DaRTree: Deadline-Aware Multicast Transfers in Reconfigurable Wide-Area Networks

Long Luo[*‡] Klaus-Tycho Foerster[‡] Stefan Schmid[‡] Hongfang Yu[*]

[*]University of Electronic Science and Technology of China, P.R. China [‡]Faculty of Computer Science, University of Vienna, Austria

ABSTRACT

The increasing amount of data replication across datacenters introduces a need for efficient bulk data transfer protocols which meet QoS guarantees, notably timely completion. We present *DaRTree* which leverages emerging optical reconfiguration technologies, to jointly optimize topology and multicast transfers, and thereby maximize throughput and acceptance ratio of transfer requests subject to deadlines. *DaRTree* is based on a novel integer linear program relaxation and deterministic rounding scheme. To this end, *DaRTree* uses multicast Steiner trees and adaptive routing based on the current network load. *DaRTree* provides its guarantees without need for rescheduling or preemption. Our evaluations show that *DaRTree* increases the network throughput and the number of accepted requests by up to 70%, especially for larger Wide-Area Networks (WANs). In fact, we also find that *DaRTree* even outperforms state-of-the-art solutions when the network scheduler is only capable of routing unicast transfers or when the WAN topology is bound to be non-reconfigurable.

CCS CONCEPTS

• **Networks → Network algorithms, performance evaluation**.

KEYWORDS

multicast transfers, deadline, reconfigurable networks

1 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 [7]. Datacenters hence become a critical infrastructure of our digital society. This also introduces increasingly stringent availability and dependability requirements, which in turn require data replication across multiple datacenters. Such replication can result in bulk transfers ranging in sizes from terabytes to petabytes [14, 15, 20, 29, 34].

Such replication-related bulk transfers are often *one-to-many*. E.g., 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 data to many destination datacenters that rely on such replica to run services. Another characteristic of such multicast transfers is their timely completion time constraints, according to Service Level Agreements (SLAs) and *Quality of Service* (QoS) requirements [16, 24, 28]. Indeed, a majority of bulk transfers have hard deadlines for completion times [16, 34], but are not very sensitive to delay nor bandwidth. In fact, a recent survey of Wide-Area Network (WAN) customers at Microsoft showed that 88% of them incur penalties on missed deadlines [16].

In order to guarantee QoS requirements and maximize network utilization, network operators employ innovative traffic engineering mechanisms (e.g., facilitated by Software-Defined Networking (SDN) [14, 15]), as well as admission control mechanisms [24, 34]. Notwithstanding, most bulk-transfer approaches today are limited to unicast transfers (e.g., *Amoeba* [34]), with only few multicast proposals surfacing [17, 25, 26]. As the latter approaches only use a single forwarding tree or are load-oblivious, selecting multiple adaptive trees could greatly improve performance.

Also, an interesting new opportunity for optimizing bulk transfers is currently emerging, related to *optics*: the physical layer workhorse of networking. In contrast to the innovations mentioned above which are software-based, optical technologies allow to optimize also the *physical layer* [8, 13, 18, 19, 31]. Recent optical WAN technology allows to reconfigure the network topology by flexibly and rapidly *shifting* wavelengths across fibers. Wavelengths, the vehicle to send data across fibers, hence become *reconfigurable*. In turn, this enables *demand-aware* [5] network topologies, which adjust the network's capacity to current traffic demands [18, 19].

However, today we do not have a good understanding of how to exploit such technologies toward efficient bulk data transfers. While recent work highlights the potential of reconfigurations, these solutions are still limited to unicast transfers [18, 19], and hence are not well suited for multicast transfers.

Contributions. In this paper, we initiate the study of how to jointly optimize one-to-many bulk transfers subject to strict deadlines, leveraging both multicasting and reconfigurable topologies in our *DaRTree*[1] approach. *DaRTree* is based on a deterministic mixed integer linear programming (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 even with only one of these optimizations:

- Even under a unicast workload, *DaRTree* outperforms prior work such as *Owan* [19] (which is based on local search

IWQoS '19, June 24–25, 2019, Phoenix, AZ, USA
© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-6778-3/19/06…$15.00
https://doi.org/10.1145/3326285.3329063

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

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.
- Moreover, *DaRTree* does not rely on rescheduling or preemption, and always guarantees deadlines.
- 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 70%, in particular for larger real-world topologies.

Motivating Example: Consider the four node WAN in Fig. 1, where we initially have one wavelength (black edge) connecting s and v, two wavelengths between v and d_1, d_2, respectively, and three between d_1 and d_2. Assume that there is a data transfer request from s to two destinations 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* [34], 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 [19]. In our approach *DaRTree*, we will combine both multicasting and reconfiguration, as in Plan D in Fig. 1(e). The objective is to improve the network throughput and accept more requests with tight deadlines. As seen in Fig. 1(f), only *DaRTree* can meet a deadline of 0.5, the other approaches need deadlines of 1 to 2.

2 BACKGROUND AND PRELIMINARIES

In this section, 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 [18, 19]. We then provide an overview of our problem setting, formalized in §3.

