Scalable Multi-Class Traffic Management in Data Center Backbone Networks

Amitabha Ghosh, Sangtae Ha, Edward Crabbe, and Jennifer Rexford

***Abstract*—Large online service providers (OSPs) often build private backbone networks to interconnect data centers in multiple locations. These data centers house numerous applications that produce multiple classes of traffic with diverse performance objectives. Applications in the same class may also have differences in relative importance to the OSP's core business. By controlling both the hosts and the routers, an OSP can perform both application rate-control and network routing. However, centralized management of both rates and routes does not scale due to excessive message-passing between the hosts, routers, and management systems. Similarly, fully-distributed approaches do not scale and converge slowly. To overcome these issues, we investigate two semi-centralized designs that lie at practical points along the spectrum between fully-distributed and fully-centralized solutions. We achieve scalability by distributing computation across multiple tiers of an optimization machinery. Our first design uses two tiers, representing the backbone and classes, to compute class-level link bandwidths and application sending rates. Our second design has an additional tier representing individual data centers. Using optimization, we show that both designs provably maximize the aggregate utility over all traffic classes. Simulations on realistic backbones show that the 3-tier design is more scalable, but converges slower than the 2-tier design.**

***Index Terms*—Traffic management; data centers; optimization; scalability; multiple traffic classes.**

I. Introduction

A. Wide-Area Traffic Management for OSPs

RECENT years have seen an unprecedented growth in the number of data centers being deployed by large OSPs [1]–[5]. These data centers store massive amounts of data and host a variety of services. Furthermore, to improve performance and reliability, multiple data centers are deployed to cover large geographical regions with high-speed backbone networks interconnecting them [6], [7]. These backbones are often owned by the same OSP; for instance, Google, Yahoo!, and Microsoft interconnect their multiple large-scale data centers with their own private backbones [8]. As the backbones themselves represent a substantial investment, it is highly desirable that they be used efficiently and effectively, while simultaneously respecting the characteristics of the traffic they carry.

Manuscript received December 15, 2012; revised July 31, 2013.

A. Ghosh is with the Networking Division at UtopiaCompression, and did this work as a post-doctoral research associate at Princeton University (e-mail: amitabhg@utopiacompression.com).

S. Ha is with the Electrical Engineering Department at Princeton University (e-mail: sangtaeh@princeton.edu).

E. Crabbe is with Google (e-mail: edc@google.com).

J. Rexford is with the Computer Science Department at Princeton University (e-mail: jrex@cs.princeton.edu).

Digital Object Identifier 10.1109/JSAC.2013.1312xx.

The types of services and applications hosted in these OSP networks can be quite diverse. In some cases, the same OSP which runs the backbone network also controls the traffic sources. The applications can range from back-end services, such as background computation, search indexing, and data replication, to client-triggered front-end services for creating end-user content [9], such as web search, online gaming, and live video streaming. Each of these applications can belong to a different class of traffic that has its own *performance objectives*. For example, interactive applications are likely to be delay-sensitive, whereas back-end applications can be throughput-sensitive. Applications that belong to the same traffic class may also have differences in *relative importance* to the OSP's core business.

Despite many challenges, these OSP network characteristics provide new opportunities for wide-area traffic management. In traditional traffic engineering (TE), an ISP controls backbone routing and link scheduling, but does not control how end hosts should perform congestion control. In contrast, an OSP controls both the hosts and the routers, offering the flexibility to jointly optimize rate control, backbone routing, and link scheduling. Not knowing the end-to-end performance objectives of the applications, ISPs typically focus on optimizing indirect measures of performance, such as minimizing congestion. In contrast, an OSP can maximize the aggregate performance across all applications, with a different utility function for each traffic class.

Centralized traffic engineering is increasingly viewed as a viable approach for wide-area networks. For instance, Google has deployed an OpenFlow-enabled Software Defined Networking (SDN) solution for centralized TE in their G-scale internal backbone for interconnecting multiple data centers [10]. Centralized TE solutions have the potential to be algorithmically simpler, faster, scalable, and more efficient. However, they are not scalable for joint rate and route control across multiple classes, due to the need for excessive message-passing between hosts, routers, and management systems. Fully-distributed solutions, likewise, are also not scalable and exhibit slow convergence.

B. Semi-Centralized Scalable Design Choices

In this paper, we explore scalable architectures that jointly optimize rate control, routing, and link scheduling. On one hand, our design choices are motivated by the advantages of centralized TE and its industry adoption. On the other hand, since we also wish to perform rate control, our approach distributes information and computation across multiple tiers of an optimization machinery. To this end, we examine two

semi-centralized designs that both use a small number of management entities to optimally allocate resources, but differ in their degree of *distributedness*. The first design has two tiers with the management entities at the backbone and class levels, whereas the second design has three tiers with an additional management entity at the data center level. The entities exchange information with each other to optimally subdivide network bandwidth between applications of different classes. Using optimization theory, we show that both our designs provably maximize the aggregate utility over all applications and traffic classes.

We summarize our two designs below:

A 2-Tier Design: The 2-tier design has a single management entity called *link coordinator* (LC) on the first tier, and multiple management entities called *class allocators* (CAs) on the second tier. The LC computes class-level aggregate bandwidth for every link in the backbone and sends it to the CAs. Each CA then subdivides this bandwidth between different applications in its own class. Using primal decomposition, we show that this 2-tier design not only can compute optimal routing paths, but also map application-level sending rates on these paths. Simulations on realistic backbone topologies show that the system converges quickly (within a few tens of iterations for suitable choices of step sizes), though at the expense of moderate amount of message-passing between the management entities.

A 3-Tier Design: The 3-tier design has an additional type of management entity called *data center allocator* (DA) on the third tier. The LC, as before, computes class-level aggregate bandwidth and sends it to the CAs. Each CA, however, now subdivides this bandwidth not between different applications like in the 2-tier design, but across multiple data centers hosting traffic in its own class. The task of computing application-level sending rates is delegated to the DAs. We use a two-level primal decomposition to prove optimality of this 3-tier design. Simulations show that the system exchanges fewer messages compared to the 2-tier design, but at the expense of slightly slower convergence.

The rest of the paper is organized as follows: In Section II, we present the network and data center traffic models, and formulate the multi-class backbone traffic management problem. In Section III, we describe the components and functionalities of the two semi-centralized designs. In Section IV, we prove the optimality of these designs using the theory of optimization decomposition. In Section V, we compare the two designs in terms of their convergence behavior and the number of messages exchanged until convergence. We present related work in Section VI, and finally conclude in Section VII.

II. Multi-Class Traffic-Management Problem

In this section, we first describe our inter-data center traffic model and backbone network model, and then formulate the multi-class traffic-management problem. We also introduce our key notations, as summarized in Table I.

A. Inter-Data Center Traffic Model

1) Traffic Classes and Flows: We consider a set $\mathcal{J}$ of data centers, geographically-distributed and interconnected over a backbone network. The backbone carries inter-data center traffic flowing between different source and destination hosts located in the data centers. This traffic can be categorized into a set $\mathcal{K}$ of distinct *classes* based on its performance objectives. For instance, some of this traffic can belong to a throughput-sensitive class, some to a delay-sensitive class, and some can have varying degrees of both throughput and delay requirements. Typically, however, the network will have just a few traffic classes. We index the data centers by j and the traffic classes by k. The cardinalities of the sets $\mathcal{J}$ and $\mathcal{K}$ are denoted by J and K, respectively.

Traffic within a given class can belong to several applications, each of which, in turn, may produce a large number of flows. For example, delay-sensitive traffic may include applications such as video streaming, Voice-over-IP (VoIP), and multimedia conferencing, and each of these can have multiple flows between different source-destination pairs. For simplicity of exposition, however, we do not differentiate between multiple applications within a class, but, instead, refer to the traffic between a given source-destination pair in a particular direction as a *flow*, regardless of the application to which it belongs. For instance, a flow can be at the granularity of an individual UDP session, or at an aggregate level between two end hosts or subnets in a particular direction. We index the flows by s, and denote by $\mathcal{F}$ the set of all flows. We use $\mathcal{F}^k$ to denote the set of flows that belongs to a particular class k.

