DCCast: Efficient Point to Multipoint Transfers Across Datacenters

Mohammad Noormohammadpour[†], Cauligi S. Raghavendra[†], Sriram Rao[‡], Srikanth Kandula[‡]

[†]*Ming Hsieh Department of Electrical Engineering, University of Southern California*
[‡]*Microsoft*

Abstract

Using multiple datacenters allows for higher availability, load balancing and reduced latency to customers of cloud services. To distribute multiple copies of data, cloud providers depend on inter-datacenter WANs that ought to be used efficiently considering their limited capacity and the ever-increasing data demands. In this paper, we focus on applications that transfer objects from one datacenter to several datacenters over dedicated inter-datacenter networks. We present DCCast, a centralized Point to Multi-Point (P2MP) algorithm that uses forwarding trees to efficiently deliver an object from a source datacenter to required destination datacenters. With low computational overhead, DCCast selects forwarding trees that minimize bandwidth usage and balance load across all links. With simulation experiments on Google's GScale network, we show that DCCast can reduce total bandwidth usage and tail Transfer Completion Times (TCT) by up to 50% compared to delivering the same objects via independent point-to-point (P2P) transfers.

1 Introduction

Increasingly, companies rely on multiple datacenters to improve quality of experience for their customers. The benefits of having multiple datacenters include reduced latency to customers by mapping users to datacenters according to location, increased failure resiliency and availability, load balancing by mapping users to different datacenters and respecting local data laws. Companies may either own the datacenters or depend on infrastructure and services from providers such as Microsoft Azure [1], Google Compute Engine [2] or Amazon Web Services [3]. Large providers use geographically distributed dedicated networks to connect their datacenters [4–7].

Nowadays, many services operate across multiple datacenters. Such services require efficient data transfers among datacenters including replication of objects from one datacenter to multiple datacenters which is referred to as geo-replication [4, 7–17]. Examples of such transfers include synchronizing search index information [7], replication of databases [18, 19], and distribution of high definition videos across CDNs [17, 20–23]. Table 1 provides a brief list of how many replicas are made for some applications.

Service	Replicas
Facebook	Across availability regions [24], ≥ 4 [25], for various object types including large machine learning configs [26]
CloudBasic SQL Server	Up to 4 secondary databases with active Geo-Replication (asynchronous) [27]
Azure SQL Database	Up to 4 secondary databases with active Geo-Replication (asynchronous) [28]
Oracle Directory Server	Up to the number of datacenters owned by an enterprise for regional load balancing of directory servers [29, 30]
AWS Route 53 GLB	Across multiple regions and availability zones for global load balancing [31]
Youtube	Function of popularity, content potentially pushed to many locations (could be across ≥ 33 datacenters [32])
Netflix	Across 2 to 4 availability regions [33], and up to 233 cache locations [34]

Table 1: Various Services Using Replication

In this paper, we focus on transfers that deliver an object from a source to multiple destinations and call them **Point to Multipoint (P2MP)** transfers. One solution to make such transfers is to initiate multiple independent point-to-point (P2P) transfers that are scheduled separately [4, 5, 14, 22, 35–42]. There may however be more efficient ways, in terms of total bandwidth usage and transfer completion times, to perform P2MP transfers by sending at most one copy of the message across any link given that the source datacenter and destination datacenters are known apriori. We present an elegant solution using minimum weight Steiner Trees [43] for

P2MP transfers that achieves reduced bandwidth usage and tail completion times.

Another approach would be to select trees that connect from sources to all destinations and complete transfers using store-and-forward [14, 35–38]. This will impose the overhead of transferring and storing additional copies of objects on intermediate datacenters during the delivery process incurring storage costs as the number of transfers increase, and wasting intra-datacenter bandwidth since such objects have to be stored on a server inside the intermediate datacenters.

Alternatively one can use multicast protocols, which are designed to form group memberships, provide support for members to join or leave, and manage multicast trees as users come and go [44, 45]. These approaches involve complex management algorithms and protocols to cope with changes in multicast group membership [46,47] which are unnecessary for our problem given that the participants of each P2MP transfer are fixed.

Application layer multicast techniques [46, 48] reduce the implementation and management complexity by creating an overlay network across the multicast group. However, since these methods are typically implemented in end-host applications, they lack full visibility to underlying network properties and status, such as topology and available bandwidth, and may still send multiple copies of packets along the same links wasting bandwidth.

How can one efficiently perform P2MP transfers? Objective is minimizing tail Transfer Completion Times (TCT) considering limited available bandwidth of networks connecting datacenters and the many transfers that share the network which arrive in an online manner. Also, the system does not have prior knowledge of incoming P2MP transfers; therefore, any solution has to be fast and efficient to deal with them as they arrive.

