Delay-Optimized Video Traffic Routing in Software-Defined Inter-Datacenter Networks

Yinan Liu, Di Niu, *Member, IEEE* and Baochun Li, *Fellow, IEEE*

***Abstract*—Many video streaming applications operate their geo-distributed services in the cloud, taking advantage of superior connectivities between datacenters to push content closer to users or to relay live video traffic between end users at a higher throughput. In the meantime, inter-datacenter networks also carry high volumes of other types of traffic, including service replication and data backups, e.g., for storage and email services. It is an important research topic to optimally engineer and schedule inter-datacenter traffic, taking into account the stringent latency requirements of video flows when transmitted along inter-datacenter links shared with other types of traffic. Since inter-datacenter networks are usually over-provisioned, unlike prior work that mainly aims to maximize link utilization, we propose a delay-optimized traffic routing scheme to explicitly differentiate path selection for different sessions according to their delay-sensitivities, leading to a software-defined inter-datacenter networking overlay implemented at the application layer. We show that our solution can yield sparse path selection by only solving linear programs, and thus in contrast to prior traffic engineering solutions, does not lead to overly fine-grained traffic splitting, further reducing packet resequencing overhead and the number of forwarding rules to be installed in each forwarding unit. Real-world experiments based on a deployment on 6 globally distributed Amazon EC2 datacenters have shown that our system can effectively prioritize and improve the delay performance of inter-datacenter video flows at a low cost.**

***Index Terms*—Inter-datacenter networks; video traffic; routing algorithm; packet delays; software-defined networking.**

I. Introduction

To reduce management costs and improve performance, many video streaming applications rely on public cloud providers, such as Amazon Web Service (AWS), Google Cloud and Microsoft Azure. For example, Netflix has migrated its entire video streaming service to AWS datacenters [1]. In addition, with a growing demand for social live streaming, many applications allow users to broadcast their videos live to other users, such as Periscope [2], Meerkat [3], and FaceTime [4]. All these video streaming applications can greatly benefit from a well provisioned inter-datacenter wide-area network (inter-DC WAN), provided by major cloud service providers. With inter-DC capacities approaching hundreds of Mbps or higher [5], content stored at one datacenter can be swiftly transferred to another that is closer to the requesting user. In terms of social live streaming, the content generated at one user's mobile device can be relayed through the inter-DC network to other users in remote regions at a much higher throughput than that of direct point-to-point links.

For similar reasons, other types of applications are also relying on the high-speed inter-DC network of cloud providers, typically for service replication, bulk transfers, data backup, and database maintenance, especially in cloud storage (e.g., Dropbox, Google Drive and Microsoft OneDrive) and email services. Therefore, flows with different service priorities share the same links in inter-DC networks. Under this scenario, the video traffic, which has a much more stringent requirement on latency, has to compete for the shortest paths with all kinds of traffic, including bandwidth-hungry flows, which are large in size yet less sensitive to latency.

However, current solutions for distributed inter-DC network resource allocation are not particularly designed to accommodate video streaming flows with high delay-sensitivities. As an example, Multiprotocol Label Switching Traffic Engineering (MPLS TE) is commonly applied in most inter-DC WANs today [6], [7]. Without using any global coordination, MPLS TE may greedily assign flows to the shortest paths with available capacity as they arrive [6], which often leads to suboptimal routing decisions. For example, delay-sensitive video streaming traffic may be forced onto detoured paths with longer latencies, when earlier background flows have occupied direct links between datacenter pairs. Moreover, with MPLS TE, it is difficult to prioritize routing decisions and path selections according to service urgency levels.

In response, it has recently become an important research topic to engineer inter-DC traffic from a global point of view. Google's software-defined inter-DC WAN, B4 [8], adopts an approximate fairness criterion to greedily maximize flow rate allocations according to certain bandwidth functions that indicate flow priorities. Microsoft has presented SWAN [5], a software-driven inter-DC WAN, that classifies flows into three classes — interactive, elastic and background flows — according to their delay-sensitivities, and solves a multi-commodity flow problem in each flow class to maximize the throughput. With centralized traffic engineering and control over Openflow-enabled switches in a software-defined network, these solutions have improved network throughput and addressed the flow priority issues to a certain degree.

Unfortunately, the routing algorithms in B4 and SWAN are not explicitly designed to minimize packet delays for delay-sensitive video flows, which are transmitted among other diverse types of inter-DC traffic. Moreover, a common limitation faced by both B4 and SWAN is that traffic engineering usually leads to fractional rate allocations on multiple paths, incurring a large number of rules to be installed on switches, disregarding rule count limits on hardware switches. Additionally,

Y. Liu and B. Li are with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON, Canada. E-mail: {yinan, bli}@ece.toronto.edu

D. Niu is with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, Canada. E-mail: dniu@ualberta.ca

splitting traffic over a large number of parallel paths may incur packet resequencing overhead at the destination, which may also increase latencies. In fact, the problem of finding a fixed number of paths that maximize throughput is NP-complete [9].

In this paper, we design, implement and evaluate a new *software-defined sparse traffic routing solution* for inter-DC networks, specifically tailored to optimize packet delays for delay-sensitive video traffic, when they are transmitted among other diverse types of flows. We make the following major contributions:

First, unlike prior work on inter-DC traffic engineering that mainly aims to maximize throughput or link utilization, we explicitly optimize flow latencies according to delay-sensitivities, which is more important for video traffic. Prior work [5] implicitly optimizes latencies by allocating rates to three flow classes (interactive, elastic, background) one after another following the order of urgency. In contrast, we do not divide delay-sensitivities into a small and fixed number of classes. Instead, we respect the diverse delay-sensitivities of all flows and incorporate a fine-grained priority measure in our optimization. Note that unlike throughput which is a linear function of rate allocation, the latency of a flow split on multiple paths (i.e., the maximum latency of all chosen paths with non-zero rates) is a non-convex function of the allocated rates, which makes the problem a non-convex integer program. We adapt the *Log-det* heuristic [10], which was originally introduced to solve matrix rank minimization [11], to tackle our delay minimization problem through a short series of linear programs (LPs). We mathematically prove the convergence of the proposed algorithm under a range of delay measures and demonstrate its fast convergence and scalability in general scenarios.

Second, while leveraging multipath diversity, our solution yields *sparse* path selection such that each flow is assigned to only one path in most cases (two paths in some rare cases), leading to a small number of switching rules, and addressing the issue of fine-grained flow splitting. Moreover, sparse path selection also helps to reduce packet reordering overhead, and thus further reducing packet delays. We show that our proposed Log-det algorithm converges under sparsity regularization and effectively limits the path selection for most sessions to a *single* path.

Third, we have implemented delay-optimized sparse traffic routing in a software-defined networking (SDN) framework at the *application layer* instead of the network layer, requiring no change on the underlying infrastructure. Since each datacenter usually has only a few WAN switches [5] and the number of flow entries a hardware switch can support is limited, these hardware switches will not be able to store a large number of computed rules after all, as the number of flows scales up. Further supported by the fact that not all switches in datacenters support OpenFlow [12] and SDN, it may be costly, if feasible at all, to implement complicated and dynamic inter-DC routing rules at large scales in hardware switches.

In our application-layer SDN implementation, we launch computing instances at different datacenters and use these virtual machines as "forwarding devices." We implement a centralized controller that optimizes traffic routing decisions based on measured delays and rate demands at the edges, and installs converted forwarding rules into the "forwarding devices" launched on demand. By adopting SDN concepts at the application layer, our solution supports a potentially unlimited number of rules and highly intelligent traffic engineering. In the meantime, our solution can still benefit from all the innovations in L2 and L3 layers without changing the current infrastructure.

To evaluate the performance of our delay-optimized inter-DC video traffic routing solution, we have implemented our system on an inter-DC network of 6 Amazon datacenters around the globe. Our experimental results show that by optimally selecting paths for coexisting flows based on their delay-sensitivities, our solution can always prioritize and reduce packet delays for sensitive video flows to meet their latency requirements, while maximizing network throughput and utilization.

The remainder of this paper is organized as follows. In Sec. II, we discuss our work in the context of related work. In Sec. III, we formulate the delay-optimized inter-DC traffic routing problem under various delay measures. In Sec. IV, we propose an algorithm to generate the sparse routing solutions, and mathematically prove its convergence. In Sec. V, we present our SDN-based implementation for delay-optimized inter-DC video traffic routing at the application layer. In Sec. VI, we show an extensive set of performance evaluations based on our deployment in the Amazon EC2 cloud. Sec. VII concludes the paper.

II. Related Work

There have been many studies focusing on improving video delivery and transfer quality in communication networks [13]–[19]. Similar to this work, Feng et al. [20], [21] focused on managing video flows in the context of inter-DC networks. In [20], Feng et al. studied live multi-party video conferencing as a cloud service while maximizing the total throughput of all sessions. In [21], they further designed algorithms to minimize cloud providers' operational cost to route inter-DC video traffic given the ISP pricing model. In contrast, our goal is to minimize packet delays for inter-DC video traffic (e.g., flows of Netflix and Periscope), and to improve quality of service (QoS) even when video flows are transmitted with high volumes of other competing traffic types among datacenters.

Inter-DC traffic engineering has been an important research topic recently, with increasingly high volumes of traffic [22]. Multiprotocol Label Switching Traffic Engineering (MPLS TE) [23], [24] were commonly used in many production inter-DC networks. In MPLS TE, equal cost multipath routing (ECMP) was used to first split the traffic at ingress router, and then a constrained shortest path first (CSPF) algorithm was adopted to find the best paths for each flow running in inter-DC networks. There are two mechanisms applied in MPLS TE to make sure that different services can enjoy the corresponding forwarding treatment based on their service priorities. First, paths with lower latencies and higher bandwidth are assigned to high-priority services. Second, there are multiple priority queues (4-8) on each switch, different types of services are

queued at different priority queues. However, MPLS TE is likely to select the locally optimal routes instead of the globally optimal ones, due to distributed knowledge of the entire network state [6]. Our Sparse Traffic Routing algorithm running in an SDN-based system has the global view of the entire network, and can coordinate all running sessions on demand to achieve their desired resource allocation objectives.

