IDN

Fig. 6. Impact of the request arrival rate. (a-d) show how many transfers meet their deadline, (e-f) show network throughput, respectively. *DaRTree*Min denotes the minimum value for *DaRTree* over all runs, whereas *DaRTree*Max shows the maximum value over all runs.

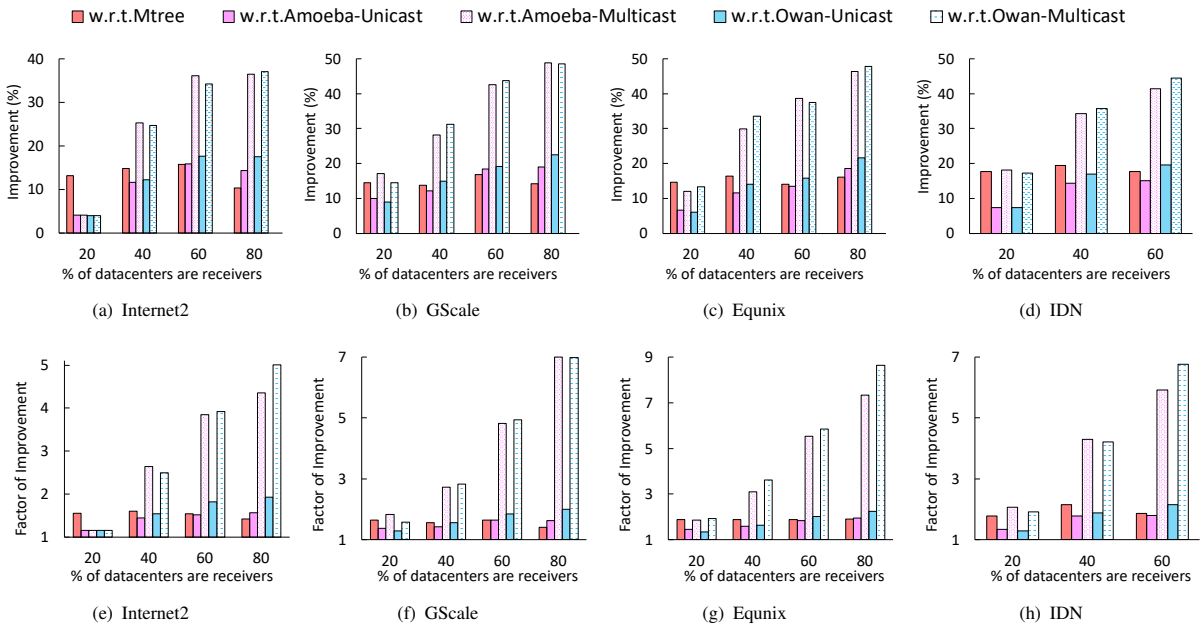


(a) Internet2 (b) GScale (c) Equnix (d) IDN

(e) Internet2 (f) GScale (g) Equnix (h) IDN

Fig. 7. Impact of the deadline factor. (a-d) show the deadline-met ratio, and (e-f) show the network throughput, respectively. *DaRTree*Min denotes the minimum value for *DaRTree* over all runs, whereas *DaRTree*Max shows the maximum value over all runs.

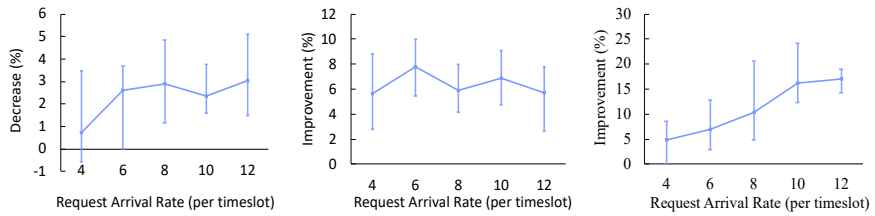
in the Internet2 topology under different request arrival rates. In these experiments, the request deadline is randomly chosen from 10 minutes to 3 hours, by changing the deadline factor δ, and the receiver factor γ is randomly chosen from 20% to 100%. In addition to the percentage of deadline-met transfers, throughput and computation time, we also collect the weighted percentage of deadline-met transfers as $\frac{\sum_{R\in\mathcal{R}^{\text{accept}}} w_R}{\sum_{R\in\mathcal{R}^{\text{all}}} w_R} \times 100\%$, where w_R denotes the weight of transfer request R and $\mathcal{R}^{\text{accept}}$, $\mathcal{R}^{\text{all}}$ denote the set of accepted deadline-meeting transfers and that of all simulated transfers, respectively. Fig. 9(a) shows that *DaRTreeWeight* obtains 1%-3% lower percentage of deadline-met transfers on average, compared to *DaRTree*. However, we can see from Fig. 9(b) that it in average obtains a 5% higher weighted percentage of deadline-met transfers than *DaRTree*. Moreover, *DaRTreeWeight* obtains 5%-15% higher throughput on average, as shown in the results of Fig. 9(c). These results are as expected as *DaRTreeWeight* prefers to admit the transfers with larger weight (potential of the throughput), which results in admitting slightly less transfers overall. In contrast, *DaRTree* treats every transfer equally and accepts slightly more transfers with smaller size in turn. We also collect the runtime of the