To perform a P2MP transfer, traffic can be concurrently sent to all destinations over a minimum weight Steiner Tree that connects the source and destinations with transfers' demands as link weights. We refer to such trees as forwarding trees using which we can reduce total bandwidth usage. A central controller with global view of network topology and distribution of load [5–7,49–51] can carefully weigh out various options and select forwarding trees for transfers.

We can implement forwarding trees using SDN [52] capable switches that support Group Tables [53,54] such as [55–58]. Using this feature, a packet can be replicated and supplied to multiple buckets each processing and forwarding it to a required output switch port. A group might have many buckets, for example one vendor supports up to 32 [54]. There is growing vendor support for group tables as later OpenFlow standards are adopted. In this paper, we use abstract simulations to verify our techniques. In the future, we plan on implementing forwarding trees using Group Tables.

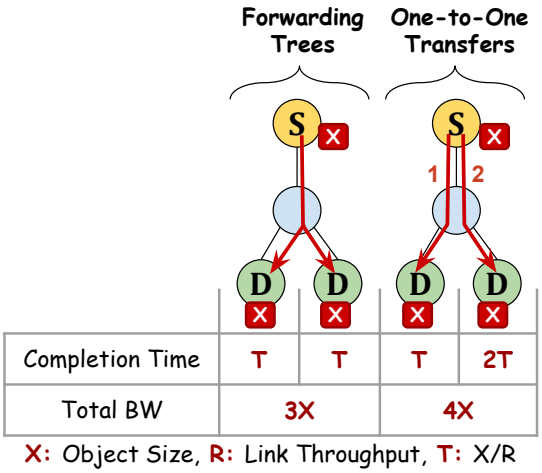

Figure 1: Benefits of using Forwarding Trees

Motivating Example: In Figure 1, an object X is to be transferred from datacenter S to two D datacenters considering a link throughput of R. In order to send X to destinations, one could initiate individual transfers, but that wastes bandwidth and increases delivery time since the link attached to S turns into a bottleneck.

In this paper, we present an efficient scheme for P2MP transfers called **DCCast**. It selects forwarding trees according to a weight assignment that tries to balance load across the network. In addition, it uses temporal planning [5] and schedules P2MP requests on a first come first serve (FCFS) basis to provide guarantees on completion times. FCFS is shown to reduce tail completion times under light-tailed job size distribution [59,60] (optimal discipline depends on this distribution [60]).

A related concept is Coflows [61] that improve performance by jointly scheduling groups of flows with a collective objective. A P2MP transfer can be viewed as a coflow; however unlike other coflows, P2MP transfers present a better opportunity for optimization since the same data is being delivered to different destinations. Using this, DCCast provides the added benefit of reduced bandwidth usage.

We conducted extensive simulation experiments to evaluate DCCast compared with other strategies for picking forwarding trees and scheduling techniques. We performed simulations using synthetic traffic over the Google's GScale topology [7] with 12 nodes and 19 edges and random topologies with 50 nodes and 150 edges. Our evaluation metrics include bandwidth usage as well as mean and tail TCT. Our current solution assumes the same class of service for all transfers. In a general setting, transfers may have different priorities and should be allotted resources accordingly (e.g. near real-time video vs. cross-region backups).

We evaluated various forwarding tree selection meth-

ods. Clearly, there is benefit in carefully picking trees and we observed up to 43% improvement in completion times while using DCCast compared to random tree selection and up to 29% compared to selection of trees that minimize the maximum load over any edge.

We compared DCCast with P2P schemes by viewing each P2MP transfer as multiple independent transfers and using multipathing (K shortest paths) to spread the load. For GScale topology and with 2 to 6 destinations per transfer, DCCast reduced both bandwidth usage and tail TCT by over 20% to 50%, respectively, while providing guarantees to users on completion times.

In summary, our contributions are the following:

- Prior work based on traffic scheduling and rate allocation [5, 6, 49] uses individual point to point transfers to deliver the same object to multiple places. We improve on this by using forwarding trees.
- Prior work on multicasting [44–48] is focused on managing multicast groups. With apriori knowledge of transfer destinations and demands, DCCast builds forwarding trees that are more efficient.
- DCCast minimizes packet reordering and provides guarantees on completion times.

2 Problem Formulation

The list of variables and their definitions is provided on table 2. To allow for flexible bandwidth allocation, we consider a slotted timeline [5, 41, 42] where the transmission rate of senders is constant during each timeslot, but can vary from one timeslot to next. This can be achieved via rate-limiting at end-hosts [6, 49]. A central scheduler is assumed that receives transfer requests from end-points, calculates their temporal schedule, and informs the end-points of rate-allocations when a timeslot begins. We focus on scheduling large transfers that take more than a few timeslots to finish and therefore, the time to submit a transfer request, calculate the routes, and install forwarding rules is considered negligible in comparison. We assume equal capacity for all links in an online scenario where requests may arrive anytime.