Although there exists a rich literature on using traditional switch-level SDNs for traffic engineering [25]–[31], the existing literature focused on the case of intra-DC networks, rather than inter-DC networks where latency is a concern. SWAN [5] and B4 [8] focused instead on inter-DC traffic engineering, and are closely related to our work. Both SWAN and B4 were implemented based on the SDN paradigm to manage traffic forwarding across multiple datacenters. SWAN [5] used linear programs (LPs), which solved a multi-commodity flow (MCF) problem, to allocate rates for sessions to maximize the throughput. It performed rate allocation first for interactive flows, then for elastic flows and finally for background flows. B4 [8] adopted approximate fairness to greedily maximize flow rate allocations according to certain bandwidth functions that indicated flow priorities. Both B4 and SWAN used heuristics to select a subset of routes from a large number of paths generated by optimization, and quantized the traffic splitting granularity down to the level supported by the switching hardware.

Our work distinguishes itself from SWAN and B4 in four aspects. *First*, instead of maximizing throughput, we aim at reducing delays for delay-sensitive video sessions. Different from Internet Service Provider (ISP) WANs that reach end users, to achieve reliability, cloud providers typically over-provision their inter-DC link capacity by 2-3$\times$ on a dedicated backbone [8], with an average utilization of 30-60% even on busy links [5]. Such a high bandwidth capacity implies that most inter-DC flows can always be accommodated at their target rates, except for a few bandwidth-hungry yet delay-tolerant background flows, which can easily be set aside during busy periods. In this case, it is less relevant to increase flow rates and more important to optimize flow latencies according to their diverse needs. *Second*, we do not divide flows into three fixed classes. Instead, we allow more fine-grained session priority measures indicated by a session weight, where a high weight implies high delay-sensitivity. *Third*, we propose a new algorithm that is able to generate sparse path selection by solving the sparsity-regularized optimization via LPs, which greatly and effectively reduces traffic splitting overhead at the source and packet resequencing overhead at the destination. *Finally*, our system is implemented as an application-layer SDN instead of at the network layer, and can thus support a large number of forwarding rules, with no concern of the flow table size limit. Therefore, it can potentially scale up to a much larger number of sessions.

It is worth mentioning that a substantial amount of work [32]–[36] has been proposed in the 1990s to improve service quality for diverse applications with different priorities. Integrated Services (IntServ) [32], [33] attempted to provide QoS guarantees in networks in a fine-grained fashion. Differentiated Services (DiffServ) [34] classified and marked packets at edge routers to manage network traffic. Our work is significantly different from this stream of work by adopting SDN-based traffic engineering at the application layer. By performing path selection and rate allocation in a centralized and data-driven manner, our scheme can better approximate global optimal solutions to prioritize flow delay minimization, whereas conventional traffic engineering approaches mentioned above are often trapped into suboptimal solutions due to local views.

III. Delay-Optimized Sparse Traffic Routing

In an inter-DC network, given a collection of coexisting unicast or multicast sessions of different delay-sensitivities (including video and other types of sessions), we solve a traffic routing problem to determine the sending rate of each session on each available path (or tree in the case of multicast). Our objective is to maximize a certain aggregate network utility that translates to delay or throughput objectives, subject to bandwidth capacity constraints on inter-DC links. Moreover, for each session, the sending rates should be *non-zero only on a couple of paths or trees* to yield sparse path selection.

We model an inter-DC network of geo-distributed datacenters as a complete and directed graph $G = (V, E)$, with $N = |V|$ representing the number of datacenters. For each edge $e \in E$, we use $C(e)$ to denote its availability bandwidth capacity, and $L(e)$ to denote the latency on link e, measured by the *one-way* delay, taking into account both propagation delay and queuing delay. Suppose there are S unicast and/or multicast sessions in G, each with a required target rate R_i, $i = 1, 2, \ldots, S$ and a *priority parameter* $w_i > 0$, where a larger w_i indicates a higher priority and a greater sensitivity to latency. Typically, a video streaming session has a higher priority value, whereas elastic and background flows have lower priority values. Since each unicast session is a special case of a multicast session, we will only consider multicast sessions henceforth.

For each session i originated at a certain datacenter, it is not difficult to find out all the feasible trees (or paths in the case of unicast) to reach all the destinations by a depth-first search algorithm to be elaborated in Sec. III-C. For each session i, we denote the feasible multicast trees as $T_{i1}, \ldots, T_{ik_i}$, and represent the packet latency on each multicast tree T_{ij} by L_{ij}, for $j = 1, \ldots, k_i$. Furthermore, let r_{ij} denote the rate allocated to the tree T_{ij}. Then, our objective is to determine a sparse rate allocation r_{ij} such that most r_{ij} are zeros while achieving different network utility objectives under different cases.

A. Latency Minimization in Over-Provisioned Networks

Most inter-DC networks are over-provisioned, as has been reported by Microsoft [5] and Google [8]. Although the inter-DC network capacity is usually sufficient to accommodate all the session demands, there could still be inefficient routing decisions that do not optimize session packet delays based on their priorities. Let us illustrate the idea in a simple toy example. Consider the network shown in Fig. 1, where each link has a bandwidth capacity of 2 units and a link latency of 1 ms. There are 5 sessions S1,. . ., S5 all sending packets from the same source, node A, to the same destination, node

C. Suppose that sessions S1 and S2 are background sessions, and each consumes 1 unit of bandwidth with lower delay-sensitivity, while sessions S3, S4, and S5 are video sessions, and each consumes 0.5 unit of bandwidth yet with higher delay-sensitivity. The total bandwidth capacity from node A to C is 4 units and is able to accommodate all 5 sessions which have a total bandwidth demand of 3.5 units. Thus, the network is over-provisioned.

Now we describe a typical scenario in which the routing decision is not respecting delay-sensitivities of different sessions, and degrades the performance of video sessions. Suppose S1 and S2 are background flows. They typically arrive first and occupy the best direct route $A \to C$. The video sessions S3, S4 and S5 join afterwards, and will have to take the detoured path $A \to B \to C$ if constrained shortest path first (CSPF) routing is used, since S1 and S2 have already used up the bandwidth on the direct route $A \to C$, as shown in Fig. 1(a). Yet, in this case, video sessions S3, S4, and S5 will suffer from a longer latency of 2 ms, whereas the direct route $A \to C$ has been occupied by background sessions who do not care about latency at all.

A better routing decision is to adjust the routing decision after sessions S3, S4 and S5 have joined, such that video sessions will take the direct route $A \to C$, while background flows should only be allocated with the remaining bandwidth on $A \to C$, and even rerouted to detoured routes if the direct route does not have enough bandwidth, as shown in Fig. 1(b).

We run a data-driven optimization framework to update traffic routing strategies periodically, respecting delay-sensitivity levels of different sessions, regardless of the order they join the network. In over-provisioned networks, all rate demands can be accommodated. Therefore, maximizing the network utility can be translated to minimizing an aggregate delay objective as follows:

$$\underset{\{r_{ij}\}}{\text{minimize}} \quad \sum_{i=1}^{S} w_i \max_{j\in\{1,\dots,k_i\}} \{L_{ij}\phi(r_{ij})\} + \gamma \sum_{(i,j)} \phi(r_{ij}) \tag{1}$$

$$\text{subject to} \quad \sum_{j=1}^{k_i} r_{ij} = R_i, \quad \forall i \in \{1,\dots,S\}, \tag{2}$$

$$\sum_{(i,j):e\in T_{ij}} r_{ij} \le C(e), \quad \forall e \in E, \tag{3}$$

$$r_{ij} \ge 0, \quad \forall (i,j), \tag{4}$$

where $\phi(\cdot)$ is an identity function defined as

$$\phi(x) = \begin{cases} 1, & \text{if } x > 0, \\ 0, & \text{if } x = 0. \end{cases} \tag{5}$$

Now we explain the rationale behind the formulation above. The objective of problem (1) is to minimize the weighted sum of session delays by controlling r_{ij}, i.e., the flow rate assigned to the tree j within each session i. The term $\max_{j\in\{1,\dots,k_i\}}\{L_{ij}\phi(r_{ij})\}$ represents the worst-case packet latency of session i, which is the longest latency among all the *chosen* trees that have non-zero r_{ij}s, since the tree T_{ij} is adopted with $\phi(r_{ij}) = 1$ only if $r_{ij} \neq 0$. The weight

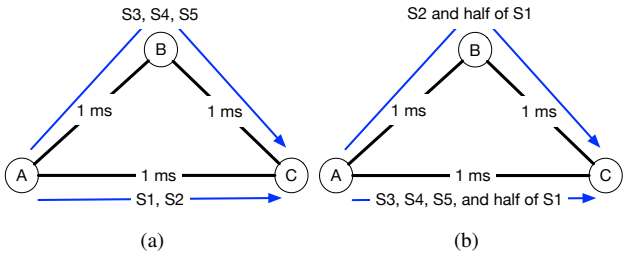


Fig. 1. A poor routing decision may lead to unoptimized packet delays even in an over-provisioned network. Each link has a bandwidth capacity of 2 units.

$w_i > 0$ is used to prioritize different sessions according to their delay-sensitivities. The weight of background flow with the minimum requirement on latency should have a w_i close to 0, and the weight of an interactive delay-sensitive session will have a larger w_i. The regularizing term $\gamma \sum_{(i,j)} \phi(r_{ij})$ is a penalty function to yield sparse tree selection by preferring solutions with fewer non-zero r_{ij}s. Constraint (2) guarantees that flow rates distributed to different trees must sum up to the session target rate in over-provisioned networks. Constraint (3) ensures that none of the link capacities is violated. Thus, congestion will never occur on each link and in this case, L_{ij} will be independent of rate allocations during optimization.

A variation to problem (1) is

$$\underset{\{r_{ij}\}}{\text{minimize}} \quad \sum_{i=1}^{S} w_i \sum_{j=1}^{k_i} L_{ij}\phi(r_{ij}) + \gamma \sum_{(i,j)} \phi(r_{ij}) \tag{6}$$

subject to the same constraints (2)−(4). In (6), the worst-case latency of each session i is replaced by the sum of latencies of all the chosen trees $\sum_{j=1}^{k_i} L_{ij}\phi(r_{ij})$. Note that $\sum_{j=1}^{k_i} L_{ij}\phi(r_{ij})$ can also be replaced by $\sum_{j=1}^{k_i} L_{ij}r_{ij}$, which indicates the average packet latency. However, problem (6) is preferred in order to yield both sparsity and low latencies, since minimizing $\sum_{j=1}^{k_i} L_{ij}\phi(r_{ij})$ also penalizes the number of trees with non-zero rates while favouring low-latency trees among the sparse selections.