Background on reconfigurable WANs. Our work is focused on multicast bulk transfers in WANs connecting multiple DCNs, empowered by SDN to centrally control the networking devices and equipment. However, it can also directly be applied by ISPs that offer bulk transfer services to clients[19]. 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 bandwidth 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 [1]. Newer technologies can support e.g. 88 or more wavelengths using dense WDM, at higher data rates of 40/100 Gbps [1, 2].

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 [19]. 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 [30]. Previous work highlighted the potential of reconfigurable WANs, but so far focused on (single-hop [18])

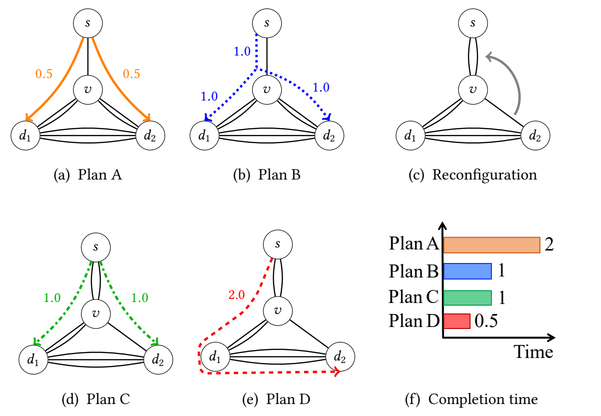


Figure 1: Example for the power of multicast transfers and topology reconfiguration. Initially, the wavelengths (black edges) are configured as shown in Fig. 1(a), where each wavelength connecting two nodes can carry 1 unit of traffic per second. When the node s wants to replicate a volume of 1 unit data 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 second with Plan C using unicast transfers (Fig. 1(d)) and just 0.5 seconds according to Plan D with multicast transfers (Fig. 1(e)).

unicast bulk transfers [19]. We go beyond these works by incorporating multicast transfers and by 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 $v_i \in V$ can send $C_i^s \in \mathbb{N}$ and receive $C_i^r \in \mathbb{N}$ 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 opposite directions: a link $l \in L$ from v_i to v_j on e can be assigned at most $\min\{C_i^s, C_j^s, C_e\}$ wavelengths.

Problem overview. In this paper, we aim to maximize the number of multicast data transfers that satisfy their deadlines by jointly optimizing the network topology together with the routing and bandwidth allocation dynamically. We assume all the requests have a strict deadline on the completion time of their data transfers and have the same priority. An extension to different priorities is straightforward hierarchical schemes can be applied. Moreover, requests appear online at discrete timeslots and cannot be aborted/modified once initiated (no preemption) in order to prevent thrashing. Note that maximizing the number of transfers to be admitted under deadlines is NP-hard [6], already for fixed topologies.

3 OFFLINE PROBLEM FORMULATION

Although we focus on the online data 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 an efficient online scheme. Note that in the offline case, all submitted multicast data transfer requests $\mathcal{R}^{\text{all}}$ are known a priori. The key notations are presented in Table 1.

Maximizing the number of successful transfers. The objective is to maximize the number of data transfers that can finish the data transmission 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 assigning the wavelengths to inter-datacenter links. Let integer variable $g_{l,t} \geq 0$ denote the number of wavelengths assigned to link l at timeslot 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 nodes and edges, respectively. Inequalities (5) enforce the valid values of variables $g_{l,t}, \forall(l,t)$.

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

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

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

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

The indicator $I(l \in L_{i,out})$ denotes whether the directed link l is an outgoing link connection of node i, and $L_{i,out}$ denotes a set of the outgoing links that connect to node i.

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

Allocating the transmission rate. We assume that each wavelength carries a bandwidth 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 §4.2. 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 tree κ of request R at time t. Inequality (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 $\boldsymbol{x}$.

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

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

Table 1: 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
$C^{s}_{i,t}$	the number of wavelengths that node $i \in V$ can send at time t
$C^{r}_{i,t}$	the number of wavelengths that node $i \in V$ can receive at time t
$C_{e,t}$	the number of wavelengths that edge $e \in E$ can carry at time t
c	the bandwidth capacity carried with per wavelength
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 to the receivers
$\mathcal{R}^{\text{all}}$	the set of all transfer requests
Internal and decision variables	
$g_{l,t}$	the number of wavelengths assigned to link l at time t
$x_{R,\kappa}$	the transmission rate on forwarding tree κ for request R during its lifetime
$x_{R,\kappa,t}$	the transmission rate on forwarding tree κ for request R at time t
z_R	binary, whether request R can be completed before its deadline

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

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

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

4 ONLINE TRANSFER ALLOCATION AND TOPOLOGY RECONFIGURATION

We now present *DaRTree*, an online approach that completes a maximum number of transfers before their deadlines by well-designed resource allocation and topology reconfiguration. We first give an overview, then describe the adaptive routing component, and finally the wavelength and rate allocation.

4.1 Overview of *DaRTree*

DaRTree relies on the following main ideas:

(1) When a new batch of requests arrives, we compute a set of k Steiner trees for each transfer. This computation is separated from the wavelength allocation part, to speed up the completion 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, 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 for the current transfer requests stay fixed: they may not be modified or preempted in future timeslots.

4.2 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 bandwidth and transfer load. In the following, we describe how we adapt link weights and then give details for the Steiner tree computation.