3 DCCast

Forwarding Trees: Our proposed approach is, for each P2MP transfer, to jointly route traffic from source to all destinations over a forwarding tree to save bandwidth. Using a single forwarding tree for every transfer also minimizes packet reordering which is known to waste CPU and memory resources at the receiving ends especially at high rates [62, 63]. To perform a P2MP transfer R with volume $\mathcal{V}_R$, the source S_R transmits traffic over a Steiner Tree that spans across $\mathbf{D_R}$. At any timeslot, traffic for any transfer flows with the same rate over all links of a forwarding tree to reach all the destinations at the same time. The problem of scheduling a P2MP transfer then translates to finding a forwarding tree and a transmission schedule over such a tree for every arriving transfer in an online manner. A relevant problem is the Minimum Weight Steiner Tree [43] that can help minimize total bandwidth usage with proper weight assignment. Although it is a hard problem, heuristic algorithms exist that often provide near optimal solutions [64, 65].

Variable	Definition
R	A P2MP transfer
$\mathcal{V}_R$	Volume of R in bytes
S_R	Source datacenter of R
$\mathbf{D_R}$	Set$\langle\rangle$ of destinations of R
G	The inter-datacenter network graph
T	A Steiner Tree [43]
$\mathbf{E_G}$	Set$\langle\rangle$ of edges of G
$\mathbf{E_T}$	Set$\langle\rangle$ of edges of T
L_e	Total load currently scheduled on edge e
$B_e(t)$	Available bandwidth on edge e at timeslot t
W	Width of a timeslot in seconds

Table 2: Definition of Variables

Scheduling Discipline: When forwarding trees are found, we schedule traffic over them according to First Come First Serve (FCFS) policy using all available residual bandwidth on links to minimize the completion times. This allows us to provide guarantees to users on when their transfers will complete upon their arrival. We do not use a preemptive scheme, such as Shortest Remaining Processing Time (SRPT), due to practical concerns: larger transfers might get postponed over and over which might lead to the starvation problem and it is not possible to make promises on exactly when a transfer would complete. Optimal scheduling discipline to minimize tail times rests on transfer size distribution [60].

Algorithms: DCCast is made of two algorithms[1]. **Update()** is executed upon beginning of every timeslot. It simply dispatches the transmission schedule, that is the rate for each transfer, to all senders to adjust their rates via rate-limiting and adjusts L_e ($e \in \mathbf{E_G}$) by deducting the total traffic that was sent over e during current timeslot.

Allocate(R) is run upon arrival of every request which finds a forwarding tree and schedules R to finish as early as possible. Pseudo-code of this function has been shown in Algorithm 1. Statically calculating minimal forwarding trees might lead to creation of hot-spots, even if there exists one highly loaded edge that is shared by all of such trees. As a result, DCCast adaptively chooses a forwarding tree that reduces the tail transfer completion times while saving considerable bandwidth.

[1]Please find an implementation of DCCast algorithms on Github: https://github.com/noormoha/DCCast

Algorithm 1: Allocate(R)

Input: $R(\mathcal{V}_R, S_R, \mathbf{D_R})$, G, W, L_e and $B_e(t)$ for $e \in \mathbf{E_G}$ and $t > t_{now}$

Output: Forwarding tree (minimum weight Steiner Tree) T and Transmission Schedule of R for $t > t_{now}$

To every edge $e \in \mathbf{E_G}$, assign weight $W_e = (L_e + \mathcal{V}_R)$;

Find the Minimum Weight Steiner Tree T that connects $S_R \cup \mathbf{D_R}$. We used GreedyFLAC [65, 66];

Schedule R over T to finish as early as possible.
$t' \leftarrow t_{now} + 1$ and $\mathcal{V}' \leftarrow \mathcal{V}_R$;
while $\mathcal{V}' > 0$ **do**
  $B_T \leftarrow \mathbf{min}_{e \in \mathbf{E_T}}(B_e(t'))$;
  Schedule R on T with rate $\mathbf{min}(B_T, \frac{\mathcal{V}'}{W})$ at timeslot t' ;
  $t' \leftarrow t' + 1$ and $\mathcal{V}' \leftarrow \mathcal{V}' - B_T \times W$;

return T *and the Transmission Schedule of* R;

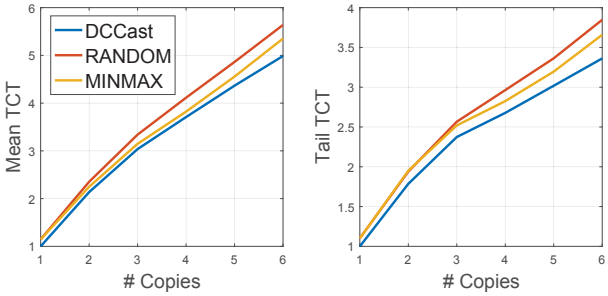

Figure 2: Tree Selection (GScale Topo)

It is possible that larger trees provide higher available bandwidth by using longer paths through least loaded edges, but using which would consume more overall bandwidth since they send same traffic over more edges. To model this behavior, we use a weight assignment that allows balancing these two possibly conflicting objectives. The weights represent traffic load allocated on links. Selecting links with lower weights will improve load balancing that would be better for future requests. The tradeoff is in avoiding heavier links at the expense of getting larger trees for even distribution of load.