2) Utility Functions and Weights: We characterize the performance objective of each class k by a utility function $U^k(\cdot)$, parameterized by two positive coefficients a^k and b^k. Thus, all the flows of a given class have the same utility function. We design this class-wise utility function as having two parts in it, $f^k(\cdot)$ and $g^k(\cdot)$, which are multiplied by the coefficients a^k and b^k, respectively, to incorporate different degrees of sensitivity to throughput and delay. We model throughput-sensitivity by making the first part, $f^k(\cdot)$, dependent on the total sending rate, and model delay-sensitivity by making the second part, $g^k(\cdot)$, dependent on the average end-to-end delay. The arguments of these functions are described later in this section.

The two coefficients can take different values to model different types of traffic. For instance, a fixed rate traffic with delay requirement (e.g., VoIP) can be modeled using $a^k = 0$ and $b^k > 0$, whereas a variable rate traffic with delay requirement (e.g., multimedia streaming) can be modeled using both $a^k > 0$ and $b^k > 0$. Likewise, a variable rate traffic with no delay requirement (e.g., database backup or file-downloading) can use $a^k > 0$ and $b^k = 0$.

Modeling Flow Importance: Despite all the flows of a given class sharing the same utility function, the individual flows may have differences in relative importance. For example, two database back-up flows that belong to a throughput-sensitive class, or two video streaming flows that belong to a delay-sensitive class, may have different business importance. We model this relative importance by assigning a positive weight to each flow, denoted by w_s^k for flow s of class k.

The weighted utility of flow s of class k can now be written

TABLE I
TABLE OF KEY NOTATIONS.

Symbol	Description
$\mathcal{L}$	Set of unidirectional links in the backbone.
$\mathcal{K}$	Set of distinct traffic classes.
$\mathcal{J}$	Set of data centers.
$\mathcal{P}$	Set of available paths.
$\mathcal{F}$	Set of all flows across all classes.
$\mathcal{F}^k$	Set of flows in class k.
c_l	Capacity of link l.
w_s^k	Weight of flow s of class k.
z_{sp}^k	Rate of flow s of class k on its p^{th} path.
y_l^k	Bandwidth allocated for class k on link l.
$U^k(\cdot)$	Utility function of class k.
a^k, b^k	Parameters to model throughput and delay sensitivity of traffic in class k.
d_l^k	Average queuing delay on link l experienced by traffic of class k.
u_l^k	Utilization of link l due to traffic of class k.
$\mathbf{A}$	Topology matrix.
$\mathbf{R}$	Path routing matrix.

as:

$$U_s^k = w_s^k \Big[a^k f^k(\cdot) - b^k g^k(\cdot) \Big], \tag{1}$$

where the subtraction indicates that a flow gains more utility by lowering its average end-to-end delay.

Modeling Throughput-Sensitivity: We assume that the rate-dependent function $f^k(\cdot)$ is non-negative, strictly concave, non-decreasing, and twice differentiable in the total sending rate x_s^k of flow s of class k. For example, the function can be logarithmic, such as $\log\left(x_s^k\right)$, or belong to a more general class of functions that subsumes different notions of "rate-fairness," typically used for Internet TCP congestion control (e.g., max-min fairness, proportional fairness, α-fairness, etc.) [11], [12]. The concavity of $f^k(\cdot)$ is justified because of diminishing returns of the resources allocated to the flows. In other words, the first unit of bandwidth allocated to a flow is often more valued than every additional unit, especially when the flow already has an adequate bandwidth allocation. We also assume that each flow has an infinite backlog.

Modeling Delay-Sensitivity: The second part $g^k(\cdot)$ of the utility function is dependent on the average end-to-end delay for traffic in class k. We consider this average end-to-end delay for a given path to be the sum of the propagation delays and the average queuing delays for all the links on that path. We choose to minimize the *average* delay because of two reasons: (1) it results in lower delay for more traffic than minimizing some other metric, for instance, the maximum delay along the paths; and (2) for mathematical convenience, which allows us to formulate the problem as a convex optimization problem and solve it more easily. We do not claim to model the queuing delay accurately, but, instead, this is our way of keeping the links below high load, while favoring paths with low propagation delay.

The average queuing delay experienced by a flow on a given physical link depends on several factors, such as the number of queues, the link-scheduling policy, and the bandwidth allocated on that link. In this work, we assume that each link maintains one queue per class, and configures aggregate link bandwidth on a per-class basis. To this end, the links employ a *non-work-conserving* scheduler. The reason we choose such a scheduler instead of a work-conserving one is as follows: Suppose there are two flows that share a common link, but belong to two different classes A and B, as shown in Figure 1. If link 1 is idle, then a work-conserving scheduler will transmit additional packets for the flow of class A. This will cause additional congestion on link 3, and incorrectly signal that class A traffic needs more bandwidth on link 3, leading to a redistribution of resources with lesser bandwidth allocated to class B.

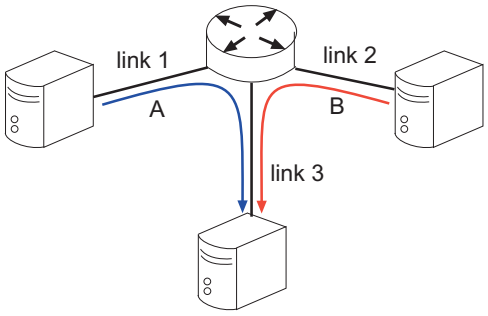

Fig. 1. Queue scheduling policies: A work-conserving scheduler would transmit additional packets for the flow of class A if link 1 is idle, thus, causing additional congestion on the common link 3. A non-work-conserving scheduler would not transmit such additional packets.

Suppose d_l^k denotes the average queuing delay experienced by a packet of class k on link l. The queuing delay is a function of the link load due to the traffic of class k, and the aggregate bandwidth on link l allocated to class k. This class-level bandwidth can be thought as the class having its own virtual link with a capacity equal to the aggregate bandwidth allocated on the physical link. We denote the link load by L_l^k, and the class-level bandwidth by y_l^k. We choose the form of the delay function in such a way that it penalizes links that approach or exceed their capacity. For mathematical convenience, we also want this delay function to be non-negative, convex, non-decreasing, and twice-differentiable. Thus, instead of using the M/M/1 queuing formula for average delay, $d_l^k = 1/(y_l^k - L_l^k)$, which is undefined for over-utilized links, we use a piecewise linear approximation to it, following the approach in [13]. Denoting the utilization of link l due to the traffic of class k by u_l^k (i.e., $u_l^k = L_l^k/y_l^k$), we define this piecewise linear function as:

$$d_l^k\left(u_l^k\right) = m_l^k L_l^k + q_l^k, \tag{2}$$

where $m_l^k = m(u_l^k)/y_l^k$ is a function that determines the slope of the region for class k, and $q_l^k = q(u_l^k)/y_l^k$ is a function that determines the associated y-intercept for class k. Like in [13], the values of d_l^k are small for low utilizations, but increase rapidly as the load approaches or exceeds the link capacity.

B. Backbone Network Model

1) Backbone Topology and Paths: We model the backbone network as a set $\mathcal{L}$ of interconnecting unidirectional links, with a finite capacity c_l, and a propagation delay p_l for every

link $l \in \mathcal{L}$. To support large volumes of data transfer between remote data center locations, in our model, we consider the links to typically have high capacities.

Let a path p be a non-empty subset of $\mathcal{L}$, and let $\mathcal{P}$ be the set of available paths. The set $\mathcal{P}$ does not necessarily include all possible paths in the physical topology, but only a subset of them chosen by operators. Since the backbone comprises long-haul, high-capacity links, most data center pairs have only a few paths connecting them. These paths are set up in advance by OSP operators using, for instance, MPLS (Multi-Protocol Label Switching) labels. Each path emerges from an edge router in a source data center, and passes through intermediate routers before ending at another edge router in a destination data center.

2) Flow Rates and Multi-Path Routing: We assume that each flow can use multiple paths from its source to destination end host. Multi-path routing for inter-data center traffic is relatively easy to implement, because the backbone as well as the sources are owned by the same OSP. Multi-path routing provides several benefits by splitting high-volume traffic along multiple paths, thereby increasing the end-to-end capacity.

We denote by z_{sp}^k the sending rate of flow s of class k on its p^{th} path. We assume that each flow can be split flexibly, i.e., the host or router originating the flow can send packets at the computed rates along each of the specified paths. Without proper handling [14], such flow-level multi-path routing might cause out-of-order delivery of packets. Conventional techniques, such as hash-based splitting of traffic, can prevent this out-of-order arrival of packets belonging to the same TCP or UDP session.