Another variation is to minimize an aggregate objective on tree depths, without measuring latencies L_{ij}, i.e.,

$$\underset{\{r_{ij}\}}{\text{minimize}} \quad \sum_{i=1}^{S} w_i \sum_{d=1}^{D} d \max_{j:\text{depth}(T_{ij})=d} \phi(r_{ij}) + \gamma \sum_{(i,j)} \phi(r_{ij}), \tag{7}$$

where for each session i, $\max_{j:\text{depth}(T_{ij})=d} \phi(r_{ij})$ is 0 if and only if no tree T_{ij} of depth d is selected. Thus, unlike problem (6) which minimizes the sum of latencies on all the chosen trees, problem (7) minimizes for each session the sum of *all distinct tree depth values* that are adopted. For example, if session i only uses one-hop trees, $\sum_{d=1}^{D} \max_{j:\text{depth}(T_{ij})=d} \phi(r_{ij}) = 1$; if it uses both one-hop and two-hop trees, $\sum_{d=1}^{D} \max_{j:\text{depth}(T_{ij})=d} \phi(r_{ij}) = 1+2 = 3$. Therefore, problem (7) penalizes the number of distinct tree depths adopted, while preferring trees with low depths. And the advantage of problem (7) is that it does not even require latency measurements between datacenters.

In all of these problems (1), (6) and (7), delay-sensitive sessions (e.g., video streaming) with larger w_is will be routed

to low-latency (low-depth) paths, while delay-tolerant sessions like background flows with low w_is may be placed on detour. However, as soon as delay-sensitive video sessions conclude, background flows will be routed back onto the shortest paths due to positive though small w_is. Apparently, problems (1), (6) and (7) are all non-convex optimization problems, because of the identity function $\phi(r_{ij})$.

B. Utility Maximization under Insufficient Bandwidth

In some rare cases, in spite of over-provisioned inter-DC networks, some bandwidth-hungry background flows (with w_i close to 0) may be too large to be accommodated by the available network capacity. Even in this case, the aggregate traffic of delay-sensitive video sessions is most likely still small enough to be accommodated by the network. Therefore, we can perform a two-stage process to handle bandwidth-hungry flows. In the first stage, we solve problem (1), (6) or (7) for all delay-sensitive sessions. Then, in the second stage, we subtract the rates allocated to delay-sensitive sessions from the bandwidth capacity to obtain the remaining link capacity. And we allocate the rates for background flows in the remaining network to maximize their throughput, e.g., by solving a multi-commodity flow (MCF) problem [5], which is an LP, or a min-cost MCF to penalize long paths. Alternatively, in the second stage, we can perform max-min fair rate allocation for all the background flows onto the remaining network [9], although in most cases, background flows are indifferent to latency performance and fairness issues.

In case bandwidth is not sufficient to accommodate even delay-sensitive sessions, which is an extremely rare scenario in today's heavily over-provisioned inter-DC networks, we propose to solve a *sparsity-penalized* network utility maximization (NUM) problem to judiciously decide the allowable sending rate $\sum_{j=1}^{k_i} r_{ij}$ for each session i:

$$\underset{\{r_{ij}\}}{\text{maximize}} \quad \sum_{i=1}^{S} w_i U_i(r_{i1},\ldots,r_{ik_i}) - \gamma \sum_{(i,j)} \phi(r_{ij}) \tag{8}$$

$$\text{subject to} \quad R_i^L \le \sum_{j=1}^{k_i} r_{ij} \le R_i, \quad \forall i \in \{1,\ldots,S\}, \tag{9}$$

$$\sum_{(i,j):e\in T_{ij}} r_{ij} \le C(e), \quad \forall e \in E, \tag{10}$$

$$r_{ij} \ge 0, \quad \forall (i,j), \tag{11}$$

where $U_i(r_{i1},\ldots,r_{ik_i})$ is a utility function for each session i depending on the achieved rate and latency, e.g.,

$$U_i(\{r_{ij}\}) = \begin{cases} \sum_{j=1}^{k_i} r_{ij} - \beta_i \max_{j\in\{1,\ldots,k_i\}}\{L_{ij}\phi(r_{ij})\}, \\ \sum_{j=1}^{k_i} r_{ij} - \beta_i \sum_{j=1}^{k_i} L_{ij}\phi(r_{ij}), \\ \sum_{j=1}^{k_i} r_{ij} - \beta_i \sum_{d=1}^{D} d \max_{j:\text{depth}(T_{ij})=d} \phi(r_{ij}), \end{cases}$$

where each U_i is a function of the total rate allocated for session i minus a certain delay measure. As NUM is a well-known formulation for optimal resource allocation and for handling network congestion, our sparsity-penalized NUM (8) further drives the rate allocations to zero on most trees. Furthermore, for a delay-sensitive video session with a larger w_i, problem (8) will prefer a solution that allocates more rate $\sum_{j=1}^{k_i} r_{ij}$ to session i until its target rate R_i is achieved in constraint (9). On the other hand, for a background flow with a small w_i, problem (8) will allocate a rate to it with a lower priority, if there is still leftover bandwidth after all delay-sensitive sessions have reached their target rates.

C. Finding Available Paths

Although the number of datacenters, N, is small, the number of possible multicast trees for each multicast session may be large. Taking the worst case of unicast sessions as an example, on a complete graph, there exist $\sum_{i=0}^{N-2} A_{N-2}^i$ possible paths between every pair of datacenters. For an Amazon EC2 cloud with 7 datacenters distributed globally, the maximum number of paths between each pair of datacenters is 326. It is not hard to imagine that the number of available multicast trees for each multicast session will be even larger.

To reduce complexity, when generating the set of feasible multicast trees for each session i, we restrict tree depth to be no more than D and *only* select k trees with the least latencies $L_{i1},\ldots,L_{ik}$. The rationale is that our goal is to reduce an aggregate delay objective; longer paths imply longer latencies and are unlikely to be chosen by our optimization outcome. Thus, we should exclude these longer paths from consideration upfront.

We use a simple variant of the depth-first search (DFS) algorithm to enumerate the k shortest multicast trees mentioned above for each session i. In particular, starting from the source node, we continue exploring as far as possible along each branch in G before reaching all the destination nodes or exceeding a hop count restriction. After all possible multicast trees $\{T_{ij}\}$ are found, we can easily calculate the latency L_{ij} based on measured link latencies, i.e., L_{ij} equals to the latency of the longest path from the source to any receiver in $\{T_{ij}\}$. Then the k shortest multicast trees can be picked. Note that in reality, choosing only 1 or 2-hop trees would be sufficient for delay-optimized routing. Furthermore, the small number of datacenters of each cloud provider makes it very fast to generate available trees using the scheme mentioned above.

IV. A Sparse Solution based on *Log-det* Heuristic

Apparently, problems (1), (6), (7) and (8) are all *non-convex* optimization problems, since $\phi(r_{ij})$ is a $0-1$ valued integer to indicate whether tree T_{ij} is chosen or not. In this section, we propose to solve these problems using a Log-det heuristic, which was first used to solve matrix rank minimization in statistical sparse recovery. As a special case, the method has been applied to *cardinality minimization* [10], i.e., finding a vector $x = (x_1,\ldots,x_n)$ with the minimum cardinality in a convex set $\mathcal{C}$, or equivalently, minimizing $\sum_i \phi(x_i)$ subject to $x \in \mathcal{C}$. The basic idea is to replace the $0-1$ valued function $\phi(x_i)$ by a smooth log function $\log(|x_i|+\delta)$ and to minimize a linearization of $\log(|x_i|+\delta)$ iteratively. In the following, we extend this technique to our particular sparse routing problems, whose objective functions are clearly far more complicated than cardinality, and provide a convergence proof for certain cases.

We present Algorithm 1, which replaces the integer value $\phi(\cdot)$ by a series of carefully re-weighted linear functions that are updated in each iteration.

Algorithm 1 Iterative Linear Relaxation for Problems (1), (6), (7), or (8)

1: $t := 0$. $r_{ij}^0 := 1 - \delta$.
2: $t := t + 1$.
3: Given the solution $\{r_{ij}^{t-1}\}$ in the previous iteration, define

$$\hat{\phi}_{ij}^t(r_{ij}) := \frac{r_{ij}}{r_{ij}^{t-1} + \delta}, \ \forall\, j \in \{1, \ldots, k_i\}, \ \forall i \in \{1, \ldots, S\},$$

where $\delta > 0$ is a small positive constant.
4: Replace $\phi(r_{ij})$ in problem (1), (6), (7), or (8) by $\hat{\phi}_{ij}^t(r_{ij})$, and solve the modified problem to obtain $\{r_{ij}^t\}$.
5: If $\{r_{ij}^t\}$ approximately equals $\{r_{ij}^{t-1}\}$, return $r_{ij}^* = r_{ij}^t$ for all possible (i, j); else, go to Step 2.

Clearly, by replacing $\phi(r_{ij})$ by $\hat{\phi}_{ij}^t(r_{ij})$ in problem (1), (6), (7), or (8), the corresponding modified problem solved by Algorithm 1 in each iteration is a convex problem, which can also be converted to an LP in the case of problem (1), (6), or (7). After a number of iterations, Algorithm 1 will eventually yield a sparse solution of $\{r_{ij}^*\}$ with most r_{ij}^* being zero. Recall that if $r_{ij}^* = 0$, the multicast tree T_{ij} is not adopted by session i.

To see why the modified problem eventually approximates the corresponding original problem, note that for a sufficiently small $\delta > 0$, upon convergence, i.e., when $r_{ij}^{t-1} \approx r_{ij}^t = r_{ij}^*$, we have

$$\hat{\phi}_{ij}^t(r_{ij}^*) = \frac{r_{ij}^*}{r_{ij}^{t-1} + \delta} \approx \begin{cases} 0, & \text{if } r_{ij}^* = 0, \\ 1, & \text{if } r_{ij}^* > 0, \end{cases}$$

which approximately equals $\phi(r_{ij}^*)$. Therefore, the objective function involving $\hat{\phi}_{ij}^t(r_{ij}^*)$ eventually approaches that of the corresponding original problem.