The forwarding tree T selected by Algorithm 1 will have the total weight $\sum_{e \in \mathbf{E_T}}(L_e + \mathcal{V}_R)$. This weight is essentially the total load over T if request R were to be put on it. Selecting trees with minimal total weight will most likely avoid highly loaded edges and larger trees. To find an approximate minimum weight Steiner Tree, we used GreedyFLAC [65, 66], which is quite fast and in practice provides results not far from optimal.

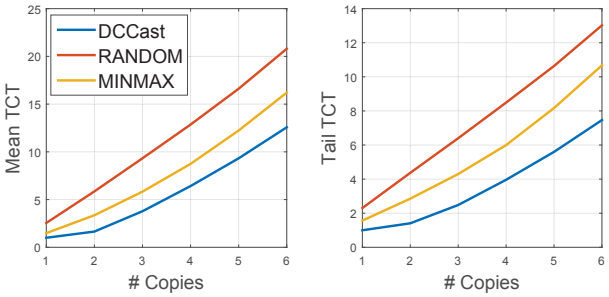

Figure 3: Tree Selection (Random Topo, $|V| = 50$)

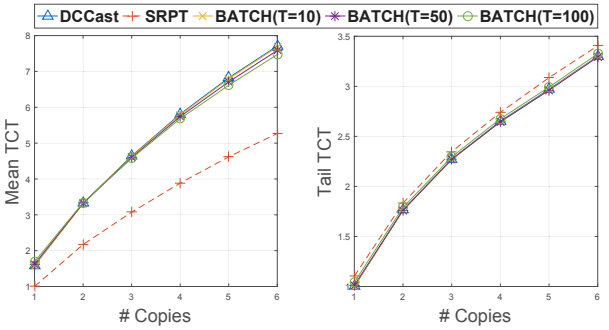

Figure 4: Various Scheduling Policies

4 Evaluation

We evaluated DCCast using synthetic traffic. We assumed a total capacity of 1.0 for each timeslot over every link. The arrival of requests followed a Poisson distribution with rate $\lambda_{P2MP} = 1$. Demand of every request R was calculated using an exponential distribution with mean 20 added to a constant value of 10 (fixing the minimum demand to 10). All simulations were performed over as many timeslots as needed to finish all requests with arrival time of last request set to be 500 or less. Presented results are normalized by minimum values in each chart.

We measure three different metrics: **total bandwidth used** and **mean and tail TCT**. The total bandwidth used is the sum of all traffic over all timeslots and all links. The completion time of a transfer is defined as its arrival time to the time its last bit is delivered to the destination(s). We performed simulations using Google's GScale topology [7], with 12 nodes and 19 edges, on a single machine (Intel Core i7-6700T and 24 GBs of RAM). All simulations were coded in Java and used Gurobi Optimizer [67] to solve linear programs for P2P schemes. We increased the destinations (copies) for each object from 1 to 6 picking recipients according to uniform distribution. Table 3 shows list of considered schemes (first 4 are P2MP schemes and last 2 are P2P).

We tried various forwarding tree selection criteria over both GScale topology and a larger random topology with 50 nodes and 150 edges as shown in Figures 2 and 3, respectively. In case of GScale, DCCast performs slightly better than RANDOM and MINMAX in com-

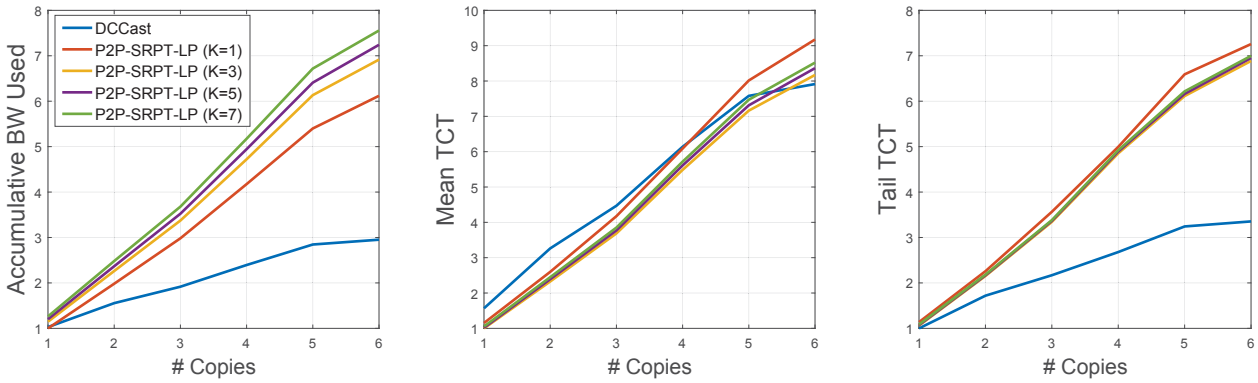

Figure 5: DCCast vs Point-To-Point (P2P-SRPT-LP)