Link weight adaption. We initialize the link weight to be the reciprocal of the leftover bandwidth. For every link $l \in L$, we set the initial link weight w_l to $\frac{1}{c_l}$, where c_l is the amount of bandwidth that is not used by the admitted data transfers.

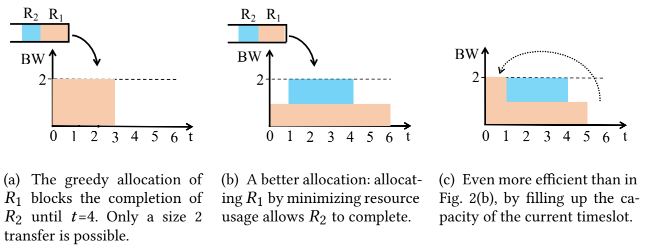
The remaining bandwidth c_l of a link l consists of two parts. The first part is the residual bandwidth of the total bandwidth capacities of the assigned wavelengths minus the bandwidth reserved for previously admitted transfers $\mathcal{R}'$. Let $c^{\text{res}}_{l,t}$ and $g'_{l,t}$ respectively denote such residual bandwidth and the number of assigned wavelengths on link l at time t. Then we can calculate $c^{\text{res}}_{l,t}$ by $cg'_{l,t} - \sum_{R \in \mathcal{R}', \kappa \in \mathcal{K}_R, t} x_{R,\kappa,t} I(l \in \kappa)$. The second part is the bandwidth potential of the yet unassigned wavelengths. If link l is from node v_i to v_j on edge e, we can calculate the maximum number of wavelengths that can be assigned to it by $\min(\overline{C}^s_{i,t}, \overline{C}^r_{j,t}, \overline{C}_{e,t})$, where $\overline{C}^s_{i,t}$, $\overline{C}^r_{j,t}$, $\overline{C}_{e,t}$ denote the number of unassigned wavelengths node i can send, node j can receive, and edge e can carry at time t, respectively. So, the total potential capacities $c^{\text{free}}_{l,t}$ of unassigned wavelengths is $c \times \min(\overline{C}^s_{i,t}, \overline{C}^r_{j,t}, \overline{C}_{e,t})$ for link l at time t. We thus calculate the amount c_l of leftover bandwidth 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 change w_l to $w_l + 1$ if link l 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. We may find the same trees for transfers in some extreme cases, e.g., where we have large k in a sparse network. However, this would not cause any interference to the execution of *DaRTree*.

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 large bandwidth is preferable over two links with small remaining capacity.

4.3 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.



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

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

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 an intuition in a small example.

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 in timeslot 0, with a deadline and size of 6, whereas R_2 appears after in timeslot 1, with a size of 3 and a deadline of 4. Fig. 2(a) shows how to greedily allocate request R_1, minimizing its completion time by assigning it the complete bandwidth of 2 for the next 3 timeslots. However, when request R_2 arrives, R_1 already blocks nearly all resources, allowing only a single timeslot with bandwidth 2, not enough to satisfy R_2. On the other hand, 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 also keep the number of needed wavelengths small, in order to freely allocate them in the next timeslots. 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 MILP $P(\eta, \varepsilon)$ = {(5), (8), (11)-(19)}, where (11) is the objective function and (5), (8), (12)-(19) are constraints (Line 4, Alg. 1).

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

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

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

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

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

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

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

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

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

Observe that $P(\eta, \varepsilon)$ has two different optimization objectives, which are minimizing the weighted assigned number η of wavelengths across links and timeslots, and maximizing the number ε of deadline-satisfied data transfers from a set $\mathcal{R}^{\text{cur}}$ that includes all processing transfers at current time.

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 timeslot 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.

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 13, Algorithm 1). Ideally, we want to complete all requests before their deadline—however, the optimization problem may be infeasible if the remaining bandwidth resources are 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. 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 a far deadline in networks with many links. We thus resort to a LP relaxation 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

$$g_{l,t} \ge 0\ . \tag{20}$$

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

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 used by the forwarding trees of requests that arrived in the current timeslot.

