

Online Allocation of Communication and Computation Resources for Real-time Multimedia Services

Jun Liao, Philip A. Chou, Chun Yuan, Yusuo Hu, and Wenwu Zhu

Microsoft Research Technical Report MSR-TR-2012-6

¹Abstract—We observe that, in a network, the location of the node on which a service is computed is inextricably linked to the locations of the paths through which the service communicates. Hence service location can have a profound effect on quality of service (QoS), especially for communication-centric applications such as real-time multimedia. In this paper, we propose an online algorithm that uses pricing to consider server load, route congestion, and propagation delay jointly when locating servers and routes for real-time multimedia services in a network with fixed computing and communication capacities. The algorithm is online in the sense that it is able to sequentially allocate resources for services with long and unknown duration as demands arrive, without benefit of looking ahead to later demands. By formulating the problem as one of lowest cost subgraph packing, we prove that our algorithm is nevertheless C -competitive with the optimal algorithm that looks ahead, meaning that our performance is within a constant factor C of optimal, as measured by the total number of service demands satisfied, or total user utility. Using mixing services as an example, we show through experimental results that our algorithm can adapt to cross traffic and automatically route around congestion and failure of nodes and edges, can reduce latency by 40% or more, and can pack 20% more sessions, compared to conventional approaches.

Keywords: *resource allocation; multimedia services; mixing; congestion pricing; subgraph packing; online algorithm; primal-dual algorithm; approximation algorithm*

I. INTRODUCTION

As both mobile and fixed users increasingly turn to the Internet for their real-time multimedia communication needs, real-time multimedia service providers are beginning to operate their infrastructures on a global scale. The large scale is driving both the opportunity and the need for real-time multimedia service providers to allocate their communication and computing resources to provide higher quality of service (QoS) to end users at lower cost.

Jun Liao, Wenwu Zhu, and Chun Yuan are with the Department of Computer Science and Technology, Tsinghua University, Beijing, China (email: {liaoj09, wwwwzhu, yuanc}@tsinghua.edu.cn). Philip A. Chou is with Microsoft Research, Redmond, USA (email: pachou@microsoft.com). Yusuo Hu is with Google, USA (email: yusuo.hu@gmail.com). This work was performed while Jun Liao, Philip A. Chou, Wenwu Zhu and Yusuo Hu were at Microsoft Research Asia, Beijing, China.

To illustrate, consider that a provider of audio and video telephony and conferencing for both mobile and fixed devices over the Internet must provide services such as proxies for mobile devices, gateways to external telephone and conferencing systems, multicast, and mixing services. Clients must connect to these services through the network either directly or indirectly through other services. In any given conferencing session for a set of clients, a large scale service provider may have a wide choice of where to locate its various services: on any of the servers in any of the provider's data centers, on any of the servers in any of the enterprise data centers belonging to any of the clients' corporate subscribers, on a server in any of the homes or conference rooms where any of the clients is located, or on any of the clients' terminal devices. These server and client nodes and the network connections between them form an overlay network. The provider's choice of where to run the services within the overlay and where to route the data between the services and the clients are inextricably linked, and can profoundly affect the quality of service to the clients' session. For satisfactory QoS, it is imperative that overloaded servers, congested routes, and routes with long propagation delay are avoided. Furthermore, the choice of where to place the services and where to route the data implies an allocation of resources: computing resources on the designated servers, and communication resources on the designated routes. Since resources are limited, and sessions can be lengthy, the choice of location also impacts the QoS of other sessions going on at the same time, and sessions that will begin in the future, which must compete for those resources. This makes optimization difficult, since the future is unknown and resources must be allocated *online* in the sense that requests for service arrive sequentially, and at the time of each request, resources must either be denied or allocated and not retracted until that session terminates. This is in addition to the usual problem of maximizing the number of sessions served within the given infrastructure while minimizing cost. Such non-trivial resource allocation problems are now being faced as global scale real-time multimedia applications grow beyond pure peer-to-peer and single service location solutions.

Traditional approaches to resource allocation for real-time multimedia services treat resource allocation for either communication or computation but not both. For example, addressing the communication aspect, Andersen et al. built Resilient Overlay Networks (RONs) over heterogeneous networks to perform routing, detect path outages, and perform path selection recovery for applications [1]. Subramanian et al. proposed OverQoS to enhance end-to-end QoS over a given fixed overlay path by conducting loss control and bandwidth allocation for each flow/application [2]. Addressing the computation aspect, Szymaniak et al. proposed a traditional round-robin algorithm to achieve load balancing, which is very efficient at a small scale but is not scalable [3]. Karger et al. used a consistent hashing algorithm to distribute load [4]. In a recent exception, Chowdhury et al. studied resource allocation for both communication and computation in a virtual network, allocating bandwidth and CPU resources to generic service requests [5]. However, theirs is an offline (i.e., batch) solution and does not consider delay; thus it is not suitable for online real-time applications.

In this paper we develop an approach to online allocation of communication and computation resources for real-time multimedia services. In our approach, we attribute prices to resources based on their level of congestion or load, and we also attribute a price to propagation delay. As each service request arrives in an online fashion, we map the request onto the overlay network (i.e., we assign the service to a server and we assign routes between the service and clients) to minimize the cost of the service according to the prices. Then we update the prices according to the new levels of congestion and load. Between service requests, we adapt the bit rates of each ongoing service and continue to refresh the prices as cross traffic and other loads change. Thus as links or servers become congested or fail we are able to route future services around the failures. Our approach is based on a formalism in which the prices arise as the multipliers in the dual of a primal linear program, which is a relaxation of an integer program whose objective is to maximize the total user utility or the total number of services in the system. We prove that our online algorithm is C -competitive; that is, our online algorithm performs within a factor C of the optimal (offline) solution to the integer program. We demonstrate the value of the approach in comparison to conventional approaches through numerous experiments. Using latency data from PlanetLab to construct a hypothetical overlay network spanning 12 branch offices across two continents, we show that we are able to reduce the average latency of audio mixing by 40% or more, compared to the conventional approach of mixing at a multipoint conferencing unit (MCU) in a centralized location. Moreover, we show that we can pack up to 20% more calls into the existing network than conventional approaches.

Our approach can be viewed as a generalization of optimal routing to computation in addition to communication aspects. In optimal routing, route requests are mapped onto the underlying network using a shortest path algorithm [6]. The maximum number of routes that can packed onto a network is a classic problem, for which online variations and approximation algorithms exist [7]. Once routes are packed, traffic can be modulated using congestion control algorithms based on congestion pricing [8]. Our approach to services, which involve both communication and computation aspects, parallels these three classic problems.

The rest of the paper is organized as follows. Throughout the paper, we use audio mixing as a canonical example of a real-time multimedia service request. In Section II, we formalize the problem of mapping a single service request onto the underlying network as a problem of finding a lowest cost subgraph, and we introduce an algorithm for finding near-optimal subgraphs for mixing services. In Section III, we add capacities to the nodes and links in the network, and formalize the problem of maximizing the total number of service requests (or maximizing the total user utility) that can be satisfied by the network subject to its resource constraints. We provide an online algorithm to solve the problem and prove that it is C -competitive with the optimal solution. In Section IV, we show how the system can adapt as cross traffic and other sessions come and go. Section V provides experimental results and Section VI concludes the paper.

II. RESOURCE ALLOCATION FOR A SINGLE SERVICE REQUEST

A. Network, Prices, Delays, and Capacities

We let a directed graph $G = \langle V, E \rangle$ model our overlay network of servers and clients. Each node $v \in V$ represents either a server $s \in S$ or a client $c \in C$, and each edge $(v_1, v_2) \in E$ represents a path through the underlay network from v_1 to v_2 . We attribute a price y_e to each edge $e \in E$ and a price y_v to each node $v \in V$. These are the prices to service a unit of load, where load on an edge is measured in units of bandwidth (such as megabits per second, or Mbps) and load on a node is measured in units of computing power (such as millions of instructions per second, or Mips). We also attribute a propagation delay d_e to each edge $e \in E$. For use in Section III, we also attribute a maximum bit rate, or communication capacity c_e , for each edge $e \in E$ and a maximum CPU load, or computation capacity c_v , for each node $v \in V$. Like load, these capacities will be measured in units of bandwidth and computing power, respectively. Examples are shown in Figure 1. For notational convenience, in this paper, edges that are shown without direction represent a pair of edges in opposite directions, with the same labels (delays, prices, and capacities) in each direction.

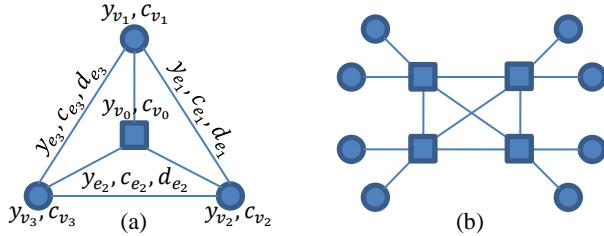


Figure 1. Networks. (a) Central server model. (b) Branch office model. Circles represent clients; squares represent servers. Nodes (clients and servers) as well as edges are labeled by prices \mathbf{y} and capacities \mathbf{c} . Edges are also labeled by propagation delays \mathbf{d} .