A. Convergence Analysis

In the following, we provide a proof for the convergence of Algorithm 1 for a class of objective functions, while prior literature can only show convergence when the objective function is the cardinality $\text{Card}(x) = \sum_i \phi(x_i)$ [10]. We point out how this result applies to various formulations proposed in the Sec. III.

Proposition 1. *Consider a non-convex optimization problem:*

$$\begin{aligned} \textit{minimize} \quad & \sum_{i=1}^{n} w_i \phi(x_i) + \sum_{j=1}^{J} l_j \max_{i \in I_j} \phi(x_i) \qquad (12) \\ \textit{subject to} \quad & x = (x_1, \ldots, x_n) \in \mathcal{C}, \end{aligned}$$

with $w_i \geq 0$ and $l_i \geq 0$, where $\mathcal{C} \subseteq \mathbb{R}^n$ is a convex set, and $I_j \subseteq \{1, \ldots, n\}$ for $j = 1, \ldots, J$. If Algorithm 1 is applied to problem (12) *with $\phi(x_i)$ replaced by $\hat{\phi}_i^t(x_i) = x_i/(x_i^{t-1} + \delta)$ with $\delta > 0$ in each iteration t, then we have $x_i^t - x_i^{t-1} \to 0$, for all i.*

Proof. Since $\phi(x_i) = \max_{i \in I_j} \phi(x_i)$, the first term in (12) is a special case of the second term. Thus, it suffices to prove the case when $w_i = 0$ for all i. Since $\{x_i^t\}$ solves problem (12) with $\phi(x_i)$ replaced by $\hat{\phi}_i^t(x_i) = x_i/(x_i^{t-1} + \delta)$, and $\{x_i^{t-1}\} \in \mathcal{C}$, we have

$$\sum_{j=1}^{J} l_j \max_{i \in I_j} \frac{x_i^t + \delta}{x_i^{t-1} + \delta} \leq \sum_{j=1}^{J} l_j \max_{i \in I_j} \frac{x_i^{t-1} + \delta}{x_i^{t-1} + \delta} = \sum_{j=1}^{J} l_j. \quad (13)$$

On the other hand, define $y_i(t) := \frac{x_i^t + \delta}{x_i^{t-1} + \delta}$. By the inequalities between geometric and arithmetic means, we have

$$\begin{aligned} \sum_{j=1}^{J} l_j \max_{i \in I_j} y_i(t) &\geq \sum_{j=1}^{J} l_j \prod_{i \in I_j} y_i(t)^{1/|I_j|} \\ &\geq \left(\sum_{j=1}^{J} l_j \right) \cdot \left(\prod_{j=1}^{J} \prod_{i \in I_j} y_i(t)^{l_j/|I_j|} \right)^{\frac{1}{\sum_{j=1}^{J} l_j}}. \end{aligned} \quad (14)$$

Combining with inequality (13), we obtain

$$\prod_{j=1}^{J} \prod_{i \in I_j} y_i(t)^{l_j/|I_j|} = \prod_{i=1}^{n} y_i(t)^{\sum_{j: i \in I_j} l_j/|I_j|} \leq 1,$$

or equivalently,

$$\prod_{i=1}^{n} (x_i^t + \delta)^{\sum_{j: i \in I_j} l_j/|I_j|} \leq \prod_{i=1}^{n} (x_i^{t-1} + \delta)^{\sum_{j: i \in I_j} l_j/|I_j|}.$$

Since $l_i \geq 0$, the left-hand side of the above is bounded below by $\delta^{\sum_{j: i \in I_j} l_j/|I_j|}$. Therefore, the non-increasing sequence $\{\prod_{i=1}^{n} (x_i^t + \delta)^{\sum_{j: i \in I_j} l_j/|I_j|}\}$ is converging over t to a nonzero limit, which implies that

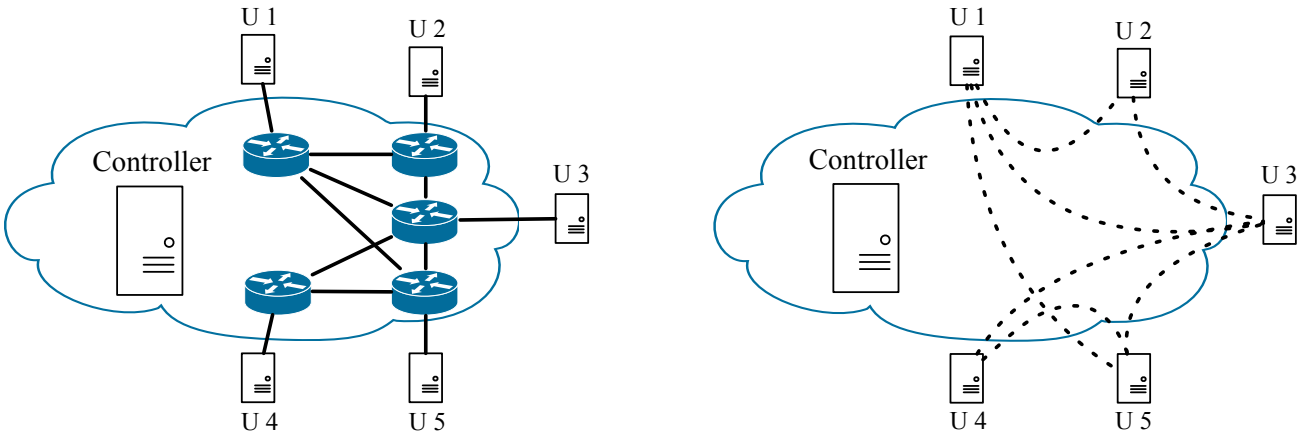
$$\lim_{t \to \infty} \prod_{i=1}^{n} y_i(t)^{\sum_{j: i \in I_j} l_j/|I_j|} = 1.$$

Now using the inequality (14), we obtain

$$\begin{aligned} &\sum_{j=1}^{J} l_j \max_{i \in I_j} \big(\lim_{t \to \infty} y_i(t) \big) \\ &\geq \left(\sum_{j=1}^{J} l_j \right) \cdot \left(\lim_{t \to \infty} \prod_{i=1}^{n} y_i(t)^{\sum_{j: i \in I_j} l_j/|I_j|} \right)^{\frac{1}{\sum_{j=1}^{J} l_j}} = \sum_{j=1}^{J} l_j, \end{aligned}$$

with equality achieved *only if* $\lim_{t \to \infty} y_i(t) = 1$ for all i. Combining with inequality (13), we obtain $\sum_{j=1}^{J} l_j \max_{i \in I_j} \lim_{t \to \infty} y_i(t) = \sum_{j=1}^{J} l_j$. Therefore, we must have $\lim_{t \to \infty} y_i(t) = 1$ for all i, which means $x_i^t - x_i^{t-1} \to 0$ for all i. ∎

It is not hard to check that problem (6) is a special case of the general form (12), when there is no second term (i.e., $l_j = 0$ for all j), and (7) is also a special case of (12) when each index group I_j corresponds to all the trees of a certain depth d in a certain session i. However, the convergence in the case of problem (1) is much harder to analyze, which involves a different L_{ij} inside each max function. We will evaluate and compare the performance of different formulations in our experiments in the Sec. VI.



(a) Existing SDN at the network layer.

(b) Our implementation: SDN applied at the application layer.

Fig. 2. Examples of software-defined networks at the network vs. application layer with the same underlying network topology.

V. Implementation

We have completed a real-world implementation of proposed sparse traffic inter-DC routing mechanism. Our implementation is based on SDN implemented at the *application* layer, which provides us with the capability to optimize packet forwarding in a globally optimal manner. Despite the merits of existing SDN solutions based on OpenFlow, the table size is limited in hardware switches, which can not scale well to support a large number of unicast and multicast rules as the session number grows. Furthermore, not all hardware switches are OpenFlow-enabled. In contrast, by implementing the full control plane and data plane at the application layer, our implementation can leverage the advantage of intelligent traffic engineering offered by SDN, without changing any current infrastructure. Furthermore, our system can still take advantage of all the latest technologies in Layer 2 and/or Layer 3 network components.

By implementing the core principles of the SDN paradigm, which is the separation of the control plane from the data plane, we are able to preserve the control flexibility of SDN at the application layer. Fig. 2 illustrates the difference between the existing SDN solution and our application-layer SDN. The application layer provides a much simpler abstraction; it hides complexities of handling flows with lower layer components and avoids fatal issues that are easy to encounter. For instance, lower-layer messages, such as ARP packets, will not be one of our concerns since they are handled naturally by the lower layer. Moreover, building our system completely at the application layer makes it readily deployable in the inter-DC network of any public cloud provider without making assumptions on their underlying infrastructure. Furthermore, by using virtual machines (VMs) as forwarding devices, we do not have a hard limit on the number rules that can be supported, which means that our application-layer solution can scale to a large number of sessions.

There are two major components in our system: a *centralized controller* implemented in Python to compute routing decisions, and *forwarding devices* implemented in C++ to forward traffic based on the rules instructed by the controller. Both the centralized controller and forwarding devices are operated as software systems running in virtual instances launched at different datacenters. As the brain of our system, the controller is launched in a virtual machine in one of the datacenters. It offers the abstraction and programmability over the network. The forwarding device for each datacenter in the inter-DC network is launched as one or multiple VMs in that datacenter.

The controller and forwarding devices interact in intricate ways. Each forwarding device is connected to the controller using a *long-lived* TCP connection. Whenever a new forwarding device comes online, the controller will save its information and send it to other existing forwarding devices. When a forwarding device has obtained information about other devices in the network, it will start measuring the one-way delays from itself to other devices in the network, and reports the measured values to the controller to be used in routing optimization. In addition, whenever a new flow emerges in the network, the first forwarding device that has received it will send the information of this flow, such as the delay-sensitivity weight and the requested target rate, to the controller. Therefore, the controller always has the up-to-date knowledge of the entire network.