3) Routing Matrix: A routing matrix typically encodes the mapping between paths, links, and traffic flows in a network. For inter-data center routing, we split this single matrix into two parts to bring scalability in the resulting system. Since the number of flows per class can be very large compared to the number of paths and links, we store the mapping between paths and links in a smaller matrix $\mathbf{A}$, called the *topology matrix*, of size $|\mathcal{L}| \times |\mathcal{P}|$, and the mapping between flows and paths in a larger matrix $\mathbf{R}$, called the *path routing matrix*, of size $|\mathcal{P}| \times |\mathcal{F}|$. Thus, if a management server in the backbone needs information only about the links comprising the paths, but not about the flows on those paths, it can store only the smaller matrix $\mathbf{A}$. This saves memory and communication overhead that would be needed otherwise to keep a single matrix updated all the time due to dynamic arrival and departure of flows. The elements of $\mathbf{A}$ and $\mathbf{R}$ are:

$$A_{lp} = \begin{cases} 1, & \text{if link } l \text{ lies on path } p \\ 0, & \text{otherwise.} \end{cases}$$

$$R_{sp}^k = \begin{cases} 1, & \text{if flow } s \text{ of class } k \text{ uses path } p \\ 0, & \text{otherwise.} \end{cases}$$

Since a flow can go over multiple paths, we consider its weighted average delay as the sum of the products of path rates and average end-to-end delays on those paths. This captures the true average delay over all bits of traffic in the flow. Using (2), the delay-dependent function $g^k(\cdot)$ takes the following form for flow s of class k:

$$g^k\left(u_l^k\right) = \sum_{p \in \mathcal{P}} R_{sp}^k z_{sp}^k \left(\sum_{l \in \mathcal{L}} A_{lp} \left(p_l + m_l^k L_l^k + q_l^k \right) \right). \quad (3)$$

The link load is given by $L_l^k = \sum_{s \in \mathcal{F}^k} \sum_{p \in \mathcal{P}} A_{lp} R_{sp}^k z_{sp}^k$.

C. Maximizing Aggregate Utility

In an OSP network, the backbone and the traffic sources are under the same ownership, simplifying the task of choosing an objective function. As a natural choice, we consider that the OSP aims to maximize the sum of the utilities of all flows across all traffic classes, thus, maximizing the global "social welfare." In other words, the flows do not have conflicting objectives, but, instead, share a common goal prescribed by the OSP. The constraints in our problem are the link capacity constraints, and the variables are the flow path rates, z_{sp}^k, and the aggregate bandwidth, y_l^k, for each class k on link l. The optimization problem can thus be written as:

$$\begin{aligned} &\textit{GLOBAL:} \\ &\text{maximize} \quad \mathcal{U} = \sum_{k \in \mathcal{K}} \sum_{s \in \mathcal{F}^k} w_s^k \left[a^k f^k\left(x_s^k\right) - b^k g^k\left(u_l^k\right) \right] \\ &\text{subject to} \quad \sum_{s \in \mathcal{F}^k} \sum_{p \in \mathcal{P}} A_{lp} R_{sp}^k \, z_{sp}^k \le y_l^k, \;\; \forall k, l \\ &\qquad\qquad \sum_{k \in \mathcal{K}} y_l^k \le c_l, \;\; \forall l \\ &\text{variables} \quad z_{sp}^k \ge 0, \;\; \forall k, s, p \\ &\qquad\qquad y_l^k \ge 0, \;\; \forall k, l \end{aligned} \quad (4)$$

where $x_s^k = \sum_{p \in \mathcal{P}} R_{sp}^k \, z_{sp}^k$ is the total sending rate of flow s of class k over all its paths. The first constraint says that the total sending rate of all the flows of class k that go over link l cannot exceed the aggregate bandwidth y_l^k. The second constraint says that the total bandwidth allocated to all the classes on link l cannot exceed the capacity c_l. In the above form, (4) is a convex optimization problem, due to the linear inequality constraints and the concave objective function.

III. Semi-Centralized Scalable Designs

In this section, we present two semi-centralized designs that are both scalable and use a small number of management entities to optimally allocate flow-level sending rates across multiple classes in an OSP network. We describe the different components and their functionalities comprising the two designs, as well as the messages exchanged between the management entities.

A. A Semi-Centralized 2-Tier Design

Typically, data centers host applications that originate a large number of flows which belong to different traffic classes with diverse performance requirements. The flows can have different weights and can flow over multiple paths from their source to destination end hosts. In such a setting, centralized traffic management solutions suffer from scalability issues, due to the need for excessive message-passing between the hosts, routers, and management systems. Likewise, fully-distributed solutions are also not scalable and exhibit slow convergence.

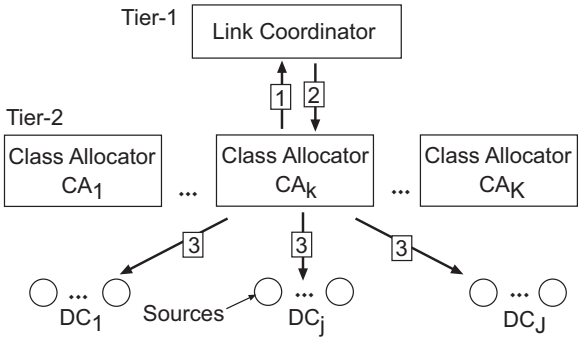

Fig. 2. A semi-centralized 2-tier design with two types of management entities: a single LC on tier-1, and multiple CAs on tier-2. The arrows indicate message-passing.

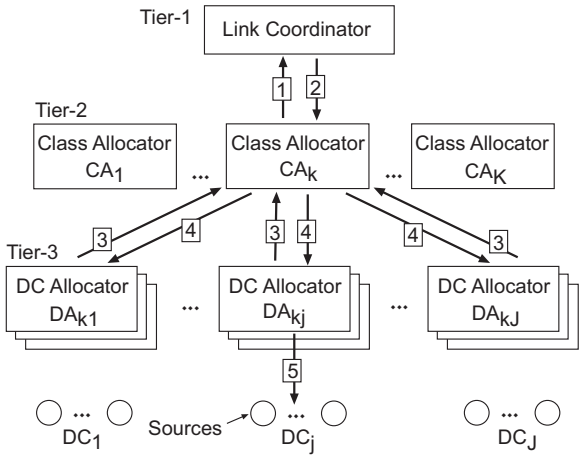

Fig. 3. A semi-centralized 3-tier design with three types of management entities: a single LC on tier-1, multiple CAs on tier-2, and multiple DAs on tier-3. The arrows indicate message-passing.

To overcome these drawbacks, we follow a top-down approach and propose a semi-centralized 2-tier design (see Figure 2) that is modular, scalable, and requires only moderate amount of message-passing. The design has two types of management entities on its two tiers. On tier-1, it has a single management entity called a *link coordinator* (LC), and on tier-2, there are multiple management entities called *class allocators* (CAs). There is one CA assigned for each class. These entities are hosted from one or multiple servers in the backbone. Each type of entity has limited knowledge about the backbone and inter-data center traffic, but can communicate with each other to optimize flow-level sending rates across multiple classes. In the following, we describe the functionalities of these components.

Link Coordinator on Tier-1: The LC is a centralized entity that knows only about the network topology. In particular, it knows the capacity c_l of every link l, and the topology matrix $\mathbf{A}$, but does not know the utility functions of the classes, the weights, or the paths traversed by the individual flows. The LC optimizes aggregate link bandwidths across multiple classes in the backbone. In particular, it computes an aggregate bandwidth y_l^k for each class k on every link l in the backbone. This aggregate bandwidth is shared by all the flows of class k that go over link l, and is called the *class-level* bandwidth. The LC then sends these y_l^ks to the CAs on tier-2 for driving flow rate allocation decisions. In turn, the LC receives a set of optimal Lagrange multipliers from each CA. This message-passing is shown in Figure 2 by the arrows labeled "1" and "2" between the LC and the CA of class k. The Lagrange multipliers, as described in the next section, are optimal subgradients of an optimization problem solved by each CA, and are functions of the y_l^ks.