B. Service Requests

Suppose a set of clients $C \subseteq V$ wants to set up a communication session among themselves, such as a conference call. To do so, they make *requests* (or *demands*) for service such as transcoding, multicasting, mixing, and routing. We can represent different types of service requests using directed graphs $G' = \langle V', E' \rangle$, as shown in Figure 2. In the directed graph G' that represents a request, nodes in V' are either client nodes $c' \in C'$ or server nodes $s' \in S'$. The client nodes $c' \in C'$ are identified with the clients $c \in C$. The server nodes $s' \in S'$ are *placeholders* for the network nodes $v \in V$ (either servers or clients) where the service will be computed, the locations of which are yet to be determined. Edges $e' \in E'$ are *placeholders* for the network edges $e \in E$ that will connect the clients to those locations. The service request typically includes a communication requirement $r_{e'}$ (in Mbps, say) for each edge $e' \in E'$ and a computation requirement $r_{v'}$ (in Mips, say) for each server node $v' \in V'$.

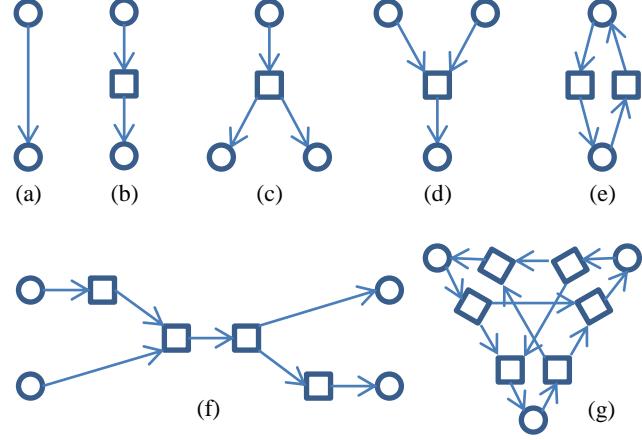


Figure 2. Service requests. (a) Routing. (b) Transcoding. (c) Multicasting. (d) Mixing. (e-g) Compositions of elementary service requests. Circles are clients; squares are placeholders for computing nodes. Resource requirements are not shown.

C. Mapping

To service a request G' , resources in the network G must be assigned to the placeholders. This is done by mapping G' onto G . In this context, a *mapping* is an isomorphism $\psi: G' \rightarrow G$ that assigns to G' a subgraph of G such that 1) ψ maps each client $c' \in C'$ to its twin $c \in C$, and 2) ψ maps each server node $s' \in S'$ to a node $v \in V$ (either a client or a server) in such a way that it respects the connections in G' and G . To be specific, if v'_1 is connected to v'_2 in G' , then $\psi(v'_1)$ must be connected to $\psi(v'_2)$ in G , or in other words, if $(v'_1, v'_2) \in E'$, then either $(\psi(v'_1), \psi(v'_2)) \in V$ or $\psi(v'_1) = \psi(v'_2)$. Various mappings of a request onto a network are shown in Figure 3.

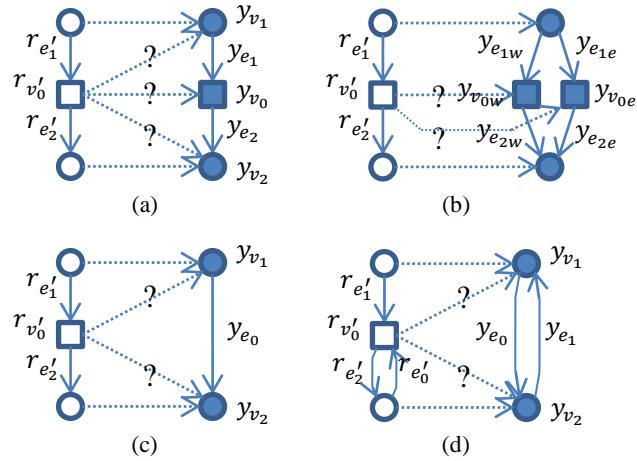


Figure 3. Mappings. (a) Mobile proxy model. (b) Data center model. (c) Computing partition model. (d) Thin vs. thick client model. Possible mappings are shown as dotted arrows.

D. Cost of a Mapping

The cost of a mapping $\psi: G' \rightarrow G$ is the sum of the costs of the resources used. That is,

$$Cost(\psi) = \sum_{v' \in V'} y_{\psi(v')} r_{v'} + \sum_{e' \in E'} y_{\psi(e')} r_{e'} + yd(\psi(G')).$$

The first term is the sum, over all computing resources, of the product of the unit price of the resource and the amount of the resource required. The second term is the same for communication resources. The last term is the product of the unit cost of delay y and the amount of delay $d(\psi(G'))$ in the subgraph $\psi(G')$. The delay $d(\psi(G'))$ is measured in units appropriate to the user experience, computed from the delays d_e on each edge $e \in \psi(G')$. We will discuss various measures of delay in Subsection G.

E. Minimum Cost Mapping / Subgraph

Minimizing the cost of a mapping means allocating the least expensive resources to satisfy a service request. Hence finding the minimum cost mapping is a key procedure in optimal resource allocation. We call the subgraph $\psi(G') \subseteq G$ a *minimum cost subgraph* of G .

F. Examples

The minimum cost subgraph model can help make decisions in various distributed computation scenarios.

1) Mobile Proxy Model

Figure 3a illustrates a scenario where a source client v_1 wants to send video encoded at bit rate $r_{e'_1}$ via a proxy server v_0 (say, located at a base station) to a mobile destination client v_2 . The video needs to be transcoded for the mobile client by a transcoding service v'_0 . Transcoding will require $r_{v'_0}$ Mips and will result in bit rate $r_{e'_2}$. If the transcoder is hosted at the source v_1 , the cost will be $y_{v_1} r_{v'_0} + (y_{e_1} + y_{e_2}) r_{e'_2} + c$, where $c = y(d_{e_1} + d_{e_2})$ is the cost of delay. If the transcoder is hosted at the proxy v_0 , the cost will be $y_{e_1} r_{e'_1} + y_{v_0} r_{v'_0} + y_{e_2} r_{e'_2} + c$. If the transcoder is hosted at the mobile device v_2 , the cost will be $(y_{e_1} + y_{e_2}) r_{e'_1} + y_{v_2} r_{v'_0} + c$. The best choice depends on the relative numbers. Whether the mobile client is a phone or a laptop, whether the wireless link is 3G or WiFi, and whether the transcoder changes the data rate from high to low (further compression), from low to high (rendering), or is neutral (format conversion) may affect the decision. Choosing the location based on minimum cost is a principled way to partition the eight-dimensional space $(r_{e'_1}, r_{v'_0}, r_{e'_2}, y_{v_0}, y_{e_1}, y_{v_0}, y_{e_2}, y_{v_2})$ into the three decision regions.

2) Data Center Model

Figure 3b illustrates a scenario where the service provider needs to decide whether to host a transcoder in its data center v_{1w} on the west coast or in its data center v_{1e} on the east coast. The west coast will be chosen if

$$y_{e_{1w}}r_{e'_1} + y_{v_{0w}}r_{v'_0} + y_{e_{2w}}r_{e'_2} + y(d_{e_{1w}} + d_{e_{2w}}) < y_{e_{1e}}r_{e'_1} + y_{v_{0e}}r_{v'_0} + y_{e_{2e}}r_{e'_2} + y(d_{e_{1e}} + d_{e_{2e}}).$$

Thus a data center will be shunned if it is overloaded (since its price will be high), if either of its routes are congested (since their prices will be high), or if it is far from the clients (since its routes will have high propagation delay). But the relative weights of these factors depends on the demands $r_{e'_1}, r_{v'_0}, r_{e'_2}$.

3) Computing Partition Model

Figure 3c illustrates the classic scenario of partitioning a computation pipeline across a communication link. The choice is to place a computation v'_1 near the input data v_1 or near the output data v_2 . The former should be chosen if $y_{v_1}r_{v'_0} + y_{e_0}r_{e'_1} < y_{v_2}r_{v'_0} + y_{e_0}r_{e'_1}$, i.e., if

$$y_{e_0}(r_{e'_1} - r_{e'_2}) > (y_{v_1} - y_{v_2})r_{v'_0},$$

that is, if the decrease in communication cost is more than the increase in computation cost. If the computation costs y_{v_1} and y_{v_2} are equal, computation should be put where the lesser amount of data is transmitted. If the data amounts $r_{e'_1}$ and $r_{e'_2}$ are equal, computation should be put where computation is less expensive. Delay is the same in either case.

4) Thin vs. Thick Client Model

Figure 3d illustrates another classic scenario: the choice of thin vs. thick client in a client-server (or client+cloud) scenario. The bottom node v_2 is the location of the client and the top node v_1 is the location of the server (in the cloud), which is connected to a backend database. The choice is whether to have a thin client (i.e., placing most computation at the server) or a thick client (i.e., placing most computation at the client). The former should be chosen if $y_{v_1}r_{v'_0} + y_{e_0}r_{e'_2} + y_{e_1}r_{e'_0} + y(d_{e_1} + d_{e_2}) < y_{v_2}r_{v'_0} + y_{e_0}r_{e'_1} + yd_{e_2}$, i.e., if

$$y_{e_0}(r_{e'_1} - r_{e'_2}) > (y_{v_1} - y_{v_2})r_{v'_0} + yd_{e_1} + y_{e_1}r_{e'_0},$$

that is, if the decrease in downlink communication cost is more than the increase in computation, delay, and uplink communication costs. The difference between this and the previous example is that the client must issue a request to get a response, possibly leading to additional delay and uplink communication costs, which must be taken into account.