Scheme	Method
MINMAX	Selects forwarding trees to minimize maximum load on any link. Schedules traffic using FCFS policy §3
RANDOM	Selects random forwarding trees. Schedules traffic using FCFS policy §3
BATCHING	Batches (enqueues) new requests arriving in time windows of T. At the end of batching windows, jointly schedules all new requests according to Shortest Job First (SJF) policy and picks their forwarding trees using weight assignment of Algorithm 1
SRPT	Upon arrival of a new request, jointly reschedules all existing requests and the new request according to SRPT policy §3 and picks new forwarding trees for all requests using weight assignment of Algorithm 1
P2P-SRPT-LP	Views each P2MP request as multiple independent point-to-point (P2P) requests. Uses a Linear Programming (LP) model along with SRPT policy §3 to (re)schedule each request over K-Shortest Paths between its source and destination upon arrival of new requests
P2P-FCFS-LP	Similar to above while using FCFS policy §3

Table 3: Schemes Used for Comparison

pletion times while using equal overall bandwidth (not in figure). In case of larger random topologies, DCCast's dominance is more obvious regarding completion times while using same or less bandwidth (not in figure).

We also experimented various scheduling disciplines over forwarding trees as shown in Figure 4. The SRPT discipline performs considerably better regarding mean completion times; it however may lead to starvation of larger transfers if smaller ones keep arriving. It has to compute and install new forwarding trees and recalculate the whole schedule, for all requests currently in the system with residual demands, upon arrival of every new request. This could impose significant rule installation overhead which is considered negligible in our evaluations. It might also lead to lots of packet loss and reordering. Batching improves performance marginally compared to DCCast and could be an alternate road to take. Generally, a smaller batch size results in a smaller initial scheduling latency while a larger batch size makes it possible to employ collective knowledge of many requests in a batch for optimized scheduling. Batching might be more effective for systems with bursty request arrival patterns. All schemes performed almost similarly regarding tail completion times and total bandwidth usage (not in figure).

In Figure 5 we compare DCCast with a Point-To-Point scheme (P2P-SRPT-LP) using SRPT scheduling policy which uses various number of shortest paths (K) and delivers each copy independently. The total bandwidth usage is close for all schemes when there is only one destination per request. Both bandwidth usage and tail completion times of DCCast are up to 50% less than that of P2P-SRPT-LP as the number of destinations per transfer increases. Although DCCast follows the FCFS policy, its mean completion time is close to that of P2P-SRPT-LP and surpasses it for 6 copies due to bandwidth savings which leave more headroom for new transfers.

In a different experiment, we compared DCCast with P2P-FCFS-LP. DCCast again saved up to 50% bandwidth and reduced tail completion times by up to almost 50% while increasing the number of destinations per transfer.

Computational Overhead: We used a network with 50 nodes and 300 edges and considered P2MP transfers with 5 destinations per transfer. Transfers were generated according to Poisson distribution with arrival times ranging from 0 to 1000 timeslots and the simulation ran until all transfers were completed. Mean processing time of a single *timeslot* increased from 1.2ms to 50ms per timeslot while increasing λ_{P2MP} from 1 to 10. Mean process-

ing time of a single *transfer* (which accounts for finding a tree and scheduling the transfer) was $1.2ms$ and $5ms$ per transfer for λ_{P2MP} equal to 1 and 10, respectively. This is negligible compared to timeslot lengths of minutes in prior work [41].

5 Conclusions and Future Work

In this paper, we presented DCCast, which aims to reduce the total network bandwidth usage while minimizing transfer completion times using point to multipoint (P2MP) transfers. To save bandwidth, DCCast uses forwarding trees that connect source of a P2MP transfer to its destinations. Selection of forwarding trees is performed in a way that attempts to balance load across the network by avoiding highly loaded links while reducing bandwidth usage by choosing smaller trees. To provide guarantees on transfer completion times, DCCast schedules new traffic to finish as early as possible while not changing the schedule of already allocated transfers. Our evaluations show that DCCast can significantly reduce bandwidth usage compared to viewing each P2MP transfer as multiple independent transfers.

In the future, we would like to perform testbed experiments using traces of real traffic. An alternate scheduling scheme to what we proposed would be Fair Sharing which we aim to study. Next, we would like to evaluate DCCast using a mix of P2MP transfers with different number of destinations to better understand how various applications might interact. It would be interesting to consider multiple classes of traffic with different priorities as well. Moreover, further investigation is needed to measure the fraction of traffic with multiple destinations which benefit from forwarding trees. Finally, it is necessary to study approaches for handling and recovering from failures.