(a) Internet2 (b) GScale (c) Equnix (d) IDN

(e) Internet2 (f) GScale (g) Equnix (h) IDN

Fig. 8. Receiver factor impact. (a-d) are improvements (in %) of transfers that meet their deadlines, (e-f) are network throughput improvements.



(a) Acceptance ratio (unweighted) (b) Acceptance ratio (weighted) (c) Throughput

Fig. 9. Impact on the acceptance ratio and throughput when employing *DaRTreeWeight* over *DaRTree*.

variants of *DaRTree* and of *DaRTreeWeight*. Fig. 10 shows the runtime time of *DaRTree* when applying the iterative search (solver) to schedule requests with uniform weights (Iterative search + uniform weight), that of the variant of *DaRTree* which adopts binary search to handle requests with non-uniform weights (Binary search + Non-uniform weight) and that of *DaRTreeWeight* which uses our two-phase computation to allocate requests with non-uniform weights (Two-phase computation + Non-uniform weight). We can see that when there are only few requests to allocate, both the iterative search and the two-phase computation can terminate within 10 seconds on average to allocate requests with uniform weight and non-uniform weight, respectively. But, as the number of to-be-allocated requests increases, the iterative search becomes slow and takes about 30 seconds and up to about 100 seconds to allocate requests. This is because when the request arrival rate is 12, the network is heavily overloaded. The iterative search takes about 8 to 9 iterations to find optimal solutions. In contrast, the two-phase computation can terminate within about 10 seconds, in nearly all cases. The binary search approach is always outperformed by the two-phase computation. Note that the plots for iterative and binary search are for uniform and non-uniform weights, respectively.

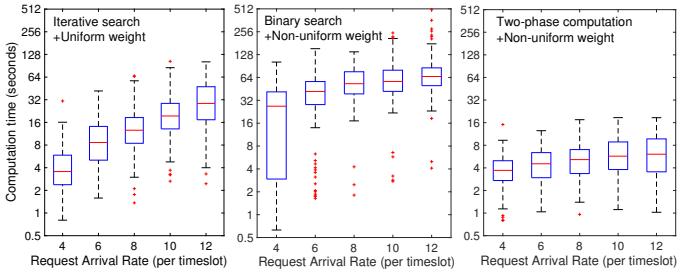
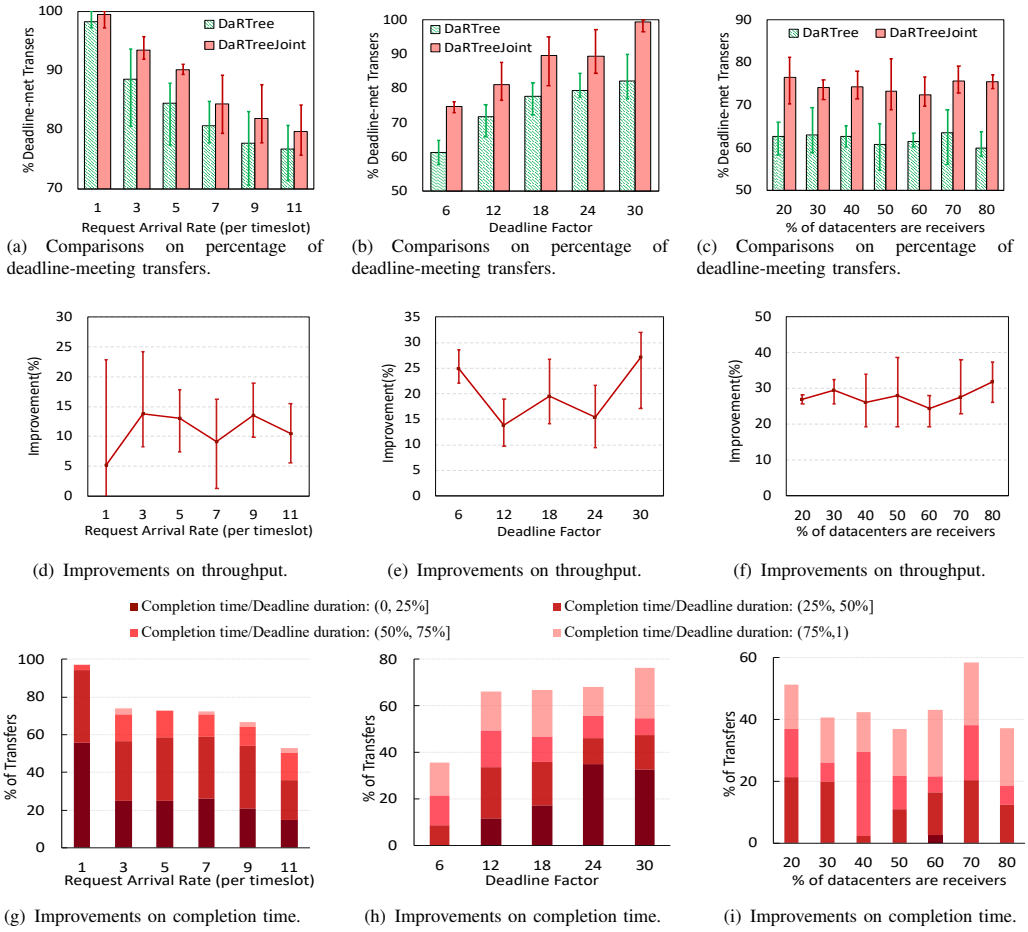