5) Routing

Though not illustrated, routing is yet another classic scenario. Suppose there are multiple network paths $\{e_j\}$, all originating at node v_1 and terminating at node v_2 . For path e , let y_e denote the cost per unit bandwidth and let d_e denote the propagation delay. A request to route from client $v'_1 = v_1$ to client $v'_2 = v_2$ at bit rate $r_{e'}$ should map e' to e^* if

$$e^* = \arg \min_e (y_e r_{e'} + y d_e).$$

Thus finding the optimal route is a simple shortest path problem with respect to these metrics.

6) Mixing

Mixing is the canonical service we consider in this paper. The elementary mixing service shown in Figure 2c creates a mixture for a single client. A generalization is a multi-way mixing service, which is a mixing service that creates mixtures for all of its clients. A multi-way mixing service could be implemented as the composition of elementary mixing services and multicast services as shown in Figure 2g. However it is usually more convenient to package a multi-way mixing service into a single component as shown in Figure 4a.

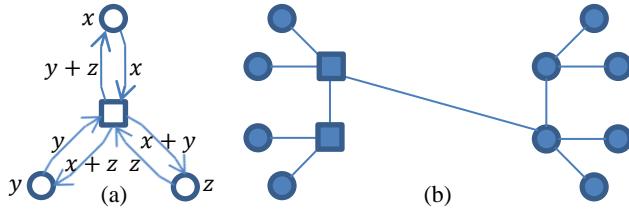


Figure 4. Multiway mixing. (a) Centralized three-way mixing request. (b) Distributed ten-way mixing service performed for a set of ten clients (circles) by multi-way mixers at interior nodes of a Steiner tree, some located at clients. The long middle edge exchanges sub-mixtures between left and right. Vertical edges also exchange mixtures.

Commonly used in audio conferencing (where it is called a multipoint conferencing unit, or MCU), a multi-way mixing component receives and decodes audio streams from a collection of clients, creates a unique mixture for each client by leaving the client's own stream out of the mixture, and encodes and sends the corresponding mixture back to the client, all at a given bit rate. The computational requirement r_v of the component can be made proportional to the number of mixtures it produces.

Although a conventional multi-way mixing service runs as a component in a single server node (i.e., an MCU), in this paper we consider a further generalization where multi-way mixing components may mix audio streams from other multi-way mixing components as well as from clients. In this way a multi-way mixing request G' for a set of client nodes can be satisfied in a distributed way by a Steiner tree $T \subseteq G$ passing through the set of client nodes and possibly some server nodes, as shown in Figure 4b. In such a tree T , if an interior node is a server with a set of neighbors N , it sends to each neighbor $n \in N$ a mixture of the streams from all

other neighbors $N - \{n\}$. When the interior node is a client, it sends to each neighbor $n \in N$ a mixture of the stream originating from itself and from all other neighbors $N - \{n\}$. When a leaf node is a client, it simply sends its own stream to its neighbor. When a leaf node is a server, it is discarded as it performs no function. The computational requirement $r_v(T)$ for each node $v \in T$ is approximately proportional to the number of mixtures it produces, i.e., the degree of v in T .

G. Measures of Delay: APD and MPD

The appropriate measure of delay $d(\psi(G'))$ depends on type of service request, as can be seen from examples (1)-(5) in the previous subsection.

For the multi-way mixing example (6), we use the following two variations. For the tree $T = \psi(G')$, let $d(v_1, v_2)$ be the sum of the delays d_e on the edges e along the unique path from v_1 to v_2 in T . We define the *Average Pairwise Delay* $d_{APD}(T)$ to be the *average* of $d(v_1, v_2)$ over all pairs of distinct clients v_1 and v_2 in the tree. Similarly we define the *Maximum Pairwise Delay* $d_{MPD}(T)$ to be the *maximum* of $d(v_1, v_2)$ over all pairs of distinct clients v_1 and v_2 in the tree.

APD and MPD are respectively the average and maximum delays experienced by the clients.

H. Heuristic Algorithm for Finding Minimum Cost Tree

Finding a minimum cost Steiner tree through a given set of nodes is an NP-hard problem [9]. We use the heuristic algorithm in Figure 5 for finding a near-minimum cost tree (MCT) for the mixing problem. The MCT heuristic performs, for each combination of potential servers, a Prim-like algorithm for estimating the spanning tree T^* that minimizes $Cost(T)$. (Note that the best tree T^* may not be a classical minimum weight spanning tree since APD and MPD are generally not functions of the *sum* of the edge weights.) As we verify in Section V.B, for the example of multi-party mixing, the heuristic performs essentially optimally on the scale of a dozen or more nodes.

```

Input: Graph  $G = \langle V, E \rangle$  where  $V = S \cup C$ ; metric  $Cost(T)$ .
Initialize: Best tree  $T^* = \emptyset$ .
For  $S_k = k^{\text{th}}$  subset of  $S$ ,  $k = 1, \dots, 2^{|S|}$ :
  Define subgraph  $\langle V_k, E_k \rangle$  where  $V_k = S_k \cup C$ .
  Let  $V_{\text{new}} = \{v_0\}$ , where  $v_0 \in V_k$  is arbitrary starting node.
  Let  $E_{\text{new}} = \emptyset$ .
  Repeat until  $V_{\text{new}} = V_k$ 
    Choose an edge  $(u, v) \in E_k$  s.t.  $\in V_{\text{new}}$ ,  $v \notin V_{\text{new}}$ , and
     $T = \langle V_{\text{new}} + v, E_{\text{new}} + (u, v) \rangle$  minimizes  $Cost(T)$ .
    Add  $v$  to  $V_{\text{new}}$ , and  $(u, v)$  to  $E_{\text{new}}$ .
  For all nodes  $v$  in tree  $T$ 
    If  $v \in S$  and  $\deg(v) = 1$ , remove  $v$  and its edge from  $T$ .
  If  $Cost(T) < Cost(T^*)$ 
    Set  $T^* \leftarrow T$ .
Return:  $T^*$ .

```

Figure 5. MCT algorithm for finding near-minimum cost trees for the mixing request problem.

III. RESOURCE ALLOCATION FOR MULTIPLE SERVICE REQUESTS

In Section II, we were given a network whose components are labeled by *costs*, and we considered how to allocate resources for a *single* service request G' , to minimize the cost of the service. A simple special case is the routing problem of finding a minimum cost route between a given sender and a given receiver in a network.

In this section, we are given a network whose components are labeled by *capacities*, and we consider how to allocate resources for a *set* of service requests G_i , $i = 1, \dots, n$, to maximize the total client utility or the total number of requests that can be served by the network. A special case is the “route packing” problem of finding routes for as many routing requests as possible subject to a network’s capacity constraints.

For concreteness, we again consider mixing requests satisfied by trees. However, the approach is applicable to any class of service requests that can be mapped to subgraphs of the network as described in the previous section.

For request G_i , let $\{T_{ij}\}_{j=1}^{N_i}$ be the set of trees that can potentially be used to satisfy the request. Let $r_e(T_{ij})$ and $r_v(T_{ij})$ be the bit rate on edge e and the computation rate at node v , respectively, that T_{ij} would need to satisfy request i . Let x_i be a variable that specifies how much of request i is satisfiable by this set of trees, either 0 or 1, and let x_{ij} be a variable that specifies how much of request i is satisfiable by tree T_{ij} , either 0 or 1, such that $x_i = \sum_j x_{ij}$. Note that the number of variables x_{ij} is generally exponential in the size of the network.

If a request is accepted, we wish to map the request to the network so as to maximize a global utility $\sum_i w_i U(x_i)$, where $U(\cdot)$ is a convex utility function and w_i is a weight or “willingness to pay” by the set of clients making the request. In this paper we as-

sume a linear utility $U(x_i) = x_i$. Thus, if all the w_i are equal to 1, then the problem is to maximize the number of requests satisfied subject to the capacity and delay constraints. Specifically, we wish to

$$\text{Maximize } \sum_{ij} w_i x_{ij} \text{ s.t.} \quad (1)$$

$$x_{ij} \geq 0, \text{ for each } i \text{ and } j, \quad (2)$$

$$\sum_j x_{ij} \leq 1, \text{ for each } i, \quad (3)$$

$$\sum_{ij: T_{ij} \ni e} x_{ij} r_e(T_{ij}) \leq c_e, \text{ for each } e, \quad (4)$$

$$\sum_{ij: T_{ij} \ni v} x_{ij} r_v(T_{ij}) \leq c_v, \text{ for each } v, \quad (5)$$

$$x_{ij} d(T_{ij}) \leq d_i, \text{ for each } i \text{ and } j, \text{ and} \quad (6)$$

$$x_{ij} \in \{0,1\}, \text{ for each } i \text{ and } j. \quad (7)$$

Constraint (2) ensures that the amount of request i satisfied by tree T_{ij} is non-negative. Constraint (3) ensures that the amount of request i satisfied by all trees doesn't exceed the amount of resources required by the request. Constraint (4) ensures that the total bit rate allocated across each edge does not exceed the edge capacity. Constraint (5) ensures that the total computational load allocated across each node does not exceed the node capacity. Constraint (6) specifies the delay constraints. Constraint (7) ensures that each request is mapped to at most one tree. This last constraint makes the program an integer program. Without it, the program is a linear program and fractional solutions are permitted: not only may requests be spread across multiple trees, but they need not be fully satisfied.