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 bandwidth 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 forwarding 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 $\varepsilon$;
5:      if feasible, i.e., solution exists then
6:          Admit $\varepsilon$ new transfers;
7:          $g_{l,t} = \lceil g_{l,t} \rceil, \forall l, t$;
8:          $\boldsymbol{c}^{\text{res}} \leftarrow$ UpdateResidualCapacity($\boldsymbol{c}^{\text{res}}, \boldsymbol{g}, \boldsymbol{z}, \boldsymbol{x}$);
9:          return Admission decisions of transfer requests ($\boldsymbol{z}$) and rate allocation $\boldsymbol{x}$ of admitted transfers.
10:     else
11:         Decrease $\varepsilon$ and set it to be $\varepsilon - 1$;
12:     end if
13: end while
14: return Reject current submitted transfers $\mathcal{R}^{\text{cur}}$.
15: function UpdateResidualCapacity($\boldsymbol{c}^{\text{res}}, \boldsymbol{g}, \boldsymbol{z}, \boldsymbol{x}$)
16:     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
17:         $c^{\text{res}}_{l,t} = c^{\text{res}}_{l,t} + g_{l,t}$
                $- \sum_{R \in \mathcal{R}^{\text{cur}}} \sum_{\kappa \in \mathcal{K}_R} z_R x_{R,\kappa} I(l \in \kappa) I'(t \in [t^{\text{arr}}_R, t^{\text{dl}}_R])$;
18:     end for
19:     Fill up the current timeslot with the traffic of current requests $\mathcal{R}^{\text{cur}}$ allocated in future timeslots;
20: end function
```

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

Updating the residual link capacity. After obtaining the solution of the wavelength assignment $\{g_{l,t} \forall l, t\}$, the rate allocation $\{x_{R,\kappa}, \forall R \in \mathcal{R}^{\text{cur}}, \kappa \in \mathcal{K}_R\}$, and the transfer acceptance status $\{z_R, \forall R \in \mathcal{R}^{\text{cur}}\}$, it is easy to update the residual capacities $\{c^{\text{res}}_{l,t}, \forall l, t\}$. For every link l, its new residual capacity is calculated by adding the capacities carried by newly assigned wavelengths, and deducting the capacities allocated to the admitted transfers (Line 17, Algorithm 1).

5 EVALUATION

We compare *DaRTree* to several state-of-the-art approaches in simulations. We study multiple different scenarios, considering different

Table 2: Topologies used in our simulations

Name	Description
Internet2 [19]	ISP network with 9 datacenters and 18 inter-DC links.
GScale [15]	Google's inter-DC WAN which has 12 datacenters and 19 inter-datacenter links.
Equnix [3]	An inter-DC WAN from Equnix, which connects 20 datacenters using 141 inter-datacenter links.
IDN [14]	Microsoft's inter-datacenter WAN with 40 datacenters, each connected to 2-16 other datacenters.

real-world topologies. The simulation setup is described in §5.1. We show in §5.2 that *DaRTree* outperforms prior work already for unicast transfers or for staticWAN topologies. We also show that the integer-relaxation used by *DaRTree* yields efficient runtimes. A comprehensive general evaluation is then performed in §5.3.

5.1 Simulation Setup

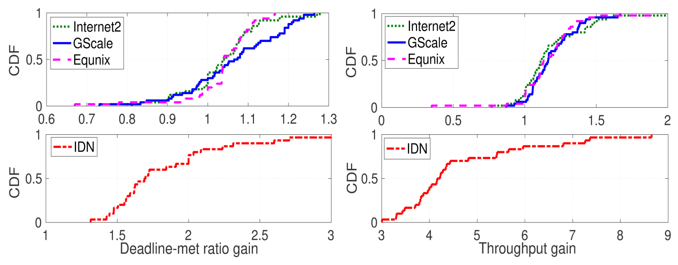
Network topologies: We run simulations over four real-world inter-datacenter networks of large cloud service providers. Table 2 shows the details about these network topologies. Following the assumptions in [34], we assign an initial uniform capacity of 160 Gbps to every link, representing the static topology configuration. Analogously to the evaluations in [19], 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 will use synthetic models to generate multicast transfer requests similar to related work [19, 20, 24, 34]. 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 $T_{span} \times \lambda$ 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 $[T, \delta T]$, 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 [4] 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 works. All the compared approaches use admission control for the data transfers submitted to the system.