Class Allocator on Tier-2: Unlike the LC, which has no information about the inter-data center traffic, each CA knows the utility function of its own class, the weights of the flows, and the paths traversed by them. These weights and paths are communicated to the CAs by the end hosts. There are a total of K CAs, one assigned for each class k, denoted by CA_k. Each CA optimizes sending rates across multiple flows that belong to its own class. In particular, the CA of class k first receives the class-level bandwidths y_l^ks from the LC, and then solves an optimization problem to subdivide these bandwidths among the flows of its own class. The CA then sends these optimal path rates to the sources hosting these flows, as shown by the arrows labeled "3" in Figure 2. It also sends the optimal Lagrange multipliers to the LC.

B. A Semi-Centralized 3-Tier Design

In our 2-tier design, each CA sends the flow-level path rates to the end hosts that belong to its own class. However, since the flows of a given class can be hosted from any of the data centers, each CA potentially communicates with *all* data centers. When the number of flows is very large, this incurs expensive control plane overhead. To overcome this drawback, we now propose a semi-centralized 3-tier design that is more scalable and has less message-passing overhead.

The 3-tier design has a third type of management entity called a *data center allocator* (DA) on tier-3, in addition to the LC on tier-1 and the CAs on tier-2 (see Figure 3). The functionality of the LC is the same as before, i.e., it optimizes aggregate link bandwidths across multiple classes in the backbone. The CAs and the DAs, however, perform different tasks. In particular, the task of allocating flow-level path rates for a given class is now split between a CA and multiple DAs assigned for that class. We describe the functionalities of the CAs and the DAs in the following.

Class Allocator on Tier-2: Each CA optimizes aggregate link bandwidths across multiple data centers. As before, there are a total of K CAs, one assigned for each class k, denoted by CA_k. Unlike the 2-tier design, where the CAs knew about the utility functions and the weights of the flows, in this 3-tier design, each CA knows only the paths traversed by the flows of its own class. The CA of class k first receives the class-level bandwidths y_l^ks from the LC, and then subdivides these among multiple data centers that host flows of class k. We call this the *DC-level* bandwidth and denote it by y_l^{kj}. This bandwidth is shared by the flows of class k that originate from data center j and go over link l. The CA then sends these y_l^{kj}s to the DAs on tier-3 for driving flow-rate allocation decisions. In turn, the CA receives a set of optimal Lagrange multipliers from each DA. These Lagrange multipliers are functions of the y_l^{kj}s. The message-passing between the CA and the DAs is

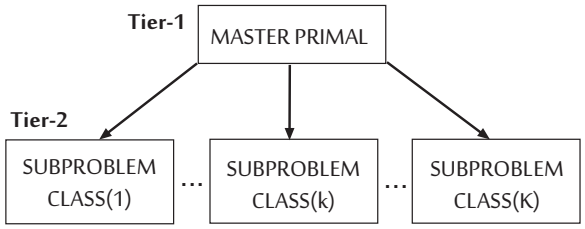

Fig. 4. A single-level primal decomposition of *GLOBAL* into a master primal on tier-1, and K class-level subproblems on tier-2.

shown in Figure 3 by the arrows labeled "3" and "4." Finally, each CA also sends the Lagrange multipliers to the LC after optimizing the DC-level bandwidths, as shown by the arrows labeled "1."

Data Center Allocator on Tier-3: The DAs lie on tier-3 and are hosted by servers located within the data centers. There is one DA assigned for each class in each data center, summing up to a total of KJ DAs. We denote the DA for class k in data center j by DA_{kj}. Each DA has a localized view of the inter-data center traffic. It only knows the utility function of its class, and the weights and paths of the flows that originate from its own data center. Each DA optimizes sending rates across multiple flows in the same data center. In particular, DA_{kj} first receives the DC-level bandwidths y_l^{kj}s from the CA, and then optimally subdivides these among the flows of class k that originate from data center j. It then sends these optimal path rates to the respective sources, shown by the arrow labeled "5" in Figure 3, as well as the optimal Lagrange multipliers to the CA from which it receives the y_l^{kj}s.

IV. Optimization Decomposition

In this section, we describe the optimization decomposition underlying the two designs to prove optimality of the traffic management protocols. We first present a single-level primal decomposition for the 2-tier design, and then a two-level primal decomposition of the 3-tier design. We also describe the associated message-passing, and the rate allocation updates by the different management entities.

A. Decomposing the 2-Tier Design

In our 2-tier design, the class-level bandwidths are first computed by the LC on tier-1, and then are subdivided by the CAs on tier-2 to compute flow-level path rates. This particular rate allocation strategy suggests a decomposition of *GLOBAL* into multiple smaller subproblems, each of which can be solved independently by a CA, while together being coordinated by the LC. Such a decomposition, where the primary resources (i.e., bandwidths) are first optimized and coordinated at an aggregate level, and then further subdivided among individual components (i.e., flows) by solving multiple smaller subproblems, is known as *primal decomposition*. The coordination problem that optimizes aggregate resources is called the *master primal*.

1) Primal Decomposition: The LC first allocates a class-level bandwidth y_l^k for each class k and link l in the backbone. This decomposes *GLOBAL* into a total of K subproblems, as shown in Figure 4. Each of these subproblems is then

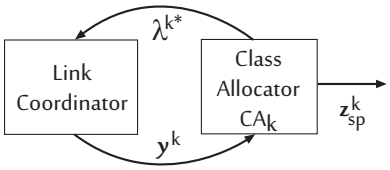

Fig. 5. Message passing in the 2-tier design.

independently solved by a CA to subdivide y_l^k among the flows of class k that use link l. We define the subproblem for class k as:

CLASS(k):

$$\begin{aligned} \text{maximize} \quad & \mathcal{U}^k = \sum_{s\in\mathcal{F}^k} w_s^k \left[a^k f^k\left(x_s^k\right) - b^k g^k\left(u_l^k\right)\right] \\ \text{subject to} \quad & \sum_{s\in\mathcal{F}^k}\sum_{p\in\mathcal{P}} A_{lp} R_{sp}^k \, z_{sp}^k \le y_l^k, \ \ \forall l \\ \text{variables} \quad & z_{sp}^k \ge 0, \ \ \forall s \in \mathcal{F}^k, p \end{aligned} \tag{5}$$

where the right hand side y_l^k of the constraint is a *fixed* quantity that plays the role of fixed link capacities, as in standard network utility maximization (NUM) problems.

These class-level subproblems are coordinated by solving a master primal in the LC. We define this master primal as:

MASTER-PRIMAL:

$$\begin{aligned} \text{maximize} \quad & \mathcal{U} = \sum_k \mathcal{U}^{k^*}\left(\mathbf{y}^k\right) \\ \text{subject to} \quad & \sum_{k\in\mathcal{K}} y_l^k \le c_l, \ \ \forall l \\ \text{variables} \quad & y_l^k \ge 0, \ \ \forall k, l \end{aligned} \tag{6}$$

where $\mathcal{U}^{k^*}\left(\mathbf{y}^k\right)$ is the optimal objective value of (5) for a fixed class-level bandwidth vector $\mathbf{y}^k = \{y_l^k, \forall l\}$.

2) Subgradient Updates: The master primal and the class-level subproblems are solved using a subgradient method in an iterative fashion until convergence. Suppose at iteration t of the master primal, the LC allocates a class-level bandwidth vector $\mathbf{y}^k(t)$ for class k. The CA then subdivides this bandwidth among the flows of class k by solving (5). Let the optimal Lagrange multipliers corresponding to the constraints in (5) be $\boldsymbol{\lambda}^{k^*}$. Clearly, this is a function of $\mathbf{y}^k(t)$. These Lagrange multipliers are sent by the CAs to the LC, which then updates the class-level bandwidths for the next iteration as:

$$\mathbf{y}^k(t+1) = \left[\mathbf{y}^k(t) + \beta(t)\cdot\boldsymbol{\lambda}^{k^*}\left(\mathbf{y}^k(t)\right)\right]_{\mathcal{X}}, \tag{7}$$

where $[\cdot]_{\mathcal{X}}$ denotes the projection onto the feasible set $\mathcal{X}$, and $\beta(t)$ is a small positive step size for iteration t. Figure 5 shows the message exchange for subgradient updates. We note that the above primal decomposition computes *optimal* sending rates for all the flows in the network across all classes and data centers.