Given the information of the sessions, link latencies and bandwidth capacities gathered from forwarding devices, our controller written in Python will run the proposed Sparse Traffic Routing algorithms to determine the best routes as well as rate assignments on the selected routes for each session according to their delay requirements. There are two steps to be conducted by the controller. *First,* it will enumerate the feasible multicast trees for each session, based on the network topology constructed from the collected information as well as the source and destination of each session reported by the forwarding devices. Combining such information, the feasible multicast trees can be readily generated using the method mentioned in Sec. III-C, and stored in a matrix to serve as the input of the next step. *Second,* by using `CVXPY` and `NumPy` packages, our program solves the optimization problem (i.e., problem (1), (6), (7) or (8) depending on

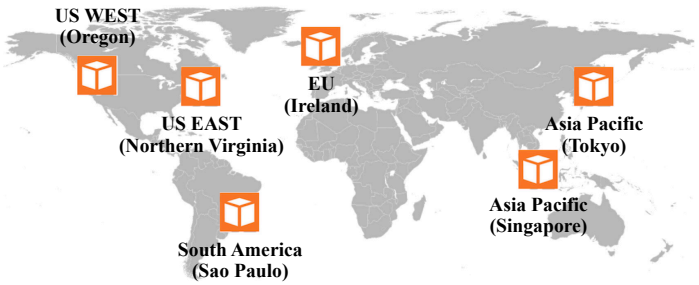


Fig. 3. The 6 Amazon EC2 datacenters used in our deployment and experiments.

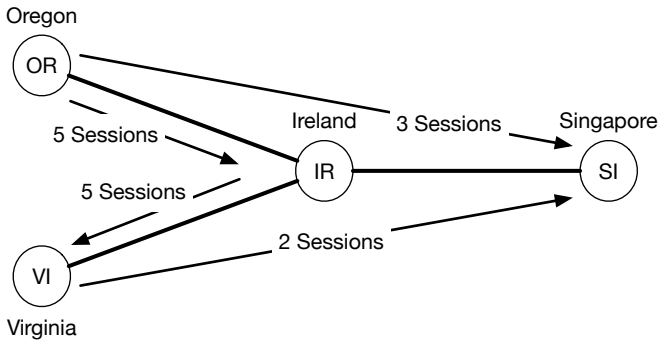


Fig. 4. Our experiment to verify the independence of link latency measurements on the throughput.

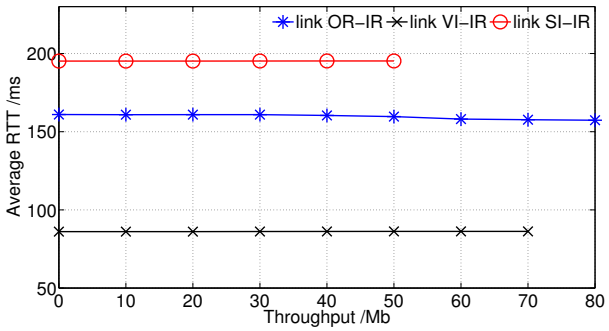


Fig. 5. The relationship between average RTT and throughput on each link in the experiment in Fig. 4.

different objectives) to obtain sparse routing decisions and optimized rate assignments within several seconds.

To reduce unnecessarily redundant computations, our routing optimization does not run whenever a new flow joins; it will be carried out every two minutes or whenever the number of new joins has exceeded a threshold. We always allow a new flow to join the system according to simple protocols, such as a greedy shortest-path-first routing based on the current network conditions. Whenever the controller has computed a new routing scheme, it will make a global re-routing decision and send out the new forwarding rules for all existing flows to all the forwarding devices.

The forwarding devices are completely managed by the controller, and are responsible for buffering and forwarding traffic. Like traditional SDN, these forwarding devices do not have control intelligence; they need to ask the controller for forwarding rules whenever they see packets of a new flow coming in. In addition to basic storing and forwarding, we have also implemented a multi-path forwarding and traffic splitting functionality in each forwarding device. Specifically, whenever a traffic splitting decision requires a proportion of the allocated flow rate to be sent on a path or tree, the forwarding device will send packets onto the path or tree according to the corresponding probability. Since our optimization algorithms may only use traffic splitting at the source, there is no need to split an incoming flow.

VI. Performance Evaluation

We have deployed our real-world implementation with delay-sensitive Sparse Traffic Routing on 6 Amazon EC2 datacenters distributed globally, whose locations are shown in Fig. 3. We have launched 6 XLarge compute instances, each with 4 compute units and 15 GB memory. Each compute instance is launched in a different datacenter and hosts our forwarding device. The centralized controller is hosted in Oregon, sharing the same compute instance as one of our forwarding devices.

A. Bandwidth and Latency Measurements

Link bandwidth capacities between the 6 datacenters can be readily measured. Measurement results have demonstrated that the available bandwidth is highly stable on all the links in the Amazon EC2 inter-DC network. To avoid queuing delays and congestion, we have been *conservative* on link capacity measurements: we use the 10th percentile of the measured available bandwidth to indicate the capacity on each link, as shown in Table I, i.e., 90% of available bandwidth values we measured are higher than the 10th percentile values used in Table I. By using a low capacity value, we make sure that our optimization will never make a decision that may potentially congest a link. During our experiments, all link latency measurements are logged once every 5 seconds.

We now show that latencies L_{ij} used as inputs in our optimization framework will not depend on rate allocations and thus can be deemed as constants during the relatively short period of a single optimization run. We conduct a simple experiment shown in Fig. 4 to verify this fact. The target rate of each session is 10 Mbps, and the available capacity on each link is listed in Table I. 15 sessions come into the network at random times in a 3-minute period. Therefore, the throughput on each link accumulates as more sessions join and start transferring data on it. Fig. 5 plots the average round-trip times (RTTs) versus the measured throughput on three links, respectively. It is obvious that the measured RTT is stable as the rate allocated to each particular link varies. Since each link's capacity is not exceeded, the queuing delay of the corresponding link will remain negligible and will not accrue, while the propagation delay still plays the most important role on each inter-DC link. Therefore, we can conclude that L_{ij}

does not depend on rate allocation decisions, and thus can be treated as a constant within a single run of optimization.

B. Schemes for Comparison

Due to the nature of inter-DC transfers, we bundle small sessions (including both unicast and multicast sessions) of similar delay-sensitivity together to form larger and longer persistent *session groups*. The rationale is that traffic engineering is meaningless for short sessions, as they may have already left the network before an optimized rerouting decision is made. Therefore, in our experiment, we bundle sessions that are originated from the same datacenter to be sent to the same destination datacenter and have similar delay sensitivities, e.g., all live video streams of a certain quality, into a same session group. Inside a session group, we might have different smaller sessions joining and leaving from time to time, while the overall session group is still kept alive and will exist in the network for a long time. A similar technique of flow groups has also been adopted in B4 [8] to avoid overhead of overly fine-grained flow management in inter-DC networks. In the following, we may abuse the term *session* to indicate session groups.

In our experiment, we will compare the following different routing solutions, in terms of achieved end-to-end delays, fine service differentiation/prioritization among sessions of different delay-sensitivities, paths sparsity and overhead of traffic split, as well as throughput when bandwidth capacity is insufficient. The latter two schemes, i.e., Shortest Path Routing and Multi-Commodity Flow, will serve as baseline schemes which have been adopted in current inter-DC routing solutions or in recent literature.

Delay-Sensitive Sparse Traffic Routing (1), (6) and (7). Our delay-sensitive Sparse Traffic Routing (1), (6) and (7) are all designed to reduce packet delays for sessions with high w_is, yet using different objective functions. It is worth noting that we have proved in the Sec. IV that Algorithm 1 for formulations (6) and (7) is guaranteed to converge in theory. Moreover, problem (7) measures the delay only by counting the number of hops and does not even require to measure link delays, which reduces the measurement overhead of system. In other words, problem (7) provides a sparse routing solution to minimize packet delays for delay-sensitive sessions, without relying on accurate latency measurements.

Shortest Path Routing. As the first baseline scheme, we have implemented the *constrained shortest path first* (CSPF) algorithm, which is commonly adopted in MPLS TE today. Specifically, it chooses the shortest path with available bandwidth for each session subject to several constraints, such as bandwidth requirements, hop limitations and session priorities.

Multi-Commodity Flow (MCF). As the second baseline scheme, MCF has been adopted in SWAN [5] to allocate rates in three classes of flows one after another, first processing interactive flows, followed by the processing of elastic flows and background flows. MCF is mainly used to increase the network utilization or total throughput while preferring shorter paths. MCF, as an LP, can be expressed as

$$\underset{\{r_{ij}\}}{\text{maximize}} \sum_{j=1}^{k_i} r_{ij} - \beta_i \sum_{j=1}^{k_i} w_i r_{ij}$$

subject to the same target rate and capacity constraints as in problem (8). In fact, MCF can be equivalently represented via problem (8), with

$$U_i(\{r_{ij}\}) := \sum_{j=1}^{k_i} r_{ij} - \beta_i \sum_{j=1}^{k_i} L_{ij} r_{ij}.$$

TABLE I
10TH PERCENTILE OF LINK CAPACITIES MEASURED IN THE AMAZON EC2 INTER-DC NETWORK.

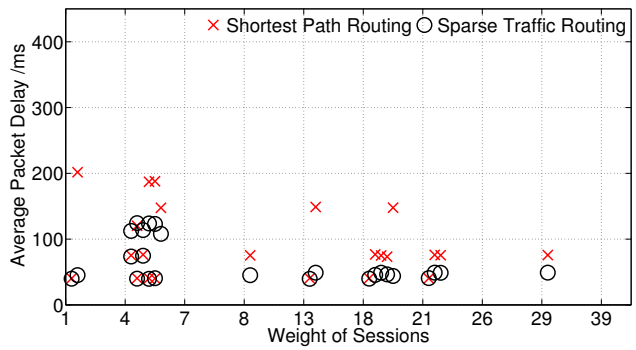
	Oregon	Virginia	Ireland	Tokyo	Singapore	Sao Paulo
Oregon	N/A	82Mbps	86Mbps	138Mbps	74Mbps	67Mbps
Virginia	-	N/A	72Mbps	41Mbps	52Mbps	70Mbps
Ireland	-	-	N/A	56Mbps	44Mbps	61Mbps
Tokyo	-	-	-	N/A	166Mbps	41Mbps
Singapore	-	-	-	-	N/A	33Mbps
Sao Paulo	-	-	-	-	-	N/A

Our evaluation mainly consists of three parts. First, we compare the performance of delay-sensitive Sparse Traffic Routing with that of Shortest Path Routing in over-provisioned networks. As randomly synthesized sessions join, Sparse Traffic Routing will be carried out *on demand* to globally re-route the sessions that already exist in the network according to their delay-sensitivities. Second, we compare different versions of delay-sensitive Sparse Traffic Routing solutions, i.e., problems (1), (6) and (7). Third, under the rare case of insufficient bandwidth, we evaluate the throughput and delay performance of our system under flow competition, as compared to Shortest Path Routing and MCF.