6 Acknowledgments

We would like to thank our shepherd Sindhu Ghanta and the anonymous reviewers of the HotCloud workshop whose comments helped us greatly improve the quality of this paper.

7 Discussion Topics

Prior work on inter-datacenter transfers uses a combination of various techniques, such as rate-control [6, 49], temporal planning [5], store-and-forward [14], and topology changes [10], to improve the performance and efficiency of point-to-point (P2P) inter-datacenter transfers. In this paper, we focused on point-to-multipoint (P2MP) transfers and a new direction in which forwarding trees are used to deliver an object simultaneously to all destinations. We discussed the benefits of this approach and presented evaluations to back up our proposal.

The extent to which our approach can benefit operators as well as cloud and datacenter applications used in industry would be one possible discussion point. This depends partly on the type of applications, desired level of reliability, and customer SLOs such as user access latency. Finally, a crucial factor is the fraction of transfers with multiple destinations, which are ones that can actually benefit from use of forwarding trees.

There are several discussion points regarding implementation. In case of using tools, such as SDN, to cheaply setup and tear down forwarding trees, how can we meet the necessary performance metrics, such as delay, considering number of required forwarding rules, transfer arrival rate, and latency overhead of rule installations? What are possible tensions between performance efficiency and practical feasibility of techniques?

Another topic of discussion would be selection of the forwarding trees. We presented and evaluated three possible selection methods. However selection of forwarding trees that reduce usage of bandwidth while minimizing transfer completion times is an open problem. In addition, would it be better to select and setup trees dynamically, or statically build many trees and use them?

Scalability of our approach considering various network topologies, different network sizes as well as arrival rate of transfers, effectiveness of the scheduling discipline used in satisfying users and operators, and handling of network or end-point failures would be other possible discussion topics. Would it be possible to apply known methods of improving scalability if necessary?

References

[1] Microsoft azure: Cloud computing platform & services. https://azure.microsoft.com/.

[2] Compute engine - iaas - google cloud platform. https://cloud.google.com/compute/.

[3] Amazon web services (aws) - cloud computing services. https://aws.amazon.com/.

[4] Yu Wu, Zhizhong Zhang, Chuan Wu, et al. Orchestrating bulk data transfers across geo-distributed datacenters. *IEEE Transactions on Cloud Computing*, PP(99):1–1, 2015.

[5] Srikanth Kandula, Ishai Menache, Roy Schwartz, et al. Calendaring for wide area networks. *ACM SIGCOMM*, 44(4):515–526, 2015.

[6] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, et al. Achieving high utilization with software-driven wan. In *ACM SIGCOMM*, pages 15–26, 2013.

[7] Sushant Jain, Alok Kumar, Subhasree Mandal, et al. B4: Experience with a globally-deployed software defined wan. *ACM SIGCOMM*, 43(4):3–14, 2013.

[8] Ashish Gupta, Fan Yang, Jason Govig, et al. Mesa: A geo-replicated online data warehouse for google's advertising system. *Communications of the ACM*, 59(7):117–125, June 2016.

[9] Tim Kraska, Gene Pang, Michael J Franklin, Samuel Madden, and Alan Fekete. Mdcc: Multi-data center consistency. In *ACM Eurosys*, pages 113–126, 2013.

[10] Xin Jin, Yiran Li, Da Wei, Siming Li, et al. Optimizing bulk transfers with software-defined optical wan. In *ACM SIGCOMM*, pages 87–100, 2016.

[11] Xiaoxue Zhao, Vijay Vusirikala, Bikash Koley, et al. The prospect of inter-data-center optical networks. *IEEE Communications Magazine*, 51(9):32–38, 2013.

[12] Ll Gifre, F Paolucci, J Marhuenda, et al. Experimental assessment of inter-datacenter multicast connectivity for ethernet services in flexgrid networks. In *Proceedings of European Conference on Optical Communications*, pages 1–3, 2014.

[13] Yasuhiro Miyao. An overlay architecture of global inter-data center networking for fast content delivery. In *IEEE International Conference on Communications (ICC)*, pages 1–6, 2011.

[14] Nikolaos Laoutaris, Georgios Smaragdakis, Rade Stanojevic, et al. Delay-tolerant bulk data transfers on the internet. *IEEE/ACM Transactions on Networking*, 21(6):1852–1865, December 2013.

[15] Ping Lu, Liang Zhang, Xiahe Liu, et al. Highly efficient data migration and backup for big data applications in elastic optical inter-data-center networks. *IEEE Network*, 29(5):36–42, 2015.

[16] Yingying Chen, Sourabh Jain, Vijay Kumar Adhikari, et al. A first look at inter-data center traffic characteristics via yahoo! datasets. In *IEEE INFOCOM*, pages 1620–1628, 2011.