- *MTree* [17] is the state-of-the-art approach that adopts k trees to optimize multicast transfers in non-reconfigurable (static) topologies.



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

Figure 3: *DaRTree* outperforms *Owan* for unicast transfers.

- *Amoeba* [34] 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* [19] also adopts k-paths to deliver data and jointly controls the network topology and transmission rates of paths to reduce the completion time or satisfy the deadline for inter-datacenter unicast transfers. More specifically, we utilize the algorithm proposed by *Owan* that optimizes the data transfers with deadlines.

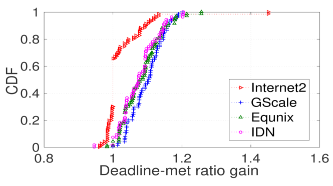
To simulate *Owan* and *Amoeba* in our setting, we split each multicast into multiple unicast transfers. For further comparisons in §5.3, we also allow fractional completion of multicast data transfers for *Owan* and *Amoeba* (e.g. to just 2 of 3 receivers), denoted as *Owan*- and *Amoeba*-Unicast, respectively.

Performance metrics. We evaluate the different approaches in a wide spectrum of performance metrics, such as deadline-met ratio of multicast (unicast) transfers, the throughput of multicast (unicast) transfers, but also allocation time. We do not allow preemption or rescheduling of admitted transfers.

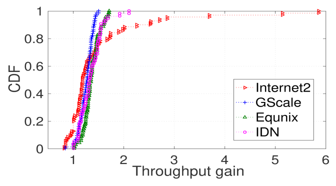
5.2 Unicast, Static WAN, and Runtime

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 requests only have one destination). 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 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



(a) Deadline-met ratio comparison



(b) Throughput comparison

Figure 4: *DaRTree* outperforms *MTree* in static WANs.

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 based 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*.

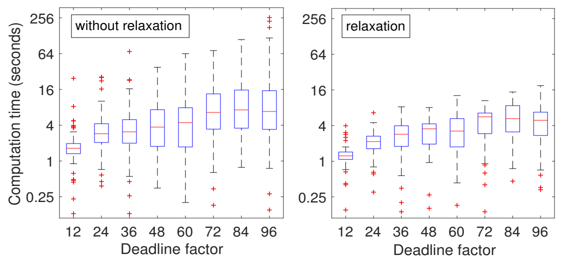
Fig. 4(a) plots the CDF of the deadline-met ratio gain of *DaRTree*. Compared to the CDFs in Fig. 3, the advantage of *DaRTree* versus *MTree* is less prevalent. However, *DaRTree* can still admit around 10% to 40% more transfers than *MTree* in 25%-55% of the experiments, with the remaining ones being 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 topology 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.

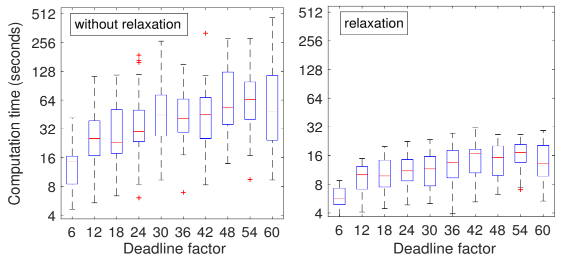
Fig. 5 plots the computation time for the Internet2 and the IDN topologies, with 40-80 allocations per algorithm and deadline factor. Figs. 5(a) and 5(b) show that the computation time is up to 250 seconds longer for the Internet2 network and up to 400 seconds longer for the IDN network, respectively.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

5.3 General 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. In the figures, *Amoeba*- and *Owan*-Unicast refer to the deadline-met receivers of (including partially allocated) requests, and *Amoeba*- and *Owan*- Multicast only refers to the requests where all the receivers meet their deadlines.



(a) Internet2



(b) IDN

Figure 5: Computation time comparison of *DaRTree* with and without relaxation. The y-axis is *logarithmic.*