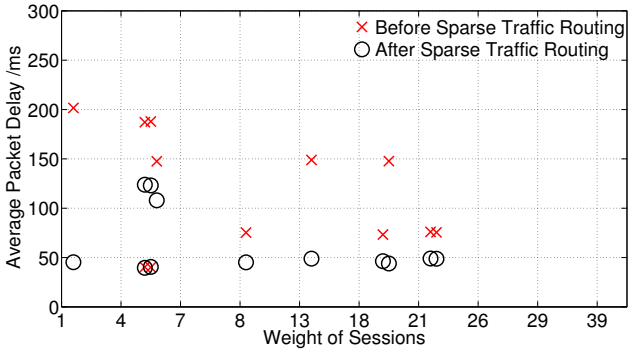
C. Sparse Traffic Routing vs. Shortest Path Routing

In this experiment, we consider the common case of over-provisioned inter-DC networks, where the total inter-DC capacity is able to support all the sessions. However, due to Shortest Path Routing, some direct links may be locally occupied by background flows or delay-insensitive sessions which have arrived earlier. This may influence the performance of subsequently arrived delay-sensitive sessions.

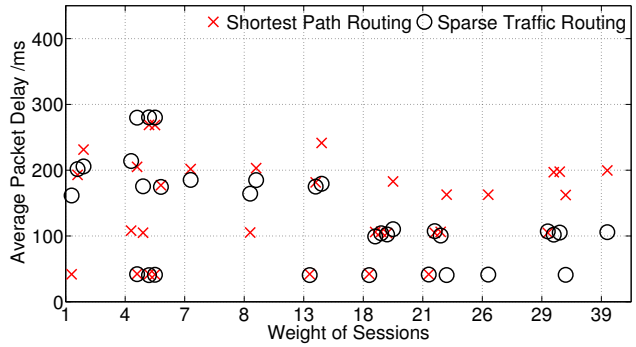
We conduct our experiment with 40 unicast/multicast sessions randomly joining the network within a 10-minute period. The weight w_i of each session takes one value out of 10 predetermined values, which are randomly drawn between 1 and 39 from an exponential distribution. The target request rate R_i for each session is chosen uniformly at random from $4 - 10$ Mbps. When a session comes, it will be immediately directed onto the *shortest path* with available bandwidth, using Shortest Path Routing. Delay-sensitive Sparse Traffic Routing will be triggered under two situations, i.e., either when 10 new sessions have joined after the last optimization, or when 2 minutes have past since the last optimization. After re-routing decisions are made by the optimization, the controller will update the new forwarding rules for all the existing sessions to be installed in the forwarding devices instantly. Under this



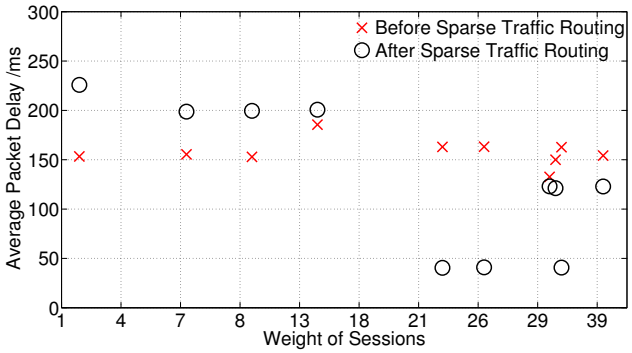
(a) Sessions 1-20 right after the 2nd optimization.



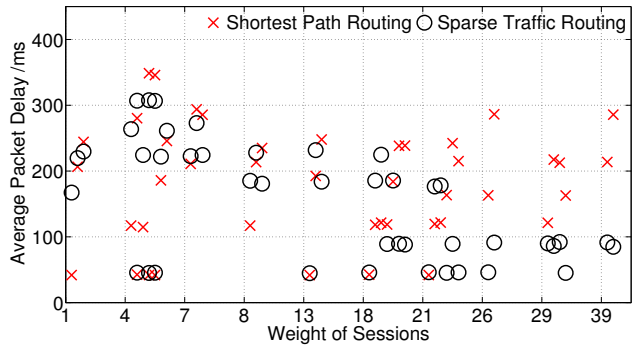
(b) Sessions 11-20 before/after the 2nd optimization.



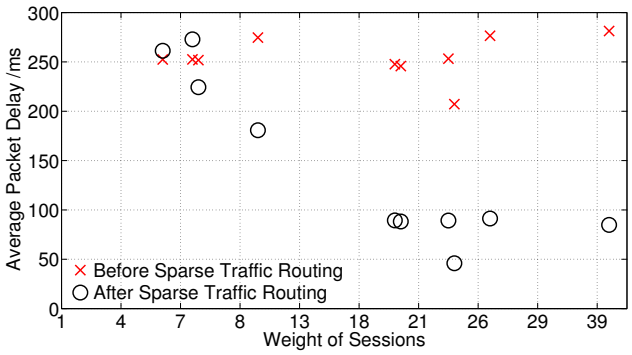
(c) Sessions 1-30 right after the 3rd optimization.



(d) Sessions 21-30 before/after the 3rd optimization.



(e) Sessions 1-40 right after the 4th optimization.



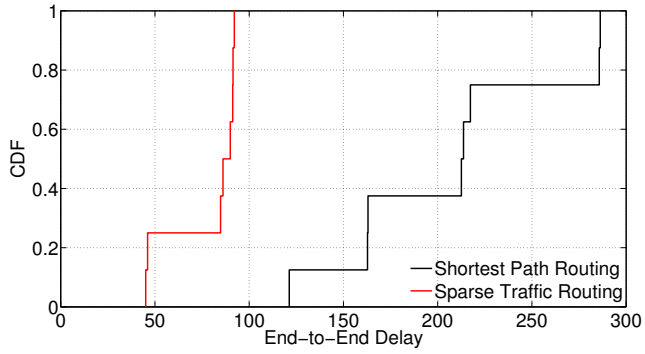
(f) Sessions 31-40 before/after the 4th optimization.

Fig. 6. A dynamic comparison of Shortest Path Routing and Sparse Traffic Routing (1), as 40 sessions join progressively over time.

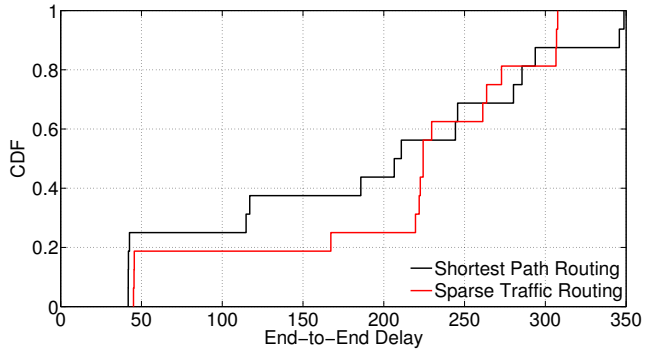
scenario, Sparse Traffic Routing has been triggered 4 times during the entire experiment.

Fig. 6(a), 6(c) and 6(e) show the average packet delays for sessions with different weights, right after the 2nd, 3rd and 4th Sparse Traffic Routing, respectively, as compared to always using Shortest Path Routing. From these figures, it is obvious that for sessions with higher weights w_i, our algorithm can effectively reduce their average packet delays compared to Shortest Path Routing. In Fig. 6(a), the delay benefit of Sparse Traffic Routing is relatively less significant, since at this early point the shortest links are not fully occupied yet, and most sessions can still go onto their desired paths. In Fig. 6(c), the advantages of Sparse Traffic Routing becomes more salient, especially for delay-sensitive video sessions with higher w_is. Fig. 6(e) evidently shows that our solution results in considerably lower average packet delays than Shortest Path Routing for delay-sensitive sessions with high w_is. It is worth noting that some low-weight sessions actually have longer delays under our scheme. This conforms to our intention of service differentiation, as background delay-insensitive flows have been placed on alternative longer paths to yield way to delay-sensitive sessions. Such a fine-grained service differentiation dictated by session weights is not possible with MLPS TE using Shortest Path Routing.

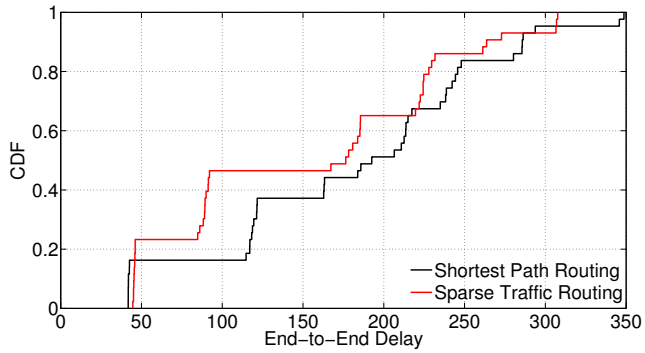
Fig. 6(b), 6(d) and 6(f) compare the average packet delays of the newly joined sessions *before and after* the next re-routing decisions made by Sparse Traffic Routing. Clearly, after each optimization, the average packet delays of high-weight sessions decrease significantly. Similarly, the benefit of our solution becomes more significant as time passes, which



(a) Sessions with $w_i \geq 26$



(b) Sessions with $w_i \leq 7$



(c) All 40 Sessions

Fig. 7. End-to-end packet delays for Sparse Traffic Routing (1) and Shortest Path Routing.

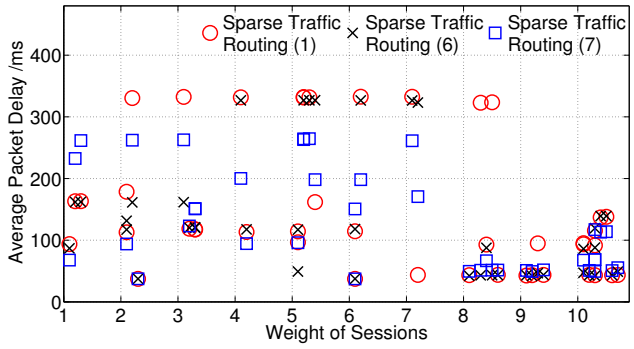


Fig. 8. Comparison among different Sparse Traffic Routing schemes (1), (6) and (7).