B. Decomposing the 3-Tier Design

In our 3-tier design, the class-level bandwidths are first computed by the LC on tier-1, and then are subdivided by

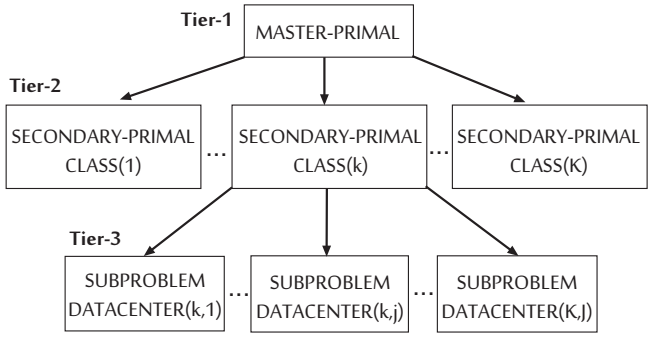

Fig. 6. A two-level primal decomposition of *GLOBAL* into a master primal on tier-1, K class-level secondary primals on tier-2, and KJ DC-level subproblems on tier-3.

the CAs on tier-2 across multiple data centers to compute DC-level bandwidths. These DC-level bandwidths are further subdivided by the DAs on tier-3 to compute flow-level path rates. Like in the 2-tier design, this rate allocation strategy also suggests a primal decomposition of *GLOBAL* into multiple smaller subproblems that can be coordinated by different management entities. However, now there are two levels of primal decomposition; one in which *GLOBAL* is decomposed into multiple subproblems at tier-2, and the other in which each tier-2 subproblem is further decomposed into multiple smaller subproblems at tier-3. We describe this two-level primal decomposition in the following.

1) A Two-Level Primal Decomposition: As before, the LC first allocates a class-level bandwidth y_l^k for each class k and link l in the backbone. This decomposes *GLOBAL* into K class-level subproblems that are coordinated by a master primal in the LC, as shown in Figure 6. Each class-level subproblem is solved by a CA. However, unlike in the 2-tier design where the CAs compute flow-level path rates, each CA now only allocates an aggregate DC-level bandwidth for each data center. Computing the flow-level path rates is delegated to the DAs, which solve the DC-level subproblems. Let $\mathcal{F}^{kj}$ denotes the set of flows of class k that originate from data center j. Also, let *DATACENTER*(k,j) denote the subproblem solved by the DA of class k in data center j, which we define as:

DATACENTER(k,j):

$$\begin{aligned} \text{maximize} \quad & \mathcal{U}^{kj} = \sum_{s\in\mathcal{F}^{kj}} w_s^k \left[a^k f^k\left(x_s^k\right) - b^k g^k\left(u_l^k\right)\right] \\ \text{subject to} \quad & \sum_{s\in\mathcal{F}^{kj}} \sum_{p\in\mathcal{P}} A_{lp} R_{sp}^k \, z_{sp}^k \le y_l^{kj}, \ \ \forall l \\ \text{variables} \quad & z_{sp}^k \ge 0, \ \ \forall s \in \mathcal{F}^{kj}, p \end{aligned} \tag{8}$$

where, as before, the right hand side y_l^{kj} of the constraint is a *fixed* quantity. We note that the structure of (8) is similar to that of (5), except for the superscript j which restricts considering only the flows that originate from data center j. These DC-level subproblems are coordinated by the CAs. In particular, CA_k coordinates all the DC-level problems that belong to class k by solving a *secondary primal*, which is defined as:

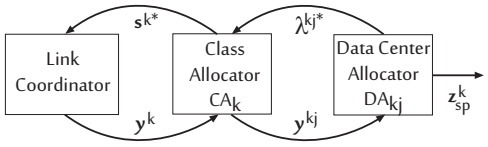

Fig. 7. Message passing in a 3-tier design.

SECONDARY-PRIMAL(k):

$$\begin{aligned} \text{maximize} \quad & \mathcal{U}^k = \sum_j \mathcal{U}^{kj^*}\left(\mathbf{y}^{kj}\right) \\ \text{subject to} \quad & \sum_{j\in\mathcal{J}} y_l^{kj} \le y_l^k, \ \ \forall l \\ \text{variables} \quad & y_l^{kj} \ge 0, \ \ \forall j, l \end{aligned} \tag{9}$$

where $\mathcal{U}^{kj^*}\left(\mathbf{y}^{kj}\right)$ is the optimal objective value of (8) for a fixed DC-level bandwidth vector $\mathbf{y}^{kj} = \{y_l^{kj}, \ \forall l\}$.

Finally, these secondary primals are coordinated by a *master primal*, defined as:

MASTER-PRIMAL:

$$\begin{aligned} \text{maximize} \quad & \mathcal{U} = \sum_k \mathcal{U}^{k^*}\left(\mathbf{y}^k\right) \\ \text{subject to} \quad & \sum_{k\in\mathcal{K}} y_l^k \le c_l, \ \ \forall l \\ \text{variables} \quad & y_l^k \ge 0, \end{aligned} \tag{10}$$

where $\mathcal{U}^{k^*}\left(\mathbf{y}^k\right)$ is the optimal objective value of (9) for a given class-level bandwidth $\mathbf{y}^k$.

2) Subgradient Updates: The primals and the subproblems are solved in an iterative fashion until convergence using a subgradient method as before. In particular, all the secondary primals need to converge first before the master primal goes on to the next iteration. Let t and t' denote the iteration indices of the master primal and secondary primals, respectively. Suppose at iteration t, the LC fixes a class-level aggregate bandwidth $\mathbf{y}^k(t)$. The CA then optimally divides this among the DAs of class k to solve (9). In particular, suppose at iteration t', the CA fixes a DC-level aggregate bandwidth $\mathbf{y}^{kj}(t')$. Then the DA uses this as the total budget to solve (8) and compute flow-level path rates.

Suppose the optimal Lagrange multipliers corresponding to the constraints in (8) is $\boldsymbol{\lambda}^{kj^*}$, which is a function of $\mathbf{y}^{kj}(t')$. These Lagrange multipliers are sent by the DAs to the CA, which then updates the DC-level bandwidth as:

$$\mathbf{y}^{kj}(t'+1) = \left[\mathbf{y}^{kj}(t') + \alpha(t') \cdot \mathbf{s}^k\left(\mathbf{y}^{kj}(t')\right)\right]_{\mathcal{X}}. \tag{11}$$

Here $\mathbf{s}^k(\cdot)$ is a subgradient corresponding to the constraints in (9), and is given by the sum of the optimal Lagrange multipliers of (8) as: $\mathbf{s}^k = \sum_{j\in\mathcal{J}} \boldsymbol{\lambda}^{kj^*}\left(\mathbf{y}^{kj}(t')\right)$. As before, $[\cdot]_{\mathcal{X}}$ denotes the projection onto the feasible set $\mathcal{X}$, and $\alpha(t')$ is a small positive step size for iteration t'.

Once all the secondary primals converge, the next iteration of the class-level bandwidth allocation takes place. Suppose, the optimal Lagrange multipliers once (9) converges be $\mathbf{s}^{k^*}$. These Lagrange multipliers are again a function of $\mathbf{y}^k(t)$, and

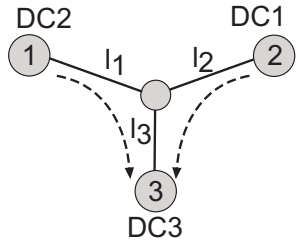

Fig. 8. A simple 3-node, 3-link topology.

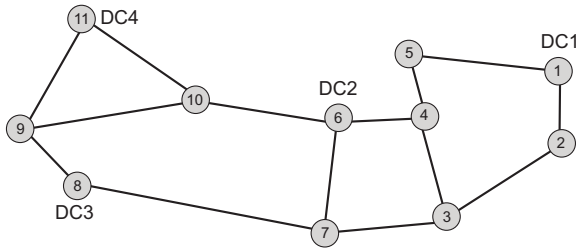

Fig. 9. Abilene backbone network with 4 data centers located at nodes 1, 6, 8, and 11.

are sent by the CAs to the LC. The LC then updates the class-level bandwidth as:

$$\mathbf{y}^k(t+1) = \left[\mathbf{y}^k(t) + \beta(t) \cdot \mathbf{s}\left(\mathbf{y}^k(t)\right)\right]_{\mathcal{Y}}. \qquad (12)$$

Here $\mathbf{s}(\cdot)$ is a global subgradient corresponding to the constraints in (10), and is given by the sum of the optimal Lagrange multipliers of (9) as: $\mathbf{s} = \sum_{k\in\mathcal{K}} \mathbf{s}^{k^*}\left(\mathbf{y}^k(t)\right)$. As before, $[\cdot]_{\mathcal{Y}}$ denotes the projection onto the feasible set $\mathcal{Y}$, and $\beta(t)$ is a small positive step size for iteration t. Figure 7 shows the message exchange for these subgradient updates. Like in the 2-tier design, here also the two-level primal decomposition computes *optimal* sending rates for all the flows in the network across all classes and data centers.