When the set of requests G_i , $i = 1, \dots, n$, is given in a batch, the problem is said to be an *offline* problem. When the requests are given sequentially, the problem is said to be *online*. In both cases, the objective is to maximize the number of requests from the set that can be satisfied subject to the resource constraints. In the online problem, the mappings must be determined sequentially and not retracted. Looking ahead is not possible. Hence there is a potential performance penalty paid if the problem is online.

We are interested in an online algorithm for mapping each request i to a tree $T_i = T_{ij(i)}$ for some $j = j(i)$, where the requests arrive sequentially and for each request, either the request must be denied, or the mapping for the request must be determined immediately and not be retracted. The performance (i.e., the utility after n requests) ϕ_n^1 of any online algorithm is bounded above by the optimal performance ϕ_n^2 of the above batch integer program, which in turn bounded above by the optimal performance ϕ_n^3 of

the linear program, which in turn is bounded above by the performance ϕ_n^4 of any feasible solution to the dual linear program:
 $\phi_n^1 \leq \phi_n^2 \leq \phi_n^3 \leq \phi_n^4$.

The last inequality follows by weak duality [6]. Specifically, if the primal problem is

$$(P): \max \sum_j c_j x_j \text{ s.t. } x_j \geq 0 \forall j, \sum_j a_{ij} x_j \leq b_i \forall i$$

and the dual problem is

$$(D): \min \sum_i b_i y_i \text{ s.t. } y_i \geq 0 \forall i, \sum_i a_{ij} y_i \geq c_j \forall j$$

then for any feasible solutions $\{x_j\}$ to (P) and $\{y_i\}$ to (D),

$$\sum_j c_j x_j \leq \sum_j \sum_i a_{ij} y_i x_j = \sum_i \sum_j a_{ij} x_j y_i \leq \sum_i b_i y_i.$$

(Here we have swapped the conventional roles of primal as a minimization and dual as a maximization.) The primal and dual problems can be viewed in a tableau of the form

$$\begin{bmatrix} c^T & 0 \\ A & b \end{bmatrix}.$$

In the following subsection, we provide an online algorithm whose performance ϕ_n^1 is within a factor C of the performance ϕ_n^4 of a feasible solution to the dual linear program (i.e., $\phi_n^1 \geq \phi_n^4/C$), and hence is within a factor C of the optimal performance ϕ_n^2 of the batch integer solution (i.e., $\phi_n^1 \geq \phi_n^2/C$). Such an algorithm is said to be C -competitive.

A. Online Joint Congestion-Load (OJCL) Algorithm

It can be seen from the tableau in Figure 6 that the linear program (1)-(6) has the following dual:

$$\text{Minimize } \sum_i y_i + \sum_e c_e y_e + \sum_v c_v y_v + \sum_{ij} d_i y_{ij} \text{ s.t.} \quad (8)$$

$$y_i \geq 0, y_e \geq 0, y_v \geq 0, \text{ and } y_{ij} \geq 0, \text{ for each } i, e, v, \text{ and } ij, \text{ and} \quad (9)$$

$$y_i + \sum_{e \in T_{ij}} y_e r_e(T_{ij}) + \sum_{v \in T_{ij}} y_v r_v(T_{ij}) + y_{ij} d(T_{ij}) \geq w_i, \text{ for each } ij. \quad (10)$$

Figure 6. Tableau for primal and dual linear programs.

The dual variables y_i , y_e , y_v , and y_{ij} can be respectively interpreted as the price for satisfying request i , the price for using unit bandwidth on edge e , the price for using unit computation on node v , and the price per unit of propagation delay.

To solve the problem in an online manner, we propose the Online Joint Congestion-Load (OJCL) algorithm in Figure 7.

1. Set all primal variables x_{ij} and all dual variables y_i, y_e, y_v , and y_{ij} to zero.
2. For each arriving request i , if there is a tree T_{ij} such that $\sum_{e \in T_{ij}} y_e r_e(T_{ij}) + \sum_{v \in T_{ij}} y_v r_v(T_{ij}) < w_i$ and $d(T_{ij}) < d_i$, then
 - a. Set $j(i) = j$
 - b. Map request i to tree $T_i = T_{ij(i)}$, and deduct the resource from the substrate.
 - c. Set $x_{ij(i)} = 1$ and $y_i = w_i$.
 - d. For each edge $e \in T_i$, set $y_e \leftarrow y_e \exp\left(\frac{\ln(1+|E|)}{c_e/r_{emax}}\right) + \frac{w_{max}}{|E|r_{emin}} \left[\exp\left(\frac{\ln(1+|E|)}{c_e/r_{emax}}\right) - 1 \right]$.
 - e. For each node $v \in \tilde{T}_i$, set $y_v \leftarrow y_v \exp\left(\frac{\ln(1+|V|)}{c_v/r_{vmax}}\right) + \frac{w_{max}}{|V|r_{vmin}} \left[\exp\left(\frac{\ln(1+|V|)}{c_v/r_{vmax}}\right) - 1 \right]$.

Figure 7. Online Joint Congestion-Load (OJCL) Algorithm.

After all variables are initialized to zero, the algorithm processes each request in turn. For each request, the algorithm checks whether the request can be met in terms of prices and delay-related demands. Specifically, the algorithm looks for a tree T_{ij} such that

$$\sum_{e \in T_{ij}} y_e r_e(T_{ij}) + \sum_{v \in T_{ij}} y_v r_v(T_{ij}) < w_i \quad (11)$$

and $D(T_{ij}) < d_i$.² If there is no such tree, the request is denied. Otherwise the request is mapped to this tree, and the prices of the corresponding resources are increased.

We now show that the algorithm is C -competitive.

Theorem. The algorithm is $\frac{w_{max}}{w_{min}} C$ -competitive, where

$$C = 1 + 2 \left(\frac{c_{emin}}{r_{emin}} \left[\exp \left(\frac{\ln(1 + |E|)}{c_{emin}/r_{emax}} \right) - 1 \right] + \frac{c_{vmin}}{r_{vmin}} \left[\exp \left(\frac{\ln(1 + |V|)}{c_{vmin}/r_{vmax}} \right) - 1 \right] \right).$$

$w_{min} = \min_i w_i$, $w_{max} = \max_i w_i$, $c_{emin} = \min_e c_e$, $c_{emax} = \max_e c_e$, $c_{vmin} = \min_v c_v$, $c_{vmax} = \max_v c_v$, $r_{emin} = \min_{e \in T_{ij}} r_e(T_{ij})$, $r_{emax} = \max_{e \in T_{ij}} r_e(T_{ij})$, $r_{vmin} = \min_{v \in T_{ij}} r_v(T_{ij})$, and $r_{vmax} = \max_{v \in T_{ij}} r_v(T_{ij})$.

Proof. When request i is satisfied by some tree T_i , the primal objective $\sum_i w_i x_i$ increases by at least w_{min} , while the dual objective $\sum_i y_i + \sum_e c_e y_e + \sum_v c_v y_v + \sum_{ij} d_i y_{ij}$ increases by at most

$$\begin{aligned} & w_i + \sum_{e \in T_i} c_e \left(y_e \left[\exp \left(\frac{\ln(1 + |E|)}{c_e/r_{emax}} \right) - 1 \right] + \frac{w_{max}}{|E|r_{emin}} \left[\exp \left(\frac{\ln(1 + |E|)}{c_e/r_{emax}} \right) - 1 \right] \right) \\ & + \sum_{v \in T_i} c_v \left(y_v \left[\exp \left(\frac{\ln(1 + |V|)}{c_v/r_{vmax}} \right) - 1 \right] + \frac{w_{max}}{|V|r_{vmin}} \left[\exp \left(\frac{\ln(1 + |V|)}{c_v/r_{vmax}} \right) - 1 \right] \right) \\ & \leq w_i + \sum_{e \in T_i} \left(y_e + \frac{w_{max}}{|E|r_{emin}} \right) c_{emin} \left[\exp \left(\frac{\ln(1 + |E|)}{c_{emin}/r_{emax}} \right) - 1 \right] \\ & + \sum_{v \in T_i} \left(y_v + \frac{w_{max}}{|V|r_{vmin}} \right) c_{vmin} \left[\exp \left(\frac{\ln(1 + |V|)}{c_{vmin}/r_{vmax}} \right) - 1 \right] \\ & = w_i + \sum_{e \in T_i} \left(y_e r_{emin} + \frac{w_{max}}{|E|} \frac{c_{emin}}{r_{emin}} \left[\exp \left(\frac{\ln(1 + |E|)}{c_{emin}/r_{emax}} \right) - 1 \right] \right) \\ & + \sum_{v \in T_i} \left(y_v r_{vmin} + \frac{w_{max}}{|V|} \frac{c_{vmin}}{r_{vmin}} \left[\exp \left(\frac{\ln(1 + |V|)}{c_{vmin}/r_{vmax}} \right) - 1 \right] \right) \\ & \leq w_i + 2w_{max} \left(\frac{c_{emin}}{r_{emin}} \left[\exp \left(\frac{\ln(1 + |E|)}{c_{emin}/r_{emax}} \right) - 1 \right] + \frac{c_{vmin}}{r_{vmin}} \left[\exp \left(\frac{\ln(1 + |V|)}{c_{vmin}/r_{vmax}} \right) - 1 \right] \right) \end{aligned}$$