Fig. 10. Runtime comparison (log-scale) for non-/uniformly weighted transfer requests. Our two-phase computation (right, *DaRTreeWeight*) is faster than a standard binary search (middle), and even than an iterative search for uniform weights (left, *DaRTree*).

Reallocation of requests with *DaRTreeJoint*. Lastly, in order to measure the benefit of reallocation, we compare *DaRTree* and *DaRTreeJoint* across different request arrival rates, deadline factors and receivers. Note that in this collection of experiments, we only simulate transfer requests with uniform weights. Fig. 11 plots the results obtained from the GScale topology. Overall, *DaRTreeJoint* outperforms *DaRTree* in deadline-met transfers, throughput and completion time. Fig. 11(a), 11(d), 11(g) show the results obtained under different request arrival rates. We can see that as the request arrival rate increases, both *DaRTree* and *DaRTreeJoint* yield a decreasing percentages of deadline-met transfers, as the results in Fig. 11(a)



(a) Comparisons on percentage of deadline-meeting transfers.
(b) Comparisons on percentage of deadline-meeting transfers.
(c) Comparisons on percentage of deadline-meeting transfers.
(d) Improvements on throughput.
(e) Improvements on throughput.
(f) Improvements on throughput.
(g) Improvements on completion time.
(h) Improvements on completion time.
(i) Improvements on completion time.

Fig. 11. *DaRTreeJoint* improves the request acceptance ratio, throughput and transfer completion time.

show. However, *DaRTreeJoint* can always achieve about 5% more deadline-met transfers. The results in Fig. 11(d) show that *DaRTreeJoint* also achieves around a stable 10% higher throughput than *DaRTree*. Fig. 11(g) shows that *DaRTreeJoint* finishes about 97% and 52% transfers earlier than their deadlines even when the request arrive rate is 1 and 11, respectively and finishes around 14% and 55% transfers within a quarter of their deadlines. Fig. 11(b), 11(e), 11(h) shows the results obtained under different deadline factors. As the deadline factor increases (or the deadline becomes looser), we can see an increase in percentages of deadline-met transfers for both algorithms from Fig. 11(b). *DaRTreeJoint* achieves at least 10% more deadline-met transfers. This more deadline transfers leads *DaRTreeJoint* to a round 15% higher throughput. Fig. 11(h) shows *DaRTreeJoint* can finish 35% to 76% earlier than their deadlines. Fig. 11(c), 11(f), 11(i) show the results obtained under different number of receivers. Fig. 11(c) shows that both algorithms maintain a fairly stable percentage of deadline-met transfers when the number of receivers changes.