[17] Yuan Feng, Baochun Li, and Bo Li. Jetway: Minimizing costs on inter-datacenter video traffic. In *ACM International conference on Multimedia*, pages 259–268, 2012.

[18] Multi-datacenter replication in cassandra. http://www.datastax.com/dev/blog/multi-datacenter-replication, 2012.

[19] Azure sql database now supports powerful geo-replication features for all service tiers. https://azure.microsoft.com/en-us/blog/azure-sql-database-now-supports-powerful-geo-replication-features-on-all-service-tiers/, 2016.

[20] Vijay Kumar Adhikari, Sourabh Jain, Yingying Chen, and Zhi-Li Zhang. Vivisecting youtube: An active measurement study. In *IEEE INFOCOM*, pages 2521–2525, 2012.

[21] Meshenberg, Ruslan and Gopalani, Naresh and Kosewski, Luke. Active-Active for Multi-Regional Resiliency. http://techblog.netflix.com/2013/12/active-active-for-multi-regional.html, 2013.

[22] Yuhua Lin, Haiying Shen, and Liuhua Chen. Ecoflow: An economical and deadline-driven inter-datacenter video flow scheduling system. In *Proceedings of the ACM international conference on Multimedia*, pages 1059–1062, 2015.

[23] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. Inside the social network's (datacenter) network. In *ACM SIGCOMM*, pages 123–137, 2015.

[24] Companies like facebook and google have multiple data centers. do these datacenters all store copies of the same information? https://www.quora.com/Companies-like-Facebook-and-Google-have-multiple-data-centers-Do-these-datacenters-all-store-copies-of-the-same-information. visited on March 3, 2017.

[25] Where are the facebook servers located worldwide? https://www.quora.com/Where-are-the-Facebook-servers-located-worldwide. visited on March 3, 2017.

[26] Chunqiang Tang, Thawan Kooburat, Pradeep Venkatachalam, et al. Holistic configuration management at facebook. In *Proceedings of the Symposium on Operating Systems Principles*, pages 328–343. ACM, 2015.

[27] Geo-replication/multi-ar (active). http://cloudbasic.net/documentation/geo-replication-active/. visited on March 5, 2017.

[28] Overview: Sql database active geo-replication. https://docs.microsoft.com/en-us/azure/sql-database/sql-database-geo-replication-overview. visited on March 5, 2017.

[29] Using replication across multiple data centers. https://docs.oracle.com/cd/E20295_01/html/821-1217/fpcoo.html#aalgm. visited on March 11, 2017.

[30] Understanding oracle internet directory replication. https://docs.oracle.com/cd/E28280_01/admin.1111/e10029/oid_replic.htm#OIDAG2201. visited on March 11, 2017.

[31] Global load balancing using aws route 53. https://www.sumologic.com/blog-amazon-web-services/aws-route-53-global-load-balancing/. visited on March 14, 2017.

[32] Ruben Torres, Alessandro Finamore, Jin Ryong Kim, Marco Mellia, Maurizio M Munafo, and Sanjay Rao. Dissecting video server selection strategies in the youtube cdn. In *Distributed Computing Systems (ICDCS)*, pages 248–257. IEEE, 2011.

[33] How netflix leverages multiple regions to increase availability (arc305). https://www.slideshare.net/AmazonWebServices/arc305-28387146. visited on March 3, 2017.

[34] Mapping netflix: Content delivery network spans 233 sites. http://datacenterfrontier.com/mapping-netflix-content-delivery-network/. visited on March 3, 2017.

[35] Yiwen Wang, Sen Su, Alex X Liu, et al. Multiple bulk data transfers scheduling among datacenters. *Computer Networks*, 68:123–137, 2014.

[36] Yang Yu, Wang Rong, and Wang Zhijun. Ssnf: Shared datacenter mechanism for inter-datacenter bulk transfer. In *IEEE International Conference on Advanced Cloud and Big Data (CBD)*, pages 184–189, 2014.

[37] Nikolaos Laoutaris, Michael Sirivianos, Xiaoyuan Yang, and Pablo Rodriguez. Inter-datacenter bulk transfers with netstitcher. In *ACM SIGCOMM*, pages 74–85, 2011.

[38] Yuan Feng, Baochun Li, and Bo Li. Postcard: Minimizing costs on inter-datacenter traffic with store-and-forward. In *IEEE International Conference on Distributed Computing Systems Workshops*, pages 43–50, 2012.

[39] Thyaga Nandagopal and Krishna PN Puttaswamy. Lowering inter-datacenter bandwidth costs via bulk data scheduling. In *IEEE International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 244–251, 2012.

[40] Jingjing Yao, Ping Lu, Long Gong, and Zuqing Zhu. On fast and coordinated data backup in geo-distributed optical inter-datacenter networks. *IEEE Journal of Lightwave Technology*, 33(14):3005–3015, 2015.