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. *DaRTree* outperforms *Owan* and *Amoeba* for deadline satisfaction of multicast transfers on the

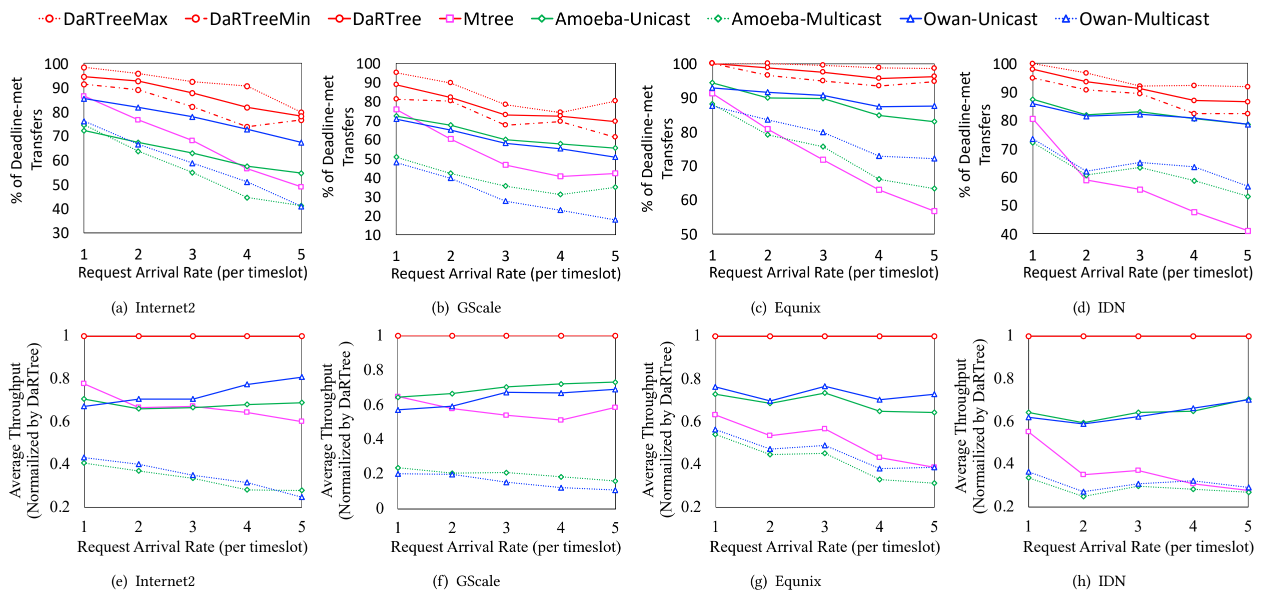


Figure 6: Impact of the request arrival rate. (a-d) show the ratio of deadline-met transfers, (e-f) show network throughput, respectively. *DaRTree*Min and *DaRTree*Max denotes minimum and maximum values of *DaRTree* over all runs, respectively. *DaRTree*, *MTree*, *Amoeba*-Unicast, *Amoeba*-Multicast, *Owan*-Unicast and *Owan*-Multicast denote average values over all runs.

Internet2 and the GScale topologies, but falls behind when the request arrival rate is $\lambda \geq 2$ on both Equnix and IDN topologies[2].

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.

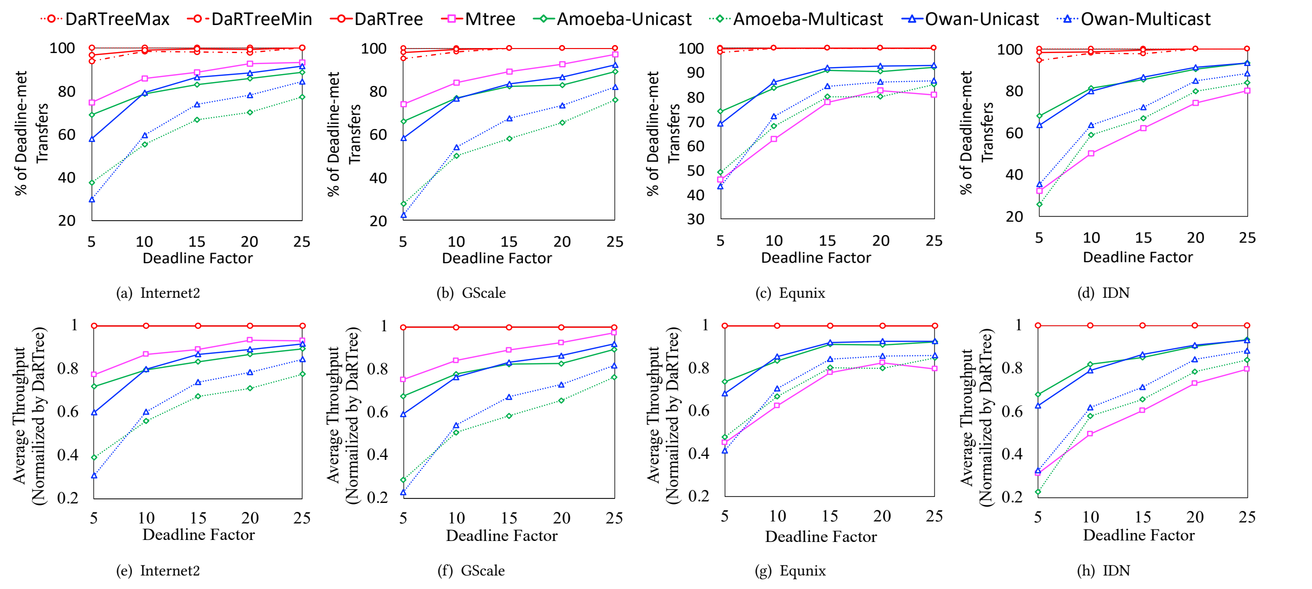
Receiver factor. In the last set of experiments, we evaluate the impact of the number of receivers for multicast transfers. To this end, we generate a constant data size for each transfer and set it to be 5TB (Internet2), 7TB (GScale), 10TB (Equinix), 14TB (IDN). Each transfer has a deadline of 6 timeslots (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× to 1.42×. In relation to *Amoeba*, *DaRTree* improves the average throughput of unicast transfers by up to 1.79× and that of multicast transfers by up to 7.37×. Lastly, for *Owan*, *DaRTree* improves the average throughput of unicast transfers by 1.16× to 2.24× and that of multicast transfers by up to 8.63×. 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 §5.2 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 §5.3, 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 70% in larger networks.

6 RELATED WORK

Traffic engineering for inter-datacenter WANs is receiving increasing attention in networking research as the number of datacenters and inter-datacenter traffic demands are growing at an unprecedented rate. Earlier work focused on network-wide objectives

[2]We note that a similar performance of *Owan* and *Amoeba* was already visible in [19, 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

Figure 7: Impact of the deadline factor. (a-d) show the deadline-met ratio, and (e-f) show the network throughput, respectively. *DaRTree*Min and *DaRTree*Max denotes the minimum and the maximum value of *DaRTree* over all runs, respectively. *DaRTree*, *MTree*, *Amoeba*-Unicast, *Amoeba*-Multicast, *Owan*-Unicast and *Owan*-Multicast denote the average values over all runs.

such as minimizing the maximum network utilization and maximizing network throughput [14, 15]. Recent work considers more fine-grained objectives, like meeting deadlines of bulk data transfers [17, 19, 20, 23, 24, 26, 34] and minimizing the completion time of large transfers [18, 29]. In this context, most work focuses on unicast transfers, with some more recent work also considering the optimization of multicast transfers.

Unicast transfers: Unicast transfers in inter-datacenter network have been the focus of much attention [18–20, 24, 34]. These works adopt k-shortest paths to deliver traffic and control the transmission rate along these paths to optimize the unicast transfers. Tempus [20] aims to allocate transfers fairly by delivering the maximum same deadline-met data fraction for all transfers. Amoeba [34] performs online admission control and focuses on guaranteeing the deadlines for a maximum number of transfers. Luo *et al.* [24] 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 [17, 23, 26, 29]. DDCCast [26] proposes to satisfy the transfer deadlines by delivering data over a forwarding tree. QuickCast [29] 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. Recently, *Iris* [27] proposes to consider mixed completion time objectives.