However, *DaRTreeJoint* also outperforms *DaRTree* in accepting at least 12% more transfers, achieving 25% higher throughput, and finishing around 40% transfers earlier than their deadline. Fig. 12 shows the runtime of *DaRTree* and *DaRTreeJoint*. We can see that the runtime of both *DaRTree*

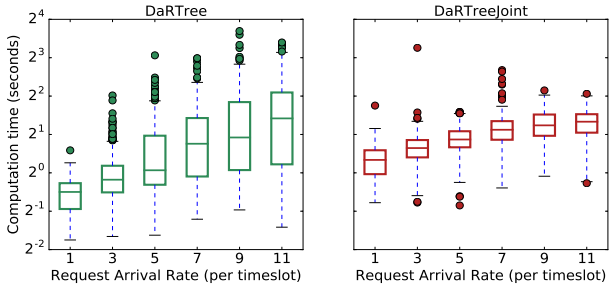


Fig. 12. Runtime comparison between *DaRTree* (left) and *DaRTreeJoint*(right)

and *DaRTreeJoint* is just a few seconds on average. For small request arrival rates from 1 to 5, *DaRTree* is a bit faster than *DaRTreeJoint* but both can terminate in up to 5 seconds. As the request arrival rate increases from 5 to 11, *DaRTreeJoint* performs slightly better than *DaRTree*.

As such, by aggressively reorganizing admitted requests during their lifetime, *DaRTreeJoint* can further increase acceptance ratio and throughput. Even though all admitted requests will be completed until their deadline, a downside is that the clients have less planning certainty about their sending/receiving rates.

IX. Conclusion

Our work was motivated by the rapidly increasing scale of geo-replication and the recently uncovered possibilities of

physical layer adaptation in the WAN. To this end, we presented *DaRTree*, an efficient approach to maximize the online admission of deadline-sensitive multicast transfer requests in reconfigurable WANs. *DaRTree* leverages 1) load-adaptive Steiner tree routing and 2) topology reconfiguration via relaxed optimization solvers for greater efficiency, without requiring rescheduling or preemption. Our extensive simulations for real-world topologies showed that *DaRTree* significantly improves the network throughput and the number of admitted requests over prior work. *DaRTree* also enhances the performance of unicast transfers in reconfigurable WANs and of multicast transfers in WANs without reconfiguration. Moreover, *DaRTree* can be efficiently adapted to handle non-uniform transfer utility functions and ongoing reallocation of admitted requests. We believe that our work opens several interesting avenues for future research. In particular, it will be interesting to explore opportunities in the context of randomized rounding or the potential benefits of allowing preemptions.

Reproducibility. In order to simplify future research and in order to make our results reproducible, we will share our implementation[4] and experimental results with the research community together with this paper.

Acknowledgements. We would like to thank the authors of [20] for providing us with their source code. We would also like to thank the anonymous reviewers for their helpful comments.

REFERENCES

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

[2] Cisco, "Cisco global cloud index: Forecast and methodology, 2016–2021 white paper," https://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.html, online: accessed 15-April-2019.

[3] C.-Y. Hong, S. Kandula *et al.*, "Achieving high utilization with software-driven WAN," in *ACM SIGCOMM*, 2013.

[4] S. Jain, A. Kumar, S. Mandal *et al.*, "B4: Experience with a globally-deployed software defined WAN," in *ACM SIGCOMM*, 2013.

[5] S. Kandula, I. Menache, R. Schwartz *et al.*, "Calendaring for wide area networks," in *ACM SIGCOMM*, 2014.

[6] N. Mohammad, C. S. Raghavendra, K. Srikanth, and R. Sriram, "QuickCast: Fast and efficient inter-datacenter transfers using forwarding tree," in *IEEE INFOCOM*, 2018.

[7] H. Zhang, K. Chen, W. Bai *et al.*, "Guaranteeing deadlines for inter-data center transfers," *IEEE/ACM Trans. Netw.*, vol. 25(1), pp. 579–595, 2017.

[8] Y. Zhang, J. Jiang, K. Xu *et al.*, "BDS: a centralized near-optimal overlay network for inter-datacenter data replication," in *ACM EuroSys*, 2018.

[9] V. Jalaparti, I. Bliznets, S. Kandula *et al.*, "Dynamic pricing and traffic engineering for timely inter-datacenter transfers," in *ACM SIGCOMM*, 2016.

[10] L. Luo, H. Yu, Z. Ye, and X. Du, "Online deadline-aware bulk transfer over inter-datacenter WANs," in *IEEE INFOCOM*, 2018.

[11] M. Noormohammadpour and C. S. Raghavendra, "Datacenter traffic control: Understanding techniques and tradeoffs," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 2, pp. 1492–1525, 2018.

[12] N. McKeown, T. Anderson, H. Balakrishnan *et al.*, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[13] P. Bosshart, D. Daly, G. Gibb *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[14] M. Wichtlhuber, J. Kessler, S. Bucker *et al.*, "Soda: Enabling CDN-ISP collaboration with software defined anycast," in *IFIP Networking*, 2017.