[41] Hong Zhang, Kai Chen, Wei Bai, et al. Guaranteeing deadlines for inter-datacenter transfers. In *ACM Eurosys*, page 20, 2015.

[42] Mohammad Noormohammadpour, Cauligi S Raghavendra, and Sriram Rao. Dcroute: Speeding up inter-datacenter traffic allocation while guaranteeing deadlines. In *High Performance Computing, Data, and Analytics (HiPC)*. IEEE, 2016.

[43] FK Hwang and Dana S Richards. Steiner tree problems. *Networks*, 22(1):55–89, 1992.

[44] Michelle Cotton, Leo Vegoda, and David Meyer. Iana guidelines for ipv4 multicast address assignments, 2010.

[45] Srinivasan Keshav and Sanjoy Paul. Centralized multicast. In *IEEE International Conference on Network Protocols*, pages 59–68, 1999.

[46] Yang-hua Chu, Sanjay G Rao, Srinivasan Seshan, and Hui Zhang. A case for end system multicast. *IEEE Journal on selected areas in communications*, 20(8):1456–1471, 2002.

[47] Bob Quinn and Kevin Almeroth. Ip multicast applications: Challenges and solutions, 2001.

[48] Suman Banerjee, Bobby Bhattacharjee, and Christopher Kommareddy. Scalable application layer multicast. In *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 205–217, 2002.

[49] Alok Kumar, Sushant Jain, Uday Naik, Anand Raghuraman, et al. Bwe: Flexible, hierarchical bandwidth allocation for wan distributed computing. In *ACM SIGCOMM*, pages 1–14, 2015.

[50] Virajith Jalaparti, Ivan Bliznets, Srikanth Kandula, Brendan Lucier, and Ishai Menache. Dynamic pricing and traffic engineering for timely inter-datacenter transfers. In *ACM SIGCOMM*, pages 73–86, 2016.

[51] Xin Jin, Yiran Li, Da Wei, Siming Li, et al. Optimizing bulk transfers with software-defined optical wan. In *ACM SIGCOMM*, pages 87–100, 2016.

[52] Nick McKeown. Software-defined networking. *IEEE INFOCOM keynote talk*, 17(2):30–32, 2009.

[53] Ben Pfaff, Bob Lantz, Brandon Heller, et al. Openflow switch specification, version 1.3.1 (wire protocol 0x04). https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.1.pdf, 2012.

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

[55] Openflow v1.3.1 compliance matrix for devices running junos os. https://www.juniper.net/documentation/en_US/junos/topics/reference/general/junos-sdn-openflow-v1.3.1-compliance-matrix.html. visited on March 14, 2017.

[56] S12700 series agile switches. http://e.huawei.com/us/products/enterprise-networking/switches/campus-switches/s12700. visited on March 14, 2017.

[57] Hp 5920 & 5900 switch series openflow command reference. http://h20565.www2.hpe.com/hpsc/doc/public/display?sp4ts.oid=5221896&docId=emr_na-c04089449&docLocale=en_US. visited on March 14, 2017.

[58] Hp 5130 ei switch series openflow configuration guide. http://h20565.www2.hpe.com/hpsc/doc/public/display?sp4ts.oid=7399420&docLocale=en_US&docId=emr_na-c04771714. visited on March 14, 2017.

[59] Alexander L Stolyar and Kavita Ramanan. Largest weighted delay first scheduling: Large deviations and optimality. *Annals of Applied Probability*, pages 1–48, 2001.

[60] Adam Wierman and Bert Zwart. Is tail-optimal scheduling possible? *Operations Research*, 60(5):1249–1257, 2012.

[61] Mosharaf Chowdhury and Ion Stoica. Coflow: A networking abstraction for cluster applications. In *ACM HotNets*, pages 31–36, 2012.

[62] Yilong Geng, Vimalkumar Jeyakumar, Abdul Kabbani, and Mohammad Alizadeh. Juggler: a practical reordering resilient network stack for datacenters. In *ACM Eurosys*, page 20, 2016.

[63] Costin Raiciu, Christoph Paasch, Sebastien Barre, et al. How hard can it be? designing and implementing a deployable multipath tcp. In *NSDI*, pages 29–29, 2012.

[64] Gabriel Robins and Alexander Zelikovsky. Tighter bounds for graph steiner tree approximation. *Journal on Discrete Mathematics*, 19(1):122–134, 2005.

[65] Dimitri Watel and Marc-Antoine Weisser. *A Practical Greedy Approximation for the Directed Steiner Tree Problem*, pages 200–215. Springer International Publishing, 2014.

[66] Evaluation of approximation algorithms for the directed steiner tree problem. https://github.com/mouton5000/DSTAlgoEvaluation. visited on Apr 27, 2017.

[67] Inc. Gurobi Optimization. Gurobi optimizer reference manual. http://www.gurobi.com, 2016.