V. Performance Evaluation

In this section, we evaluate the performance of the two designs on two different topologies. We first compare their convergence behavior, and then the number of messages exchanged until convergence.

A. Experimental Setup

Backbone Topologies: We experiment with two network topologies: (1) a simple 3-node, 3-link topology, as shown in Figure 8, and (2) the Abilene backbone network [15], as shown in Figure 9. The simple topology allows us to reason about the system more easily, while the Abilene topology is more realistic and has more number of nodes and path diversity.

The simple topology has a data center located at each node. We assume that both data centers, DC1 and DC2, host two classes of traffic that are throughput-sensitive. We set the parameters for the two classes as: $a^1 = 2$, $b^1 = 0$ for class C1, and $a^2 = 10$, $b^2 = 0$ for class C2. We assume that the rate-dependent function is logarithmic, and the weights of all the flows in both classes are set to one. Each link has a capacity of 100 Mbps. Following (1), the utility functions are therefore given by: $U_s^k = a^k \log(\sum_{p\in\mathcal{P}} R_{sp}^k z_{sp}^k)$.

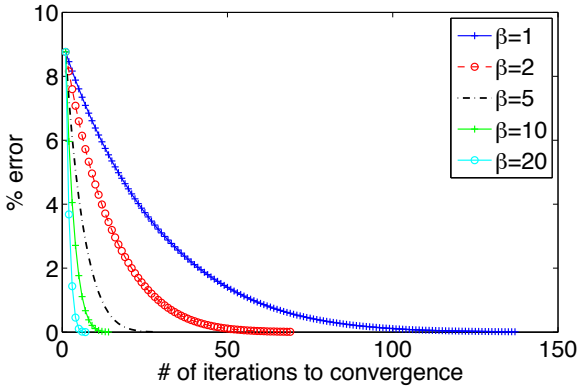

Fig. 10. Rate of convergence of the 2-tier design for different values of the class-level step size β.

The Abilene backbone has a link capacity of 1 Gbps in each direction, and the propagation delays are chosen so as to approximate realistic values between respective pairs of nodes. We suppose that there are four data centers located at nodes 1, 6, 8, and 11. Out of the many possible paths between the data centers, we choose the first three shortest paths between each pair, resulting in a total of 36 paths.

B. Convergence: Sensitivity to Step Sizes α and β

In our 2-tier design, the tunable step size β controls how the class-level bandwidths and the global utility react to changes in the subgradient $\boldsymbol{\lambda}^k$ (Equation (7)). Similarly, in our 3-tier design, the step sizes β and α, respectively, controls how the class-level and DC-level bandwidths react to changes in the subgradients $\boldsymbol{\lambda}^k$ and $\mathbf{s}^k$ (Equations (11) and (12)). In the following experiments, we compare the convergence behavior of the global utility in our two designs for constant values of these step sizes. We define convergence as being within 0.01% of the optimal. We call β the *class-level step size*, and α the *DC-level step size*. We first experiment with the simple topology.

Convergence Rate of the 2-Tier Design: In Figure 10, we plot the percentage error of the global utility for the 2-tier design for different values of the class-level step size β. We observe that the utility converges for all values of β between 1 and 20, and the rate of convergence increases rapidly for larger β. We ran the experiment for different values of β up to 50, and all of them converge very quickly (not all those graphs are shown here). For instance, starting at $\beta = 25$, the convergence is always within 3 to 5 iterations. For $\beta > 50$, however, the utility values oscillate and the system never converges; the reason being, as β gets larger, there is a tendency for the allocations to overshoot beyond the feasible region every single iteration. Thus, the convergence of the 2-tier design is more sensitive to values of β when they are small (i.e., $1 \leq \beta \leq 20$), but not much once β exceeds 25.

Convergence Rate of the 3-Tier Design: Figure 11 shows the percentage error of the global utility for the 3-tier design for different values of the class-level step size β, when the DC-level step size α is fixed at 2. We observe that the utility con-

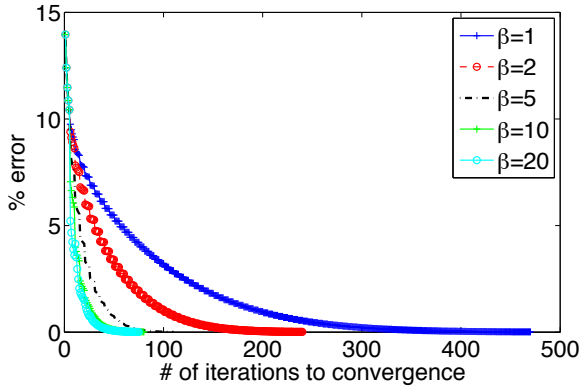

Fig. 11. The rate of convergence of the 3-tier design for different values of the class-level step size β, when the DC-level step size α is fixed at 2.

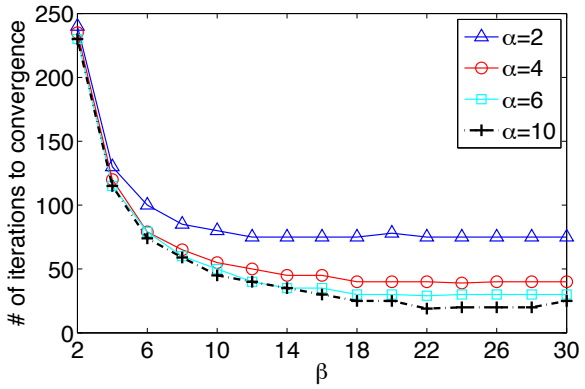

Fig. 12. Number of iterations to convergence in the 3-tier design for sweeping the class-level step size β for different values of the DC-level step size α.

verges for all values of β below 40, and the rate of convergence increases for higher β. The experiment is run with the number of iterations of the inner loop set to five for every iteration of the outer loop. We recall that the inner loop allocates DC-level bandwidths (controlled by α), while the outer loop allocates class-level bandwidths (controlled by β). Thus, for $\beta = 1$, which takes about 470 iterations to converge, the number of class-level allocations is $470/5 = 94$. Comparing Figure 10 and 11, we see that the rate of convergence for the 3-tier design is slower than the 2-tier design. For β above 40, however, the class-level bandwidths overshoot the feasible region in every iteration of the outer loop, and, as a result, the system never converges.

To get a better understanding of how sensitive the 3-design is in terms of its convergence behavior, we fix the DC-level step size α at different values and sweep the class-level step size β through all the values between 2 and 30 in steps of 2. Figure 12 shows the plots for four different values of α. We observe that for fixed α, the number of iterations to convergence initially decreases for increasing β, but gradually becomes almost constant for large β. For β more than 18, we see that all the four curves flatten out and the β values have almost no effect on the rate of convergence. This behavior is similar to the 2-tier design, where the convergence rate is very quick and remains almost unaffected for large values of β. We also see that the plots for large values of α (i.e., $\alpha = 4, 6, 10$) are very close to each other, indicating that the 3-tier design is

TABLE II
RATE OF CONVERGENCE OF THE TWO DESIGNS FOR DIFFERENT VALUES OF THE CLASS-LEVEL STEP SIZE β. ALSO SHOWN ARE THE "GOOD" VALUES OF THE DC-LEVEL STEP SIZE α FOR THE 3-TIER DESIGN.

Class-level step size β	**2-tier design**	**3-tier design**
small $\beta = 1, 2$	slow	very slow, all α
medium $\beta = 5, 10$	moderate	slow, all α
large $\beta = 20, 30$	fast	moderate, all α
very large $30 < \beta < 40$	fast	moderate, $\alpha \leq 16$
extremely large $40 \geq \beta < 50$	fast	does not converge
$\beta \geq 50$	does not converge	does not converge

less sensitive to large α. Furthermore, we find that for $\alpha > 16$ and $\beta > 30$, the system oscillates and never converges. Thus, the "good" values of α and β for which the 3-tier design is stable are when $\alpha \leq 16$ and $\beta \leq 30$.