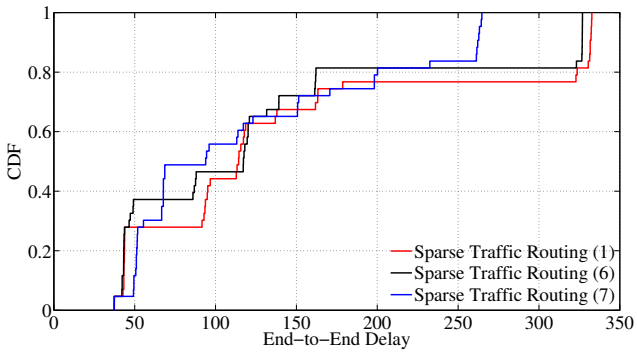


Fig. 9. Packet delays of different Sparse Traffic Routing schemes (1), (6) and (7).

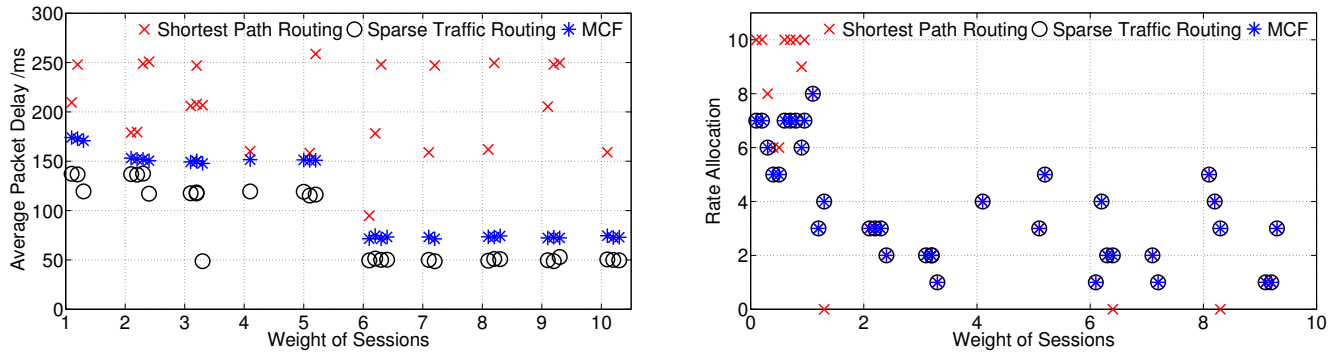
complies with the observations in Fig. 6(a), 6(c) and 6(e). Meanwhile, the sessions that join between two optimizations only need to suffer a relatively high delay for a short period of time. As soon as the next optimization has been executed, delay-sensitive sessions will be directed to shorter paths.

Fig. 7 plots the CDF of measured end-to-end packet delays for sessions with different weights. Fig. 7(a) shows that the packet delays of delay-sensitive sessions with $w_i \geq 26$, are much reduced due to the prioritization effect of our scheme. In contrast, Fig. 7(b) confirms our expectation that for delay-insensitive sessions with $w_i \leq 7$, our algorithm has a greater chance to incur a larger packet delay than Shortest Path Routing to make direct paths available for delay-sensitive sessions. And Fig. 7(c) illustrates that our algorithm evidently outperforms Shortest Path Routing.

To evaluate the sparsity of our solution in terms of path selection, Table II lists the number of sessions with traffic splitting, i.e., the sessions that have been routed on multiple paths/trees by the optimization. Recall that traffic splitting not only incurs overhead for traffic engineering and routing, but can also lead to additional packet reordering latency in live video streaming sessions. Therefore, it is desirable to select a single path/tree for each delay-sensitive session. From Table II, we can see that at most 5 out of all the 40 sessions have traffic splitting, while most sessions are still routed on a single path or tree. Furthermore, in each optimization, Sparse Traffic Routing does not choose the same sessions to routed on multiple paths. In other words, all the sessions are treated fairly, taking turns in terms of traffic splitting. In general, the results in Table II have demonstrated the effectiveness of using $\{0, 1\}$ sparsity regularizers $\phi(\cdot)$ in our optimization formulation.

D. Latency-Based vs. Hop-Based Optimizations

In this experiment, we have randomly generated all the parameters including requested target rates and weights of the sessions. We also have 40 sessions coexisting in the Amazon EC2 inter-DC network. However, we only focus on the results from one optimization this time, since we aim to compare the performance of different Sparse Traffic Routing solutions (1), (6) and (7) with different objective functions.



(a) Delay performance

(b) Rate allocation

Fig. 10. Performance comparison among Shortest Path Routing, MCF and Sparse Traffic Routing (8).

TABLE II
NUMBER OF TREES/PATHS SELECTED IN EXPERIMENT 1 WITH SPARSE TRAFFIC ROUTING (1).

Weight	1	4	7	8	13	18	21	26	29	39	total
Optimization 1	0/1/1	0/3/3	0/0/0	0/0/0	0/1/1	0/3/3	0/1/1	0/0/0	0/1/1	0/0/0	0/10/10
Optimization 2	0/2/2	1/6/7	0/0/0	0/1/1	0/2/2	0/5/5	0/3/3	0/0/0	0/1/1	0/0/0	1/20/21
Optimization 3	0/3/3	0/6/6	0/1/1	0/2/2	1/3/5	0/5/5	0/4/4	0/1/1	0/4/4	0/1/1	1/30/32
Optimization 4	0/3/3	1/7/8	1/3/4	0/3/3	0/3/3	1/7/8	2/6/8	0/2/2	0/4/4	0/2/2	5/40/45

Note: the values 1)/2)/3) are: 1) the number of sessions with traffic splitting, 2) the number of sessions in this weight range, 3) the total number of paths/trees adopted by these sessions.

TABLE III
NUMBER OF TREES/PATHS SELECTED IN EXPERIMENT 2 WITH SPARSE TRAFFIC ROUTING (1), (6) AND (7).

Weight	1	2	3	4	5	6	7	8	9	10	total
Sparse Traffic Routing (1)	0/3/3	0/3/3	0/4/4	0/2/2	0/5/5	0/3/3	0/2/2	3/6/9	0/4/4	0/8/8	3/40/43
Sparse Traffic Routing (6)	0/3/3	0/3/3	0/4/4	0/2/2	0/5/5	0/3/3	1/2/3	0/6/6	0/4/4	0/8/8	1/40/41
Sparse Traffic Routing (7)	0/3/3	0/3/3	0/4/4	0/2/2	1/5/6	1/3/4	0/2/2	0/6/6	0/4/4	0/8/8	2/40/42

Note: the values 1)/2)/3) are: 1) the number of sessions with traffic splitting, 2) the number of sessions in this weight range, 3) the total number of paths/trees adopted by these sessions.

Fig. 8 shows a similarity of the formulations (1), (6) and (7) in terms of the measured average packet delays. From Fig. 8, we can easily tell that packet delays achieved by (1) mostly lie very close to those achieved by (6) and (7). It is worth noting that the tree-depth-based formulation (7), although only based on counting hops without measuring link latencies, can yield good performance as compared to formulations (1) and (6) which require latency measurements, verifying the usefulness of hop-based formulation (7). This is because the latency on each path is highly correlated to the number of hops along the path, especially in inter-DC networks. Fig. 9 further plots the CDF of packet delays under formulations (1), (6) and (7). The formulation (6) performs better at the lower end of packet delays, while (7) outperforms the other two schemes at the higher end. Generally speaking, their performance is similar.

Table III shows that all three formulations (1), (6) and (7) have the ability to generate sparse path selection, yielding only a small number of sessions with traffic splitting, while most sessions have only adopted one tree/path.

E. Flow Competition under Insufficient Bandwidth

In the third experiment, we consider the rare scenario of insufficient bandwidth and aim to maximize the total allocated rates according to session priorities while minimizing packet delays for high-weight sessions. In order to simulate insufficient bandwidth (which rarely happens in the Amazon EC2 inter-DC network), we assume that the available bandwidth on each link in the Amazon EC2 inter-DC is capped by *half of its actual available bandwidth*. We assume that there exist some bandwidth-hungry sessions, which are large in size, but have little requirements on latency performance. Therefore, their weights are set close to 0, which means that they are delay insensitive.

We test three routing algorithms, Shortest Path Routing, Sparse Traffic Routing (8) and MCF, when 40 sessions are transferring data in the network at the same time. Here, we want to compare delay, throughput, and path sparsity of these three schemes.

Fig. 10(a) clearly shows that the routing decision made by Sparse Traffic Routing (8) performs the best for high-weight sessions (with $w_i \geq 1$) in terms of packet latency. It is worth noting that some background flows, with w_i close to 0, actually suffer longer packet delays under Sparse Traffic Routing (8) and MCF. The reason is that, even if background sessions come into the network first and use up direct paths, Sparse Traffic Routing and MCF algorithm will re-route background sessions to other paths, so that the shortest path is reserved

TABLE IV
NUMBER OF TREES/PATHS SELECTED IN EXPERIMENT 3 WITH SPARSE TRAFFIC ROUTING (8) AS COMPARED TO MULTI-COMMODITY FLOW (MCF).

Weight	close to 0	1	2	3	4	5	6	7	8	9	10	total
Sparse Traffic Routing (8)	6/10/16	0/3/3	0/4/4	0/4/4	0/1/1	1/3/4	0/4/4	0/2/2	0/3/3	0/3/3	0/3/3	7/40/47
MCF	10/10/30	3/3/6	0/4/4	0/4/4	0/1/1	3/3/6	0/4/4	0/2/2	0/3/3	0/3/3	0/3/3	16/40/66

Note: the values 1)/2)/3) are: 1) the number of sessions with traffic splitting, 2) the number of sessions in this weight range, 3) the total number of paths/trees adopted by these sessions.

for delay sensitive sessions.