Luo *et al.* [23] consider k-path routing and aim to maximize the number of deadline-satisfied transfers by allowing the receivers that have already received the replica to send data to the remaining receivers. There also exists a group of works [9, 21, 22, 32, 33, 35] that adopt a store-and-forward mechanism to optimize bulk inter-datacenter 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 non-reconfigurable networks.

Reconfigurable networks: The power of dynamic inter-datacenter WAN reconfiguration was recently showcased by *Owan* [18, 19]. In order to satisfy deadlines for unicast transfers and 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. An orthogonal direction is to change the fiber bandwidth depending on the SNR [31] respectively to virtualize/provision unused fiber connections [8, 13]. Notwithstanding, there is also a wide spectrum of reconfigurable network research *inside* datacenters, e.g., optimizing path lengths [10–12].

7 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 comparative simulations for real-world topologies showed that *DaRTree* significantly improves the network throughput and the number of admitted requests over prior work. Moreover, *DaRTree* also enhances the performance of unicast transfers in reconfigurable WANs and of multicast transfers in WANs without reconfiguration.

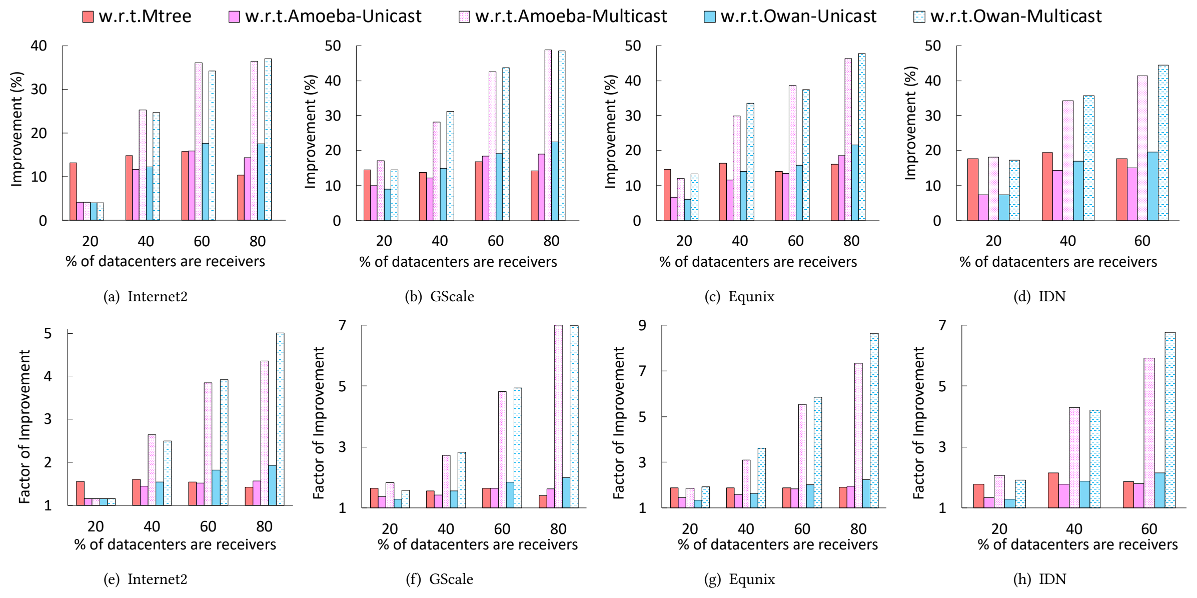


Figure 8: Impact of the receiver factor. (a-d) are improvements (in %) of transfers that meet their deadlines, (e-f) are improvements on network throughput.

We believe that our work opens several interesting avenues for future research. In particular, it will be interesting to explore the potential benefits of allowing rescheduling or preemption.

ACKNOWLEDGMENTS

We would like to thank the authors of [19] for providing us with their source code. We would also like to thank Mohammad Noormohammadpour and the anonymous reviewers for their helpful comments. This work was partially supported by the National Natural Science Foundation of China (61571098); the 111 Project (B14039); and the China Scholarship Council (201806070026).

REFERENCES

[1] 2011. The path to 100G (Fujitsu Network Communications). http://www.fujitsu.com/downloads/TEL/fnc/whitepapers/Path-to-100G.pdf.
[2] 2015. White Paper: Next-Generation ROADM Architectures and Benefits. https://www.fujitsu.com/us/Images/Fujitsu-NG-ROADM.pdf.
[3] 2019. Global Data Centers. https://www.equinix.com/locations/.
[4] 2019. MOSEK. https://www.mosek.com/.
[5] Chen Avin and Stefan Schmid. 2018. Toward demand-aware networking: a theory for self-adjusting networks. *Computer Communication Review* 48, 5 (2018), 31–40.
[6] M. A. Bonuccelli and M. C. Clò. 2001. Scheduling of real-time messages in optical broadcast-and-select networks. *IEEE/ACM Trans. Netw.* 9, 5 (2001), 541–552.
[7] Cisco. 2018. Cisco Global Cloud Index: Forecast and Methodology, 2016–2021.
[8] Ramakrishnan Durairajan et al. 2018. GreyFiber: A System for Providing Flexible Access to Wide-Area Connectivity. *arXiv:1807.05242* (2018).
[9] Yuan Feng, Baochun Li, and Bo Li. 2012. Postcard: Minimizing costs on inter-datacenter traffic with store-and-forward. In *IEEE 32nd International Conference on Distributed Computing Systems Workshops.* 43–50.
[10] T. Fenz, K.-T. Foerster, S. Schmid, and A. Villedieu. 2019. Efficient Non-Segregated Routing for Reconfigurable Demand-Aware Networks. In *IFIP Networking.*