In summary, Table II lists the different values of α and β, and the convergence behavior of the two designs. In practice, one should choose those values of α and β for which the system converges quickly. We have not experimented with dynamic traffic demands in this work; however, for the private OSP backbone networks under consideration, the demand variability can be controlled to some extent because the traffic sources are owned by the OSP, and the step sizes can be chosen accordingly. With respect to fault tolerance (e.g., link failures), our proposed designs can exploit multi-path routing to split traffic over the "good" paths, and also re-run the optimization on the modified topology after failure to compute new optimal sending rates [16].

C. Message-Passing Overhead

In both our designs, the management entities exchange messages with each other in order to allocate class-level and/or DC-level bandwidths in an iterative fashion. In this section, we compute and compare the number of messages exchanged in the two designs.

In the analysis below, $\mathcal{F}^{kj}$ is the set of flows of class k that are hosted from data center j, and the routing matrix R^k_{sp} encodes the paths used by each flow s. Also, as stated in Table I, K is number of traffic classes and L is the number of links in the backbone. We also assume that the number of class-level allocations required for the 2-tier design to converge is N, and that required for the 3-tier design to converge is N'. For the 3-tier design, we further assume that M iterations of DC-level allocations are needed for each iteration of the class-level allocation.

Message-Passing in the 2-Tier Design: In our 2-tier design, the LC first allocates the class-level bandwidths, y^k_ls, to each CA in the beginning of each iteration. The total number of variables passed by the LC to all the CAs is therefore KL. Each CA then computes the path rates for the individual flows of its own class and sends the rates to the respective sources. Since the flows of each class can originate from any of the data centers, each CA potentially communicates with all the data centers. Thus, the number of path rate variables sent by the CA of class k is $\sum_{j\in\mathcal{J}} \sum_{s\in\mathcal{F}^{kj}} \sum_p R^k_{sp}$. Finally, after computing

the flow-level path rates, each CA passes the optimal Lagrange multipliers, $\boldsymbol{\lambda}^{k^*}$, to the LC at the end of each iteration, thus, sending a total of KL variables. Summing up, the number of variables exchanged until convergence is

$$\#\text{ of messages} = N\left[2KL + \sum_{k\in\mathcal{K}}\sum_{j\in\mathcal{J}}\sum_{s\in\mathcal{F}^{kj}}\sum_{p} R_{sp}^{k}\right]. \quad (13)$$

Message-Passing in the 3-Tier Design: In our 3-tier design, the LC first allocates the class-level bandwidths, y_l^ks, to each CA in the beginning of each iteration of the outer loop, thus, sending a total of KL variables. Each CA then subdivides the y_l^ks into DC-level bandwidths, y_l^{kj}s, for each data center j, and sends these to the DAs. The number of variables passed by the CAs to all the DAs in each iteration of the inner loop is therefore JKL. Each DA then computes the path rates for the individual flows and sends the rates to the respective sources. However, since the DAs are located within each data center, these messages are sent locally and are not counted in our analysis. Once the flow-level path rates are computed, each DA sends the optimal Lagrange multipliers, $\boldsymbol{\lambda}^{kj^*}$, to the CA of its class. This incurs passing a total of JKL variables for each inner loop iteration. Finally, each CA sends the optimal Lagrange multipliers, $\mathbf{s}^{k^*}$, to the LC at the end of each iteration of the outer loop. This requires passing a total of KL variables from all the CAs. Summing up, the number of variables exchanged until convergence is

$$\#\text{ of messages } = N'[2KL + 2JKLM]. \quad (14)$$

Comparison of Message-Passing Overhead: From (13), we see that the number of messages exchanged in the 2-tier design depends on the number of the flows, $\mathcal{F}^{kj}$, of each class k hosted from each data center j. However, the 2-tier design has only a single loop which iteratively allocates class-level bandwidths until convergence. On the other hand, from (14) we observe that the number of messages exchanged in the 3-tier design does not depend on the number of flows; however, the design has two loops which iteratively allocate both class-level and DC-level bandwidths.

In Figure 13, we compare the message-passing overhead in the two designs as a function of the number of flows in each class hosted between every pair of data centers in the Abilene topology. We choose step sizes for the two designs that converge within 100 iterations. We observe that, as expected, the number of messages exchanged in the 3-tier design remains constant with the number of flows. In the 2-tier design, however, though the number of messages exchanged is fewer for small number of flows, it increases once the number of flows per class hosted from each data center exceeds slightly more than 110. This amounts to a total of $110 \times 2 \times 4 = 880$ flows. Thus, in practice, if the number of flows is small, one would prefer to use the 2-tier design; otherwise, the 3-tier design is preferred.

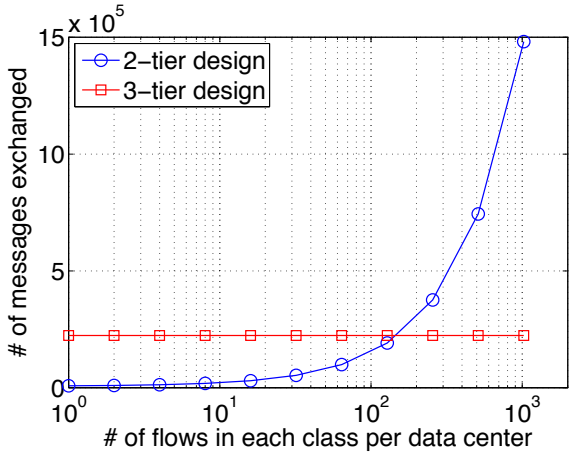

Fig. 13. Message-passing overhead of the two designs as a function of the number of flows in each class hosted from each data center in the Abilene topology.

VI. Related Work

Traffic management using optimization theory [17], [18] has been extensively studied in literature [13], [19]–[21]. Using multiple decompositions, a distributed protocol called TRUMP is proposed in [20], which is adaptive, robust, and flexible. In [21], a distributed architecture called DaVinci is presented which can run customized protocols on multiple virtual networks sharing the same physical topology to support applications with diverse performance objectives. Using adaptive network virtualization, it is shown that DaVinci maximizes the aggregate utility of all virtual networks. In [13], a distributed multi-path protocol is proposed for delay-sensitive, inelastic traffic using decomposition techniques.

Despite using decomposition theory, the protocols proposed in [13], [20], [21] are fully-distributed, which rely on periodic link feedbacks from the routers to adjust the sending rates. In an ISP network, this is perhaps more desirable because the traffic sources are typically owned by end users, whereas the operators control the routing paths and tune the link weights. In comparison, the focus of this work is on wide-area OSP networks [22], where the sources as well as the backbone are owned by the same OSP. To this end, our proposed designs are semi-centralized, which use a small number of management entities to compute optimal flow-level path rates across multiple classes and data centers. This makes our protocols better in terms of scalability, message-passing overhead, and convergence behavior. Moreover, while TRUMP does not consider multiple classes of traffic and different applications within each class, DaVinci does not consider scalability, or applications within each virtual network having different weights.

There have been several recent studies on traffic modeling and characterization within a single data center [23]–[25]. However, little is known about traffic management in OSP networks [9]. The study in [9] uses Yahoo! datasets to analyze both inter-data center and client traffic, revealing a hierarchical deployment of data centers at different geographical locations. In [24], the authors capture both macroscopic and microscopic traffic characteristics in data center networks from large-scale data sets. In [23], an empirical study of end-to-end traffic patterns is carried out across multiple data centers to examine

temporal and spatial variations in link loads and losses. The work also provides a framework to derive parameters that define sending behavior of the traffic sources. Finally, Google's globally-deployed wide area network (WAN) called B4 [26] uses Software Defined Networking (SDN) principles and OpenFlow [27] to simultaneously support standard routing protocols and centralized TE. Leveraging the common ownership (by Google) of all the applications, servers, and data center networks all the way up to the edge of the backbone, B4 can (1) adjudicate among competing demands under resource constraints, (2) use multi-path forwarding/tunneling to leverage available network capacity according to application priority, and (3) dynamically reallocate bandwidth in the face of link/switch failures or shifting application demands.