² Any such tree will do for the proof of competitiveness, since the bound on C is loose. However performance better than C can be obtained (as shown in the experiments of Section V) by finding a tree T_{ij} that minimizes $\sum_{e \in T_{ij}} y_e r_e(T_{ij}) + \sum_{v \in T_{ij}} y_v r_v(T_{ij})$ s.t. $d(T_{ij}) \leq d_i$, or else minimizes $\sum_{e \in T_{ij}} y_e r_e(T_{ij}) + \sum_{v \in T_{ij}} y_v r_v(T_{ij}) + \lambda_i d(T_{ij})$ for some Lagrange multiplier $\lambda_i > 0$. This tree can be found with the minimum cost tree (MCT) algorithm in Figure 5.

$$\leq w_{\max} C,$$

where the first inequality follows from the fact that $c[\exp\{\ln(1 + m)/c\} - 1]$ is monotonically decreasing in c , and the second inequality follows from (11). When the request is not satisfied, there is no change to either objective. Hence the ratio between dual and primal objectives is always at most $(w_{\max}/w_{\min})C$. It remains to check that the dual and primal variables up through request i remain feasible. The dual variables remain feasible because $y_i = w_i$ whenever (11) holds, and the variables y_e and y_v never decrease. The primal variables remain feasible because the algorithm maps at most c_e/r_{\max} trees through edge e , and at most c_v/r_{\max} trees through node v . To see that, observe that for each edge e , the value y_e is the sum of a geometric series with initial value $\frac{w_{\max}}{|E|r_{\min}} \left[\exp\left(\frac{\ln(1+|E|)}{c_e/r_{\max}}\right) - 1 \right]$ and multiplier $\exp\left(\frac{\ln(1+|E|)}{c_e/r_{\max}}\right)$. Thus, after c_e/r_{\max} trees are mapped through edge e , the value of y_e is

$$\begin{aligned} y_e &= \frac{w_{\max}}{|E|r_{\min}} \left[\exp\left(\frac{\ln(1+|E|)}{c_e/r_{\max}}\right) - 1 \right] \frac{\exp\left(\frac{c_e}{r_{\max}} \frac{\ln(1+|E|)}{c_e/r_{\max}}\right) - 1}{\exp\left(\frac{\ln(1+|E|)}{c_e/r_{\max}}\right) - 1} \\ &= \frac{w_{\max}}{|E|r_{\min}} (1 + |E| - 1) = \frac{w_{\max}}{r_{\min}}. \end{aligned}$$

The algorithm never maps trees to edges for which $y_e r_{\min} \geq w_{\max}$, and likewise never maps trees to nodes for which $y_v r_{\min} \geq w_{\max}$, for otherwise (11) would be violated.

Since the performance $\sum_i w_i x_i$ of the algorithm is within a factor C of the performance of a feasible solution to the dual, it is also is within a factor C of the performance of the optimal batch integer solution, and hence is C -competitive. \square

The size of C is estimated in Section V.

IV. ADAPTATION TO NETWORK DYNAMICS BETWEEN SERVICE REQUESTS

Although we were able to prove that the OJCL algorithm is C -competitive in an online framework, in real scenarios service requests not only arrive sequentially; they also eventually depart. This allows resources to be released, so that as time goes on an unlimited number of service requests can be processed. Moreover, between arrivals and departures, cross traffic and other events intervene, requiring the ongoing services to adapt. In this section, we discuss how we adapt the OJCL algorithm to a dynamic environment, denoting the new algorithm A-OJCL.

The core of our adaptation approach is to generalize congestion control algorithms already in place in the Internet: TCP and TCP-friendly rate control (TFRC) [10]. The purpose of TCP and TFRC is to adapt the bit rates on existing routes in the presence of time-varying congestion. We start with TCP Vegas, which is a variant of TCP that adapts its bit rate using delay as a congestion

signal. TCP Vegas has been shown to solve the following optimization problem in equilibrium [11]. Given a network $G = \langle V, E \rangle$ with capacities c_e on each edge $e \in E$, and a set of routes (or paths) P_i from s_i to t_i in G : Determine a bit rate x_i on each path P_i in order to

Maximize $\sum_i w_i U(x_i)$ s.t.

$$x_i \geq 0, \text{ for each } i, \text{ and} \quad (12)$$

$$\sum_{i: P_i \ni e} x_i \leq c_e, \text{ for each } e. \quad (13)$$

Here, $U(\cdot)$ is a convex utility function (logarithmic) and w_i is a weight or “willingness to pay” by the user of the route (equivalent to ganging together w_i simultaneous TCP connections). Constraint (13) ensures that the total bit rate allocated across each edge does not exceed the edge capacity. The solution can be found mathematically by the method of Lagrange multipliers. For some set of non-negative Lagrange multipliers $y_e, e \in E$, the optimal values of x_i maximize the Lagrangian

$$\sum_i w_i U(x_i) - \sum_e y_e \sum_{i: P_i \ni e} x_i = \sum_i w_i U(x_i) - \sum_i x_i \sum_{e \in P_i} y_e = \sum_i (w_i U(x_i) - x_i Price(P_i)).$$

Here we define $Price(P_i) = \sum_{e \in P_i} y_e$ as the price of using path P_i (per unit of bandwidth). If $U(x_i)$ is differentiable, the maximum is achieved when $w_i U'(x_i) = Price(P_i)$. In particular, if $U(x_i) = \ln x_i$, the maximum is achieved when $Price(P_i) = w_i/x_i$. In [11] it is shown that $x_i = w_i/Price(P_i)$ is the equilibrium bit rate for TCP Vegas when $Price(P_i)$ is the sum of queuing delays y_e on the edges $e \in P_i$ and w_i is the number of bits desired in flight over path P_i .

The Adaptive OJCL algorithm generalizes TCP Vegas by defining each edge price y_e to be the queuing delay on edge e . Between the arrivals and departures of service requests, the bit rate x_i on each allocated tree T_i is allowed to vary dynamically in response to congestion according to $x_i = w_i/Price(T_i)$, where $Price(T_i) = \sum_{e \in T_i} y_e$ is the sum of the queuing delays along the edges in T_i . When a new service request G_i arrives, A-OJCL uses these edge prices when allocating the minimum cost tree T_i , thus avoiding edges that are congested (i.e., having long queuing delay).

The A-OJCL algorithm uses a queuing model also for the node prices y_v . In an M/M/1 queue with Poisson arrivals at rate λ and exponential service at rate μ , the expected time waiting in the queue is $(\lambda/\mu)/(\mu - \lambda)$ [12]. Thus A-OJCL uses $y_v = (\lambda/\mu)/(\mu - \lambda)$ as the node price when server v with capacity $\mu = c_v$ Mips is loaded with jobs requiring a total of λ Mips. This heuristic allows the price of the computational resource at node v to gradually increase as it is loaded from $\lambda = 0$ up to its capacity $\mu = c_v$, and then to decrease again as the node is unloaded.

In short, the A-OJCL algorithm is shown in Figure 8. In the next section, we show that compared to conventional approaches, A-OJCL is able to route around both congested edges and loaded nodes, is able to increase total client utility by increasing the number of services supportable by the network, and is able to make better use of existing communication and computation resources.

On each arriving service request i :

1. Choose T_i to minimize $\sum_{e \in T_i} y_e + \sum_{v \in T_i} y_v r_v(T_i) + d(T_i)$, where y_e is the current queuing delay on edge e and y_v is the current “queuing delay” on node v .
2. For each $v \in T_i$, increase the load $\lambda_v \leftarrow \lambda_v + r_v(T_i)$, and set $y_v \leftarrow (\lambda_v/c_v)/(c_v - \lambda_v)$.

On each a departing service i :

For each $v \in T_i$, decrease the load $\lambda_v \leftarrow \lambda_v - r_v(T_i)$, and set $y_v \leftarrow (\lambda_v/c_v)/(c_v - \lambda_v)$.

Between arrivals and departures:

1. Run TCP Vegas (or other TCP-friendly variant) to adapt the bit rate x_i on each T_i . The queuing delay y_e is determined continuously by the network.

Figure 8. Adaptive OJCL (A-OJCL) Algorithm.

V. EXPERIMENTAL RESULTS

In this section, we conduct experiments to verify our approach and evaluate its performance compared with existing approaches.