[15] M. Noormohammadpour *et al.*, "DCCast: Efficient point to multipoint transfers across datacenters," in *USENIX HotCloud*, 2017.

[16] M. Noormohammadpour and C. S. Raghavendra, "DDCCast: Meeting point to multipoint transfer deadlines across datacenters using ALAP scheduling policy," *arXiv preprint arXiv:1707.02027*, 2017.

[17] S. Ji, S. Liu, and B. Li, "Deadline-aware scheduling and routing for inter-datacenter multicast transfers," in *IEEE International Conference on Cloud Engineering (IC2E)*, 2018, pp. 124–133.

[18] 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.

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

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

[21] S. Jia, X. Jin, G. Ghasemiesfeh, J. Ding, and J. Gao, "Competitive analysis for online scheduling in software-defined optical WAN," in *IEEE INFOCOM*, 2017.

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

[23] R. Singh, M. Ghobadi, K.-T. Foerster, M. Filer, and P. Gill, "RADWAN: rate adaptive wide area network," in *ACM SIGCOMM*, 2018.

[24] L. Luo, H. Yu, and Z. Ye, "Deadline-guaranteed point-to-multipoint bulk transfers in inter-datacenter networks," in *IEEE ICC*, 2018.

[25] N. Laoutaris, M. Sirivianos, X. Yang *et al.*, "Inter-datacenter bulk transfers with netstitcher," in *ACM SIGCOMM*, 2011.

[26] N. Laoutaris, G. Smaragdakis, R. Stanojevic *et al.*, "Delay-tolerant bulk data transfers on the Internet," *IEEE/ACM Trans. Netw.*, vol. 21, no. 6, pp. 1852–1865, 2013.

[27] Y. Feng, B. Li, and B. Li, "Postcard: Minimizing costs on inter-datacenter traffic with store-and-forward," in *IEEE ICDCS Workshops*, 2012, pp. 43–50.

[28] Y. Wang, S. Su, A. X. Liu, and Z. Zhang, "Multiple bulk data transfers scheduling among datacenters," *Computer Networks*, vol. 68, pp. 123–137, 2014.

[29] Y. Wu, Z. Zhang, C. Wu *et al.*, "Orchestrating bulk data transfers across geo-distributed datacenters," *IEEE Transactions on Cloud Computing*, vol. 5, no. 1, pp. 112–125, 2017.

[30] R. Singh, M. Ghobadi, K.-T. Foerster, M. Filer, and P. Gill, "Run, walk, crawl: Towards dynamic link capacities," in *ACM HotNets*, 2017.

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

[32] L. Luo, K.-T. Foerster, S. Schmid, and H. Yu, "Splitcast: Optimizing multicast flows in reconfigurable datacenter networks," in *IEEE INFOCOM*, 2020.

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

[34] K.-T. Foerster and S. Schmid, "Survey of reconfigurable data center networks: Enablers, algorithms, complexity," *SIGACT News*, vol. 50, no. 2, pp. 62–79, 2019.

[35] "The path to 100g (fujitsu network communications)," http://www.fujitsu.com/downloads/TEL/fnc/whitepapers/Path-to-100G.pdf, online: accessed 15-April-2019.

[36] "White paper: Next-generation roadm architectures and benefits," https://www.fujitsu.com/us/Images/Fujitsu-NG-ROADM.pdf, online: accessed 15-April-2019.

[37] Y. Sheng, Y. Zhang, H. Guo *et al.*, "Benefits of unidirectional design based on decoupled transmitters and receivers in tackling traffic asymmetry for elastic optical networks," *J. Opt. Commun. Netw.*, vol. 10, no. 8, pp. C1–C14, Aug 2018.

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

[39] M. Reitblatt, N. Foster, J. Rexford *et al.*, "Abstractions for network update," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 323–334, 2012.

[40] K.-T. Foerster, S. Schmid, and S. Vissicchio, "Survey of consistent software-defined network updates," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1435–1461, 2018.

[41] M. A. Bonuccelli and M. C. Clò, "Scheduling of real-time messages in optical broadcast-and-select networks," *IEEE/ACM Trans. Netw.*, vol. 9, no. 5, pp. 541–552, 2001.

[42] "Global data centers," https://www.equinix.com/locations/, online: accessed 15-April-2019.

[43] "Mosek." https://www.mosek.com/, online: accessed 15-April-2019.

[4] https://github.com/ilongluo/DaRTree.git