[11] K.-T. Foerster, M. Ghobadi, and S. Schmid. 2018. Characterizing the algorithmic complexity of reconfigurable data center architectures. In *ANCS.*
[12] K.-T. Foerster, M. Pacut, and S. Schmid. 2019. On the Complexity of Non-Segregated Routing in Reconfigurable Data Center Architectures. *ACM SIGCOMM Computer Communication Review* 49, 2 (2019).
[13] Matt Hall, Vijay Chidambaram, and Ramakrishnan Durairajan. 2018. vFiber: Virtualizing Unused Optical Fibers (Extended Abstract). In *NSDI.*
[14] Chi-Yao Hong, Srikanth Kandula, et al. 2013. Achieving high utilization with software-driven WAN. In *SIGCOMM.*
[15] S. Jain et al. 2013. B4: Experience with a globally-deployed software defined WAN. In *SIGCOMM.*
[16] Virajith Jalaparti et al. 2016. Dynamic pricing and traffic engineering for timely inter-datacenter transfers. In *SIGCOMM.*
[17] S. Ji et al. 2018. Deadline-Aware Scheduling and Routing for Inter-Datacenter Multicast Transfers. In *IEEE International Conference on Cloud Engineering.*
[18] Su Jia et al. 2017. Competitive analysis for online scheduling in software-defined optical WAN. In *IEEE INFOCOM.*
[19] Xin Jin et al. 2016. Optimizing bulk transfers with software-defined optical WAN. In *SIGCOMM.*
[20] Srikanth Kandula, Ishai Menache, Roy Schwartz, et al. 2014. Calendaring for wide area networks. In *SIGCOMM.*
[21] Nikolaos Laoutaris et al. 2013. Delay-tolerant bulk data transfers on the Internet. *IEEE/ACM Trans. Netw.* 21, 6 (2013), 1852–1865.
[22] Nikolaos Laoutaris, Michael Sirivianos, Xiaoyuan Yang, et al. 2011. Inter-datacenter bulk transfers with netstitcher. In *SIGCOMM*, Vol. 41. 74–85.
[23] Long Luo, Hongfang Yu, and Zilong Ye. 2018. Deadline-guaranteed Point-to-Multipoint Bulk Transfers in Inter-Datacenter Networks. In *IEEE ICC.*
[24] Long Luo, Hongfang Yu, Zilong Ye, and Xiaojiang Du. 2018. Online Deadline-aware Bulk Transfer over Inter-Datacenter WANs. In *IEEE INFOCOM.*
[25] M. Noormohammadpour et al. 2017. DCCast: Efficient Point to Multipoint Transfers Across Datacenters. In *HotCloud.*
[26] M. Noormohammadpour et al. 2017. DDCCast: Meeting Point to Multipoint Transfer Deadlines Across Datacenters using ALAP Scheduling Policy. *arXiv:1707.02027* (2017).
[27] M. Noormohammadpour et al. 2019. Efficient Inter-Datacenter Bulk Transfers with Mixed Completion Time Objectives. *arXiv:1905.01749v1* (2019).
[28] M. Noormohammadpour and C. S. Raghavendra. 2018. Datacenter traffic control: Understanding techniques and tradeoffs. *IEEE Communications Surveys & Tutorials* 20, 2 (2018), 1492–1525.
[29] M. Noormohammadpour, C. S. Raghavendra, S. Kandula, and S. Rao. 2018. QuickCast: Fast and Efficient Inter-Datacenter Transfers Using Forwarding Tree Cohorts. In *IEEE INFOCOM.* 225–233.
[30] Yang Sheng et al. 2018. Benefits of Unidirectional Design Based on Decoupled Transmitters and Receivers in Tackling Traffic Asymmetry for Elastic Optical Networks. *J. Opt. Commun. Netw.* 10, 8 (Aug 2018), C1–C14.
[31] Rachee Singh, Manya Ghobadi, Klaus-Tycho Foerster, Mark Filer, and Phillipa Gill. 2018. RADWAN: rate adaptive wide area network. In *SIGCOMM.*
[32] Yiwen Wang et al. 2014. Multiple bulk data transfers scheduling among datacenters. *Computer Networks* 68 (2014), 123–137.
[33] Yu Wu et al. 2017. Orchestrating bulk data transfers across geo-distributed datacenters. *Trans. on Cloud Comput.* 5, 1 (2017), 112–125.
[34] Hong Zhang et al. 2017. Guaranteeing Deadlines for Inter-Data Center Transfers. *IEEE/ACM Trans. Netw.* 25(1) (2017), 579–595.
[35] Yuchao Zhang et al. 2018. BDS: a centralized near-optimal overlay network for inter-datacenter data replication. In *EuroSys.*