A. Experimental Setup

The network used in our experiments is a hypothetical 12-node network based on 12 PlanetLab nodes [13] in Hong Kong, Taiwan, the United States, and Canada, shown in Figure 9a. We measured round trip times between all 66 pairs of PlanetLab nodes and divided by two to get one-way propagation delays, a subset of which are shown on the 14 edges in Figure 9b (in ms). For our hypothetical network, we arbitrarily set the bandwidths of the 14 edges and the computing capacities of the 12 nodes, which are shown in Figure 9c (in Kbps and number of computable mixtures, respectively).

In our first set of experiments (Subsection B), we focus on optimizing a single mixing service and show through experiments that our approach can reduce both the average and maximum pairwise delays between clients by 40% or more compared to existing MCU-based and P2P-based mixing approaches.

In our second set of experiments (Subsection C) we validate that the OJCL algorithm conforms to theory. We first consider only two-party calls, i.e., the route-packing problem, for which the optimal performance is known. We plot the values of the primal and dual objective functions of our online algorithm and show that the optimal performance is indeed bounded between these quantities. Moreover, the performance of the online algorithm is within a factor 1.3 of the optimum and within a factor 3 of the upper bound. For multiparty calls, the performance of the online algorithm is also within a factor 3 of the upper bound, suggesting that it may also be within a factor 1.3 of the optimum, although the optimum is not known in this case.

In a final set of experiments (Subsection D), we validate the Adaptive OJCL algorithm. We show that the A-OJCL algorithm is able to map around congested edges and nodes, and can adapt the service bit rates to cross traffic. We then show that in steady state, as services come and go (and resources are allocated and freed) that A-OJCL can map almost 20% more services onto the network by making about 20% better use of both communication and computing resources, compared to other approaches.

B. Resource Allocation for a Single Service Request

In this set of experiments, all 12 nodes in the network of Figure 9 are client nodes. We wish to find the best mixer for these clients, in (1) the 14-edge network of Figure 9b, and (2) the network of Figure 9b completed into a 66-edge completely connected network. In both cases we set the communication and computation costs to zero and consider only delay as measured by APD and MPD, that is, $Cost(T) = d_{APD}(T)$ and $Cost(T) = d_{MPD}(T)$. In each case, we use the MCT algorithm in Figure 5 to obtain the near-minimum delay mixer, and compare the solution to the minimum delay mixer obtained through exhaustive search (which is possible to compute on a network of this size). We also compare these tree-based mixers to (a) the conventional multipoint conferencing unit (MCU) approach in which a single mixer is placed at one of the four central nodes, and (b) the MutualCast approach of time-sharing the single mixing location among all clients.

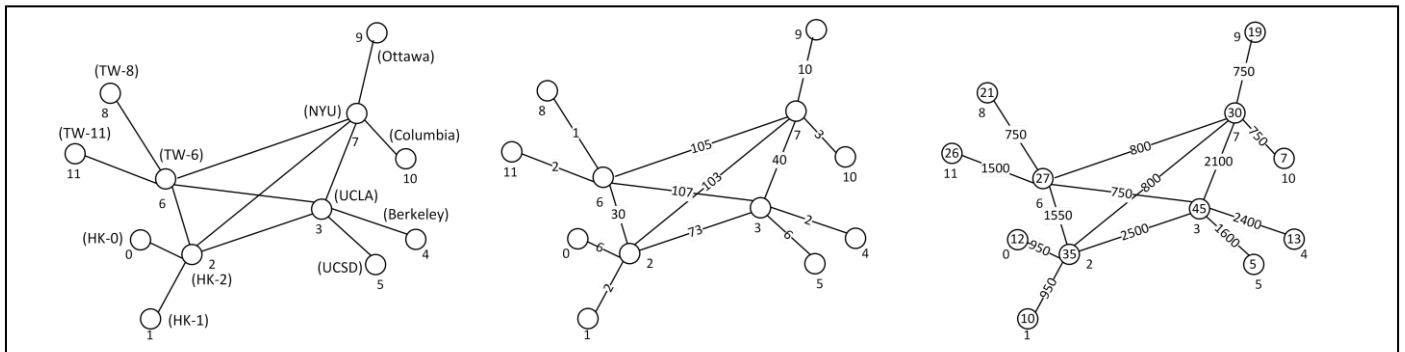


Figure 9. Topology of the network. (a) PlanetLab nodes. (b) Measured propagation delays. (c) Assigned edge and node capacities.

From Tables 1 and 2 we can see that tree-based mixers reduce both APD and MPD by 40% or more compared to the other schemes. In addition, on the 14-edge network, the minimum APD tree is the same as the minimum MPD tree, and this tree, which is shown in Figure 10a, is the same as the tree obtained by the MCT algorithm, whether optimizing for APD or MPD. On the 66-edge complete network, there is a slight difference between the minimum APD and minimum MPD trees, which are shown in Figure 10b,c. As expected, the APD is slightly smaller for the minimum APD tree than for the minimum MPD tree, and vice versa for the MPD. However, the MCT algorithm find the optimal tree in both cases. The running time of the MCT algorithm is 2-4 orders of magnitude faster than exhaustive search, and hence can be used for larger networks.

Table 1. Minimum Delay Mixer results on the 14-edge network of **Figure 9b**.

Scheme	APD	MPD	Running Time
Conventional MCU	152 ms	254 ms	N/A
MutualCast	130 ms	254 ms	N/A
Tree (MCT-APD)	74 ms	155 ms	730 ms
Tree (MCT-MPD)	74 ms	155 ms	738 ms
Tree (exhaustive-APD)	74 ms	155 ms	59973 ms
Tree (exhaustive-MPD)	74 ms	155 ms	59681 ms

Table 2. Minimum Delay Mixer results on the 66-edge completely connected network.

Scheme	APD	MPD	Running Time
Conventional MCU	139 ms	245 ms	N/A
MutualCast	119 ms	245 ms	N/A
Tree (MCT-APD)	66 ms	136 ms	725 ms
Tree (MCT-MPD)	68 ms	129 ms	729 ms
Tree (exhaustive-APD)	66 ms	136 ms	4567458 ms
Tree (exhaustive-MPD)	68 ms	129 ms	4557683 ms

C. Resource Allocation for Multiple Service Requests

In this set of experiments, we consider the interior PlanetLab nodes to be servers in a branch office (or analogously servers in regional data centers), where mixing can be performed. Clients connect to their nearest servers, while servers mix and relay the

mixtures to support multiparty conferencing. Propagation delays for each edge are again taken from the PlanetLab measurements, but edge bandwidths and node computational capacities are set arbitrarily to the values shown in Figure 9c.

During multiparty conferencing, each request for a mixing service corresponds to a multiparty call. The number of parties in a service request is chosen randomly according to the following geometric distribution: 50% 2-party calls, 25% 3-party calls, 12.5% 4-party calls, 6.25% 5-party calls, and so forth. Once the number of parties (i.e., clients) has been randomly chosen from the distribution, each client is randomly located (uniformly) at one of the 12 network nodes. We assume that each service request demands 64 Kbps for each edge, and 1 unit of computing power for each mixture produced. Service requests arrive sequentially, but in this set of experiments they do not terminate.

The OJCL algorithm either accepts or rejects each service request. Once a request is accepted, the algorithm maps the request to a tree that connects all parties in the call, using the MCT algorithm to minimize bandwidth and computing costs subject to a constraint on the average pairwise delay.

We verify that the OJCL algorithm conforms to theory. First we consider only two-party calls. Since packing the maximum number of two-party calls into a network in a fractional way is a standard multi-commodity flow problem [14], it can be solved using glpk [15]. In comparison to this fractional batch solution, in Figure 11 we plot the values of the primal and dual objective functions of our online integral algorithm. It can be seen that our primal objective (which is the number of calls actually packed by our algorithm) is within a factor 1.3 of the optimal batch fractional solution and within a factor 3 of its dual bound.

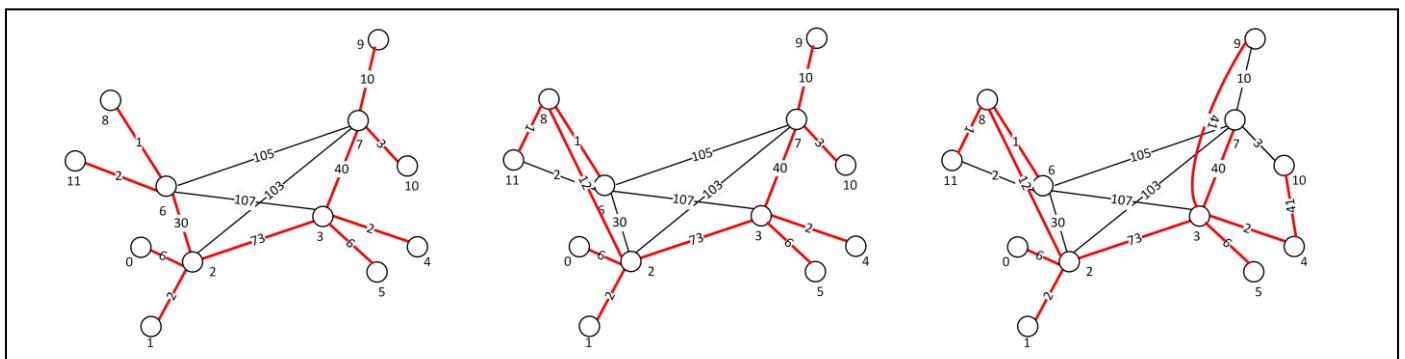


Figure 10. (a) Topologies generated by (a) Virtual Mixer, (b) PT-APD, and (c) PT-MPD.