A similar prioritization phenomenon is observed for the allocated rates. Fig. 10(b) shows the rate allocated to each session when they compete for bandwidth. While Shortest Path Routing may allocate 0 rates to some delay-sensitive sessions with high w_is that arrive later than background flows, Sparse Traffic Routing (8) and MCF will always satisfy high-weight sessions first and then allocate the remaining capacity to accommodate low-weight background flows as much as possible.

Though there are not too much differences between Sparse Traffic Routing (8) and MCF in terms of the average packet delay and rate allocation, Sparse Traffic Routing (8) greatly outperforms MCF in terms of path sparsity. As shown in Table IV, for all the 40 sessions, Sparse Traffic Routing (8) only splits traffic onto 2 paths for 7 sessions, while 6 of them are background flows. This is reasonable since wherever there is available bandwidth, we would split background flows onto different paths to maximize their throughput and fully utilize network resource. In contrast, MCF splits 16 sessions onto multiple paths, while only 8 of them are background flows. In other words, MCF does not minimize the number of paths assigned for each session; 8 delay-sensitive sessions have experienced traffic splitting and had to suffer from packet reordering. This demonstrates the unique strength of our solution to generate sparse routing solutions, while achieving even lower packet delays than MCF (for delay-sensitive sessions) and the same rate allocation as MCF, as shown in Fig. 10.

VII. CONCLUDING REMARKS

In this paper, we study the problem of optimal inter-DC traffic routing, targeting a large volume of delay-sensitive video traffic with stringent yet diverse latency requirements, transmitted among other types of flows between datacenters. Since inter-DC networks are usually over-provisioned to accommodate the aggregate requested flow rates at most times, we explicitly focus on minimizing the packet delays according to the diverse delay-sensitivity levels of different sessions, which can prioritize and improve the performance of delay-sensitive video streaming traffic. We propose a sparse traffic routing solution that selects a single path for most delay-sensitive sessions to limit the overhead of fine-grained traffic splitting and packet reordering at the destination.

We have implemented our proposed delay-optimized sparse routing technique in an application-layer software-defined network, which enables traffic engineering from a global point of view without modifying the current network infrastructure. Unlike switch-level SDN, our system can scale to a large number of forwarding rules and thus accommodates a large number of sessions. Extensive experiments performed on Amazon EC2 datacenters have shown that our solution can effectively reduce packet delays for time-sensitive video sessions according to various urgency levels, even when they are transmitted among a diverse range of other types of flows, with little overhead.

ACKNOWLEDGMENT

The co-authors would like to acknowledge the gracious research support from the NSERC Discovery Research Program and the SAVI NSERC Strategic Networks Grant at the University of Toronto.

REFERENCES

[1] A. Cockcroft, "Netflix in the Cloud," 2011.
[2] "Periscope," https://www.periscope.tv/.
[3] "Meerkat," http://meerkatapp.co/.
[4] "FaceTime," https://en.wikipedia.org/wiki/FaceTime/.
[5] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving High Utilization with Software-Driven WAN," in *Proc. ACM SIGCOMM*, 2013.
[6] A. Pathak, M. Zhang, Y. C. Hu, R. Mahajan, and D. Maltz, "Latency Inflation with MPLS-Based Traffic Engineering," in *Proc. ACM SIGCOMM*, 2011.
[7] A. Elwalid, C. Jin, S. Low, and I. Widjaja, "MATE: MPLS Adaptive Traffic Engineering," in *Proc. IEEE INFOCOM*, 2001.
[8] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a Globally-Deployed Software Defined Wan," in *Proc. ACM SIGCOMM*, 2013.
[9] E. Danna, S. Mandal, and A. Singh, "A Practical Algorithm for Balancing the Max-Min Fairness and Throughput Objectives in Traffic Engineering," in *Proc. IEEE INFOCOM*, 2012.
[10] M. Fazel, H. Hindi, and S. Boyd, "Log-Det Heuristic for Matrix Rank Minimization with Applications to Hankel and Euclidean Distance Matrices," in *Proc. American Control Conference*, 2003.
[11] M. Fazel, "Matrix Rank Minimization with Applications," Ph.D. dissertation, 2002.
[12] "OpenFlow Specification," http://www.openflow.org/wp/documents/, 2012.
[13] Y. Wu, C. Wu, B. Li, L. Zhang, Z. Li, and F. Lau, "Scaling Social Media Applications into Geo-Distributed Clouds," in *Proc. IEEE INFOCOM*, 2012.
[14] R. Krishnan, H. V. Madhyastha, S. Srinivasan, S. Jain, A. Krishnamurthy, T. Anderson, and J. Gao, "Moving Beyond End-to-End Path Information to Optimize CDN Performance," in *Proc. ACM SIGCOMM*, 2009.
[15] V. K. Adhikari, S. Jain, and Z.-L. Zhang, "YouTube Traffic Dynamics and Its Interplay with a Tier-1 ISP: an ISP Perspective," in *Proc. ACM SIGCOMM*, 2010.
[16] X. Chen, M. Chen, B. Li, Y. Zhao, Y. Wu, and J. Li, "Celerity: a Low-Delay Multi-Party Conferencing Solution," in *Proc. ACM Multimedia*, 2011.
[17] M. Chen, M. Ponec, S. Sengupta, J. Li, and P. A. Chou, "Utility Maximization in Peer-to-Peer Systems," in *Proc. ACM SIGMETRICS*, 2008.
[18] J. Li and P. A. Chou, "Mutualcast: an Efficient Mechanism for Content Distribution in a Peer-to-Peer (P2P) Network," Microsoft Research, Tech. Rep. MSR-TR-2004-98, September 2004.
[19] M. Ponec, S. Sengupta, M. Chen, J. Li, and P. A. Chou, "Multi-Rate Peer-to-Peer Video Conferencing: A Distributed Approach Using Scalable Coding," in *Proc. IEEE International Conference on Multimedia and Expo*, 2009.

[20] Y. Feng, B. Li, and B. Li, "Airlift: Video Conferencing as a Cloud Service Using Inter-Datacenter Networks," in *Proc. IEEE Internet Conference on Network Protocol*, 2012.
[21] ——, "Jetway: Minimizing Costs on Inter-Datacenter Video Traffic," in *Proc. ACM Multimedia*, 2012.
[22] Y. Chen, S. Jain, V. Adhikari, Z.-L. Zhang, and K. Xu, "A First Look at Inter-Data Center Traffic Characteristics via Yahoo! Datasets," in *Proc. IEEE INFOCOM*, 2011.
[23] D. O. Awduche and J. Agogbua, "Requirements for Traffic Engineering over MPLS," RFC 2702, September, Tech. Rep., 1999.
[24] M. Meyer and J. Vasseur, "MPLS Traffic Engineering Soft Preemption," RFC 5712, January, Tech. Rep., 2010.
[25] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic Flow Scheduling for Data Center Networks," in *Proc. USENIX NSDI*, 2010.
[26] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, "Managing Data Transfers in Computer Clusters with Orchestra," in *Proc. ACM SIGCOMM*, 2011.
[27] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for Network Update," in *Proc. ACM SIGCOMM*, 2012.
[28] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling Flow Management for High-Performance Networks," in *Proc. ACM SIGCOMM*, 2011.
[29] T. Benson, A. Anand, A. Akella, and M. Zhang, "MicroTE: Fine Grained Traffic Engineering for Data Centers," in *Proc. ACM International Conference on emerging Networking EXperiments and Technologies*, 2011.
[30] J. Jiang, T. Lan, S. Ha, M. Chen, and M. Chiang, "Joint VM Placement and Routing for Data Center Traffic Engineering," in *Proc. IEEE INFOCOM*, 2012.
[31] T. Benson, A. Anand, A. Akella, and M. Zhang, "The Case for Fine-Grained Traffic Engineering in Data Centers," in *Proc. IEEE INFOCOM*, 2012.
[32] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: a New Resource ReSerVation Protocol," *IEEE Network*, 1993.
[33] R. Braden, D. Clark, and S. Shenker, "Integrated Services in the Internet Architecture: an Overview," RFC 1633, June, Tech. Rep., 1994.
[34] M. Carlson, W. Weiss, S. Blake, Z. Wang, D. Black, and E. Davies, "An Architecture for Differentiated Services," RFC 2475, December, Tech. Rep., 1998.
[35] E. Crawley, H. Sandick, R. Nair, and B. Rajagopalan, "A Framework for QoS-Based Routing in the Internet," RFC 2386, August, Tech. Rep., 1998.
[36] R. Callon and E. C. Rosen, "Multiprotocol Label Switching Architecture," RFC 3031, January, Tech. Rep., 2001.

Yinan Liu received the B.Engr. degree from the Department of Telecommunications Engineering, Xidian University, China, in 2010. She is currently a M.A.Sc. student in the Department of Electrical and Computer Engineering at the University of Toronto. Her general research interests cover software-defined networking, inter-datacenter networks, and traffic engineering.

Di Niu received the B.Engr. degree from the Department of Electronics and Communications Engineering, Sun Yat-sen University, China, in 2005 and the M.A.Sc. and Ph.D. degrees from the Department of Electrical and Computer Engineering, University of Toronto, Toronto, Canada, in 2009 and 2013. Since 2012, he has been with the Department of Electrical and Computer Engineering at the University of Alberta, where he is currently an Assistant Professor. His research interests span the areas of cloud computing and storage, multimedia delivery systems, data mining and statistical machine learning for social and economic computing, distributed and parallel computing, and network coding. He is a member of IEEE and ACM.

Baochun Li received the B.Engr. degree from the Department of Computer Science and Technology, Tsinghua University, China, in 1995 and the M.S. and Ph.D. degrees from the Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, in 1997 and 2000. Since 2000, he has been with the Department of Electrical and Computer Engineering at the University of Toronto, where he is currently a Professor. He holds the Nortel Networks Junior Chair in Network Architecture and Services from October 2003 to June 2005, and the Bell Canada Endowed Chair in Computer Engineering since August 2005. His research interests include large-scale multimedia systems, cloud computing, peer-to-peer networks, applications of network coding, and wireless networks. Dr. Li was the recipient of the IEEE Communications Society Leonard G. Abraham Award in the Field of Communications Systems in 2000. In 2009, he was a recipient of the Multimedia Communications Best Paper Award from the IEEE Communications Society, and a recipient of the University of Toronto McLean Award. He is a member of ACM and a Fellow of IEEE.