VII. Conclusion

In this work, we examined two alternative traffic management designs for OSP networks. Our designs are both scalable and semi-centralized, but differ in their degree of distributedness. They have a tiered structure and use a small number of management entities to optimize sending rates at the flow-level across multiple traffic classes. Using primal decomposition, we showed that both the designs are provably optimal in maximizing the aggregate utility of all flows. Simulations on a realistic backbone topologies revealed quicker convergence rate of the 2-tier protocol as compared to the 3-tier one, but with increased message-passing. Our future research includes experimenting with real data sets.

Acknowledgment

Our work was supported in part by NSF grant 1162112 and a gift from Google. The authors would like to thank Professor Mung Chiang for his insightful comments, and Neelanjana Gautam for proofreading the manuscript that helped us improve the quality of this work.

References

[1] Z. Zhang, M. Zhang, A. Greenberg, Y. C. Hu, R. Mahajan, and B. Christian, "Optimizing Cost and Performance in Online Service Provider Networks," in *USENIX NSDI*, San Jose, CA, April 2010, pp. 33–47.

[2] "Google Data Centers," http://www.google.com/about/datacenters/.

[3] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "BCube: A High Performance, Server-Centric Network Architecture for Modular Data Centers," in *ACM SIGCOMM*, Barcelona, August 2009, pp. 63–74.

[4] "Google Throws Open Doors to Its Top-Secret Data Center," http://www.wired.com/wiredenterprise/2012/10/ff-inside-google-data-center/.

[5] "How Dirty is Your Data?" http://www.greenpeace.org/international/en/publications/reports/How-dirty-is-your-data/.

[6] A. Mahimkar, A. Chiu, R. Doverspike, M. D. Feuer, P. Magill, E. Mavrogiorgis, J. Pastor, S. L. Woodward, and J. Yates, "Bandwidth on Demand for Inter-Data Center Communication," in *ACM HotNets*, Cambridge, MA, November 2011, pp. 24:1–24:6.

[7] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The Cost of a Cloud: Research Problems in Data Center Networks," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 1, pp. 68–73, December 2008.

[8] "Inside Microsoft's Internet Infrastructure & Its Plans For The Future," http://gigaom.com/2008/06/30/microsofts-internet-infrastructure-its-big-plans/.

[9] Y. Chen, S. Jain, V. K. Adhikary, Z.-L. Zhang, and K. Xu, "A First Look at Inter-Data Center Traffic Characteristics via Yahoo! Datasets," in *IEEE INFOCOM*, Shanghai, April 2011, pp. 1620–1628.

[10] Google, "Inter-Datacenter WAN with Centralized TE using SDN and OpenFlow," Open Networking Summit 2012, https://www.opennetworking.org/images/stories/downloads/misc/googlesdn.pdf.

[11] F. P. Kelly, A. Maulloo, and D. Tan, "Rate Control for Communication Networks: Shadow Prices, Proportional Fairness and Stability," *J. Operational Research Society*, vol. 49, pp. 237–252, March 1998.

[12] F. P. Kelly, "Fairness and Stability of End-to-End Congestion Control," *European Journal of Control*, pp. 159–176, 2003.

[13] U. Javed, M. Suchara, J. He, and J. Rexford, "Multipath Protocol for Delay-sensitive Traffic," in *COMSNETS*, Bangalore, January 2009, pp. 438–445.

[14] S. Kandula, D. Katabi, S. Sinha, and A. Berger, "Dynamic Load Balancing Without Packet Reordering," *ACM SIGCOMM Computer Communication Review*, vol. 37, pp. 51–62, April 2007.

[15] "Abilene Backbone," http://abilene.internet2.edu.

[16] M. Suchara, D. Xu, R. Doverspike, D. Johnson, and J. Rexford, "Network Architecture for Joint Failure Recovery and Traffic Engineering," in *ACM SIGMETRICS*, San Jose, CA, June 2011, pp. 97–108.

[17] D. Palomar and M. Chiang, "Alternative Distribution Algorithms for Network Utility Maximization: Framework and Applications," *IEEE/ACM Trans. Autom. Control*, vol. 52, no. 12, pp. 2254–2269, December 2007.

[18] M. Chiang, S. H. Low, R. Calderbank, and J. C. Doyle, "Layering as Optimization Decomposition: A Mathematical Theory of Network Architectures," *Proc. IEEE*, vol. 95, no. 1, pp. 255–312, January 2007.

[19] S. Floyd and V. Jacobson, "Link-Sharing and Resource Management Models for Packet Networks," *IEEE/ACM Trans. Netw.*, vol. 3, no. 4, pp. 365–386, August 1995.

[20] J. He, M. Suchara, M. Bresler, J. Rexford, and M. Chiang, "Rethinking Internet Traffic Management: From Multiple Decompositions to a Practical Protocol," in *ACM CoNEXT*, New York, December 2007, pp. 1–12.

[21] J. He, R. Zhang-Shen, Y. Li, C.-Y. Lee, J. Rexford, and M. Chiang, "DaVinci: Dynamically Adaptive Virtual Networks for a Customized Internet," in *ACM CoNEXT*, Madrid, December 2008, pp. 15:1–15:12.

[22] J. W. Jiang, "Wide-Area Traffic Management for Cloud Services," *Ph.D. Thesis, Princeton University*, 2012.

[23] T. Benson, A. Anand, A. Akella, and M. Zhang, "Understanding Data Center Traffic Characteristics," *ACM SIGCOMM Computer Communication Review*, vol. 40, pp. 92–99, January 2010.

[24] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The Nature of Data Center Traffic: Measurements & Analysis," in *ACM IMC*, Chicago, IL, November 2009, pp. 202–208.

[25] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "VL2: A Scalable and Flexible Data Center Network," in *ACM SIGCOMM*, Barcelona, August 2009, pp. 51–62.

[26] 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 *ACM SIGCOMM*, Hong Kong, August 2013.

[27] "OpenFlow Specification," http://www.openflow.org/wp/documents/.

Amitabha Ghosh received his Ph.D. in Electrical Engineering from the University of Southern California in 2010, and was a post-doc at Princeton University during 2010-2012. His research interests lie in the design and analysis of algorithms and protocols for sensor networks, cellular networks, Wi-Fi, satellite communications, cognitive radios, and data center networks. Between 2000 and 2004, he worked for Alcatel-Lucent and Honeywell Labs in Bangalore on GSM/GPRS and ad hoc networks. Prior to that, he received his B.S. in Physics from Indian Institute of Technology, Kharagpur, in 1996, and his M.E. in Computer Science and Engineering from Indian Institute of Science, Bangalore, in 2000. He is currently a Research & Development Scientist at UtopiaCompression based in Los Angeles, where he works on developing wireless networking and communication technologies for tactical networks.

Sangtae Ha is the CTO and VP Engineering at DataMi. He received his Ph.D. in Computer Science from North Carolina State University in 2009. During his Ph.D. study, he co-authored CUBIC, a new mechanism for TCP congestion control, and was the primary engineer pushing it into Linux 2.6.18. CUBIC has been the default TCP congestion control algorithm in Linux since 2006 and is currently used by more than 40% of Internet servers around the world as well as several hundred million Android/Linux devices. After completing his Ph.D., he co-founded the Princeton EDGE Lab (http://scenic.princeton.edu) as its first Associate Director in 2009 and led its research team as an Associate Research Scholar at Princeton University from 2010 to 2013. In particular, he led the DataMi (http://www.datami.com) project team, which develops Smart Data Pricing (SDP) solutions to help Internet Service Providers (ISPs) meet the increasingly untenable demand for mobile data. He is a co-founder of DataMi Inc. and a technical consultant to a few startups. He is an IEEE Senior Member and was a general co-chair of the Smart Data Pricing workshop in INFOCOM 2013. In 2014, he will join the University of Colorado Boulder as an Assistant Professor.

Edward Crabbe is a network architect at Google, where his responsibilities range from overall network design including backbone switching and routing architectures to novel protocol design and specification. He has been working on large networks in one form or another since the late nineties, and has held senior positions at a number of tier one backbones and several of today's top enterprises and cloud hosting providers.

Jennifer Rexford is the Gordon Y.S. Wu Professor of Engineering in the Computer Science department at Princeton University. While working at AT&T Research from 1996 to 2005, she designed network-management techniques that are in daily use in AT&Ts backbone network. Jennifer was chair of ACM SIGCOMM from 2003 to 2007, and currently co-chairs the advisory council of NSF's CISE directorate. She is an ACM Fellow and received ACM's Grace Murray Hopper Award for outstanding young computer professional of the year for 2004.