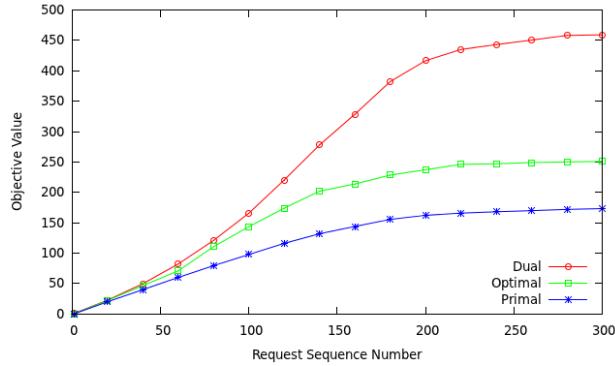


Figure 11. Performance for two-party calls only. The online integer algorithm (primal) is within 1.3x of the optimal batch fractional solution and within 3x of its dual bound.

Next we extend to all-party calls. Computing the optimal fractional solution for all-party calls is NP-hard. However, its performance must be bounded by the primal and dual values of our algorithm’s solution. The (primal) performance of OJCL is again within a factor 3 of its dual bound, shown in Figure 12. Hence it must again be close to the optimal batch fractional solution.

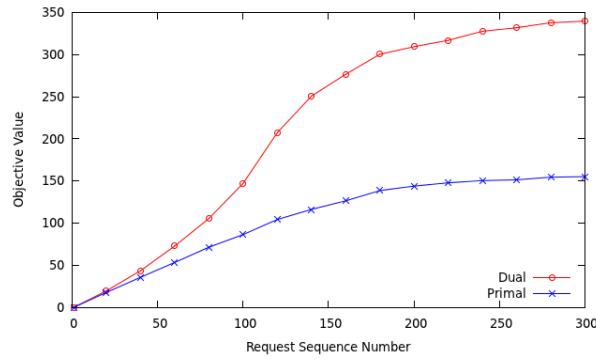


Figure 12. Performance of all-party calls. The online integer algorithm (primal) is again within 3x of its dual bound, and hence must be close to optimal.

In comparison, the value of C for this network can be estimated from its definition

$$C = 1 + 2 \left(\frac{c_{emin}}{r_{emin}} \left[\exp \left(\frac{\ln(1 + |E|)}{c_{emin}/r_{emax}} \right) - 1 \right] + \frac{c_{vmin}}{r_{vmin}} \left[\exp \left(\frac{\ln(1 + |V|)}{c_{vmin}/r_{vmax}} \right) - 1 \right] \right)$$

and from the values $c_{emin}/r_{emin} = c_{emin}/r_{emax} = 750/64$, $c_{vmin}/r_{vmin} = 5/1$, $c_{vmin}/r_{vmax} = 5/3$, $|E| = 14$, $|V| = 12$, as $C = 43.7$.

D. Adaptation to Network Dynamics between Service Requests

In this subsection we continue with the previous conditions, but let the service requests terminate. In addition we use Network Simulator 2 (ns2) to simulate real traffic in the network, with edge capacities and propagation delays set as in Figure 9. In the ns2 network, the queuing delay across each edge e is used as the edge cost y_e in the A-OJCL algorithm.

1) Detailed Performance Verification

In this set of experiments we perform a detailed verification of the service mapping and rate adaptation aspects of the A-OJCL algorithm. We consider the sequence of five mixing service requests listed in Table 3, under 3 conditions: no excess congestion or load (Case A), cross traffic on edge (2,3) (Case B), and cross traffic on edge (2,3) plus extra load on node 7 (Case C). We show that A-OJCL automatically maps around congested edges and nodes, and adapts to traffic.

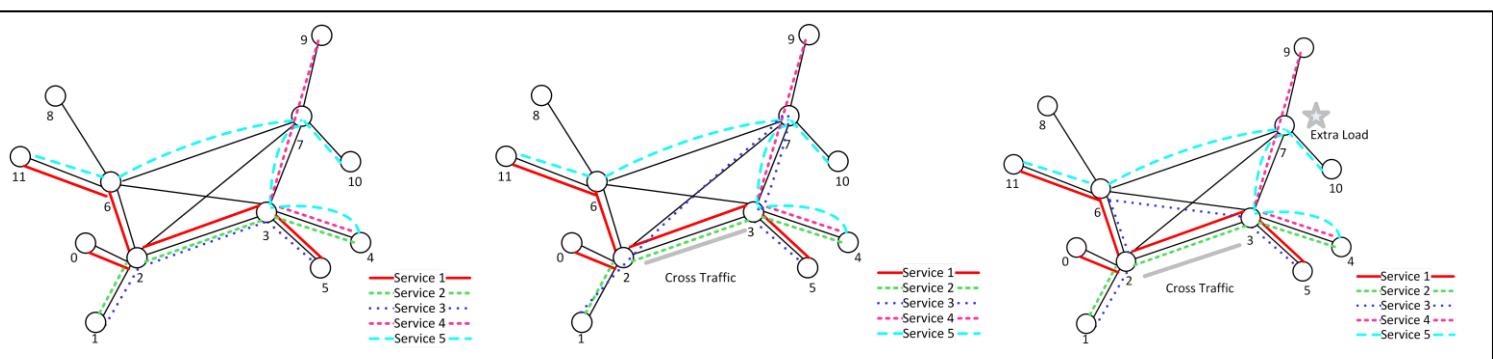


Figure 13. Service mappings in cases (A), (B), and (C).

Table 3. Mixing requests and durations of service.

Service No.	Clients	Duration
1	0, 5, 11	[0, 380]
2	1, 4	[50, 255]
3	1, 5	[120, 380]
4	4, 9	[100, 320]
5	4, 10, 11	[150, 270]
Cross Traffic	2, 3	[25, 70], [80, 160], [230, 250], [300, 400]

a) *Service Mapping*

The A-OJCL algorithm maps the requests to the network in each case as shown in Figure 13. In Case A, there is no excess congestion or load. Therefore A-OJCL maps the five services to the network along shortest paths between clients. In Case B, cross traffic appears across edge (2,3). The cross traffic has a 600 Kbps constant bit rate during the intervals listed in Table 3. Since the cross traffic is active at the time of the request for Service 3 (at time $t = 120$ s), the price y_e on edge (2,3) is relatively high at that time, as shown in Figure 14. Consequently A-OJCL maps Service 3 through edges (2,7) and (7,3) instead of through the congested edge (2,3). In Case C, in addition to the cross traffic on edge (2,3), a heavy computing load appears on node 7, sending the price y_v on that node to a high value. Thus A-OJCL maps Service 3 through edges (2,6) and (6,3) instead of by way of the loaded node 7 or through the congested edge (2,3). The evolution of the prices y_e and y_v are shown in Figure 14 and Figure 15 for the different cases. Using prices, A-OJCL is automatically able to map around congested edges and nodes.

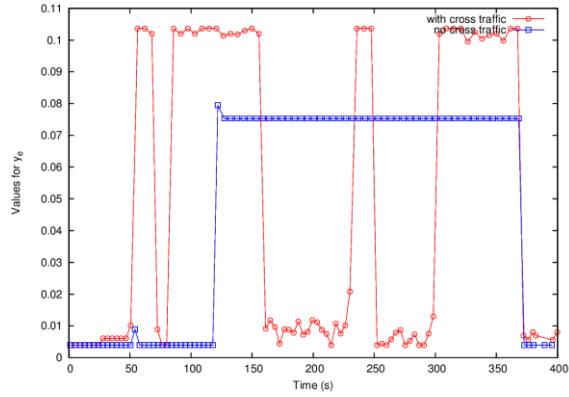


Figure 14. Price y_e of edge (2,3) in cases A and B/C.

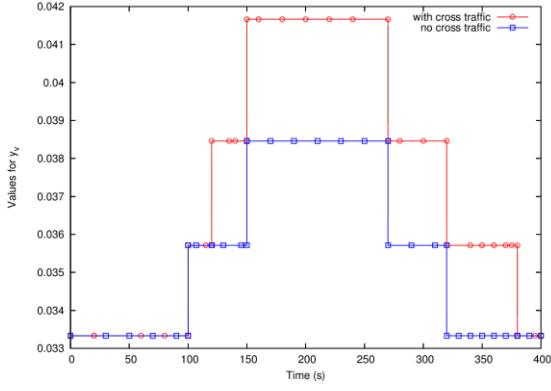


Figure 15. Price y_v of node 7 in Cases A and B.

b) *Rate Adaptation*

After each service is mapped to the network, it adapts its bit rate according to network dynamics. We plot the bit rates of the five services under two cases: Case A and Case B. We can see from Figure 16 and Figure 17 how the rates adapt as services arrive and depart.

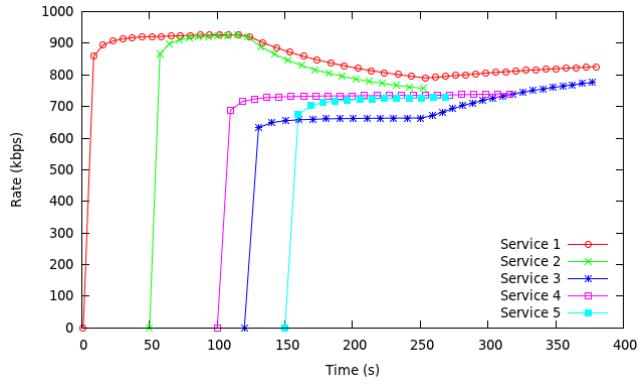


Figure 16. Rate adaptation in Case A (no cross traffic).

In Case A, Figure 16 shows (with reference to the service mappings in Figure 13a and the edge capacities in Figure 9c) that at time 0, Service 1 is mapped to a tree with bottleneck link (0,2) at 950 Kbps and adapts its bit rate close to this limit. Similarly, at time 50, Service 2 is mapped to a path with bottleneck link (1,2) at 950 Kbps. At time 100, Service 4 is mapped to a path with bottleneck link (7,9) at 750 Kbps. Then, at time 120, Service 3 is mapped to a path that shares link (2,3) with Services 1 and 2. This becomes the bottleneck as all three split its 2400 Kbps capacity. Finally at time 150, Service 5 is mapped to a tree with bottleneck link (7,10) at 750 Kbps. At time 255, Service 2 departs and allows Services 1 and 3 to split link (2,3).

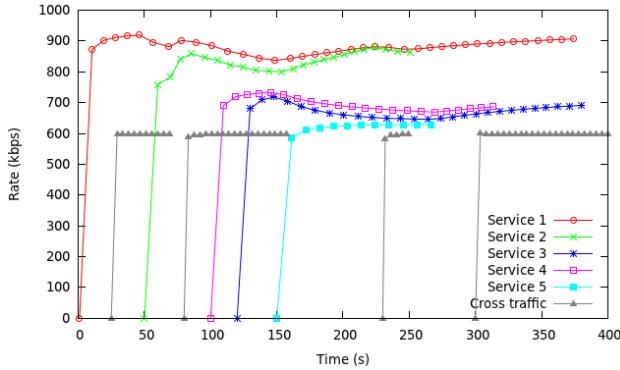


Figure 17. Rate adaptation in Case B (cross traffic).

In Case B, Figure 17 shows what happens when there is 600 Kbps constant bit rate cross traffic on edge (2,3), giving the edge only 1800 Kbps residual capacity. From time 25 to time 50, the 950 Kbps load from Service 1 fits comfortably within the 1800 Kbps residual capacity, but at time 50, when Service 2 starts, Services 1 and 2 must split the 1800 Kbps on edge (2,3). This situation largely remains until Service 2 departs at time 255, although there are temporary reprieves in the cross traffic during the intervals [70,80] and [160,230]. Thus at time 120 when Service 3 arrives, there is a large queuing delay on edge (2,3) as shown in Figure 14. Hence Service 3 is mapped around the congested edge (2,3), and instead is mapped through edges (2,7) and (7,3), with edge (2,7) becoming Service 3's bottleneck at 800 Kbps. Meanwhile, at time 100, Service 4 had also been mapped through edge (7,3), with edge (7,9) becoming its bottleneck at 750 Kbps. But these rates are not to be realized for long. At time 150, Service 5 is also mapped across edge (7,3), which then becomes the bottleneck for all three services (3, 4, and 5) as they share its 2100 Kbps.

2) Statistical Performance Evaluation

In this final set of experiments, we evaluate the long term steady state behavior of A-OJCL as services come and go. In our simulations, services (i.e., calls) arrive according to a Poisson process at arrival rate λ (which we vary from 5 to 50 calls per minute), and they have lifetimes that are independently and identically distributed according to an exponential distribution with average duration 4 minutes. We assume that each service requires a minimum of 64 Kbps on each allocated edge and requires exactly 1 unit of computing power per mixture on each allocated node.

We compare A-OJCL to congestion-control-based algorithms, load-balance-based algorithms, and conventional MCU-based algorithms. As the representative of congestion-control-based (CC) algorithms, we use a version of A-OJCL that disregards load on the nodes (but must reject a request if the resulting mapping would put more load on a node than its computational capacity). Similarly, as the representative of load-balance-based (LB) algorithms, we use a version of A-OJCL that disregards congestion on the edges (but must reject a request if the resulting mapping would put more bandwidth on an edge than its communication capacity).

As the representative of conventional MCU-based algorithms, we use an algorithm that simply maps each service request to a star-shaped topology with the central node (the MCU) on the server nearest to the clients in terms of average pairwise delay.

We evaluate the performances of A-OJCL, CC, LB and Conventional algorithms with respect to the number of services that can be mapped (i.e., packed) onto the network with a given set of resources, and resource utilization. The number of services that can be mapped to the system is the steady state average of the number of service requests mapped to the network at any given time. Resource utilization is the steady state average of the total amount of all computing (resp., communication) resources allocated at any given time divided by the total amount of computing (resp., communication) resources in the network. All averages are taken over 100 minutes of simulation time.

a) Number of Services Mapped to the Network

In this experiment, we use edge and node capacities shown in Figure 9c. The average number of services mapped to the network in steady state is shown in Figure 18 as a function of the service request arrival rate. When the arrival rate is low, the average number of services mapped to the network is equal to the arrival rate times the average call duration. However as the arrival rate increases, this average number of services in the network cannot be supported by the available resources, and some number of service requests must be rejected. At high arrival rates, many service requests are rejected, but the network is able to support some number of calls, which in some sense is the effective capacity of the network in terms of calls. From the figure we can see that A-OJCL is able to increase the effective capacity of the network compared to the other algorithms by up to 20%.

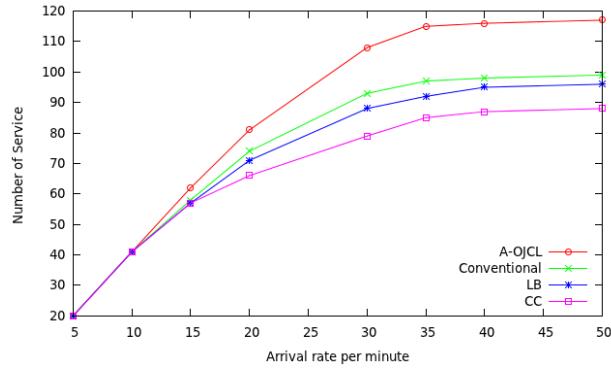


Figure 18. Average number of services in the network at any given time.

b) Bandwidth Resource Utilization

In this experiment, the bandwidth settings remain as in Figure 9c, but we increase the computing capacities of the nodes so that computational resources are not a bottleneck. Figure 19 shows that when bandwidth resources are constrained, the A-OJCL and CC

algorithms achieve higher bandwidth utilization than the other algorithms. This is due to the fact that the bandwidth-oriented optimization can find better ways to allocate and utilize bandwidth resources, including routing around congested edges.

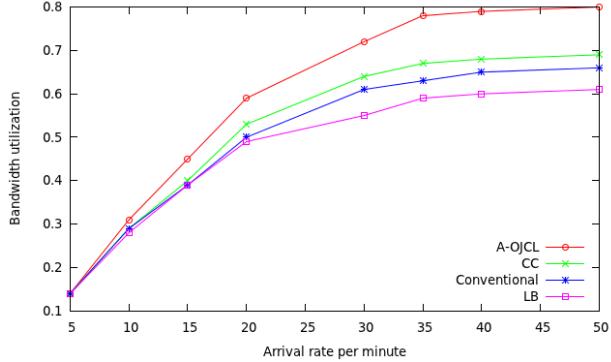


Figure 19. Bandwidth utilization when computing resources are plentiful.

c) CPU Resource Utilization

In this experiment, the node capacities remain as in Figure 9c, but we increase the bandwidths of the edge so that communication resources are not a bottleneck. Figure 20 shows that when computing resources are constrained, the A-OJCL and LB algorithms achieve higher CPU utilization than the other algorithms. This is due to the fact that the CPU-oriented optimization can find better ways to use computing resources, including routing around overloaded nodes.

Thus A-OJCL outperforms the other algorithms in either situation. These results are consistent with the ability of A-OJCL to increase the network capacity to support a higher number of calls.

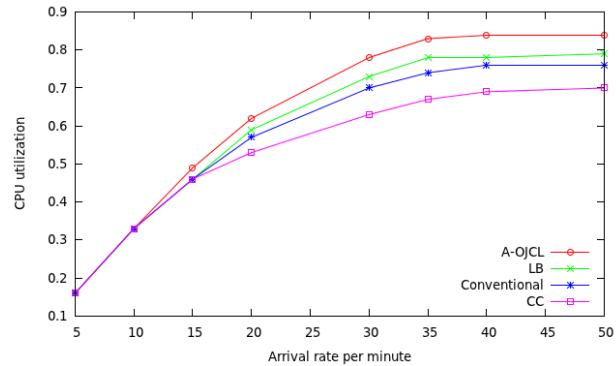


Figure 20. CPU utilization when communication resources are plentiful.

VI. CONCLUSIONS

In this paper we established a framework for joint allocation of communication and computing resources in a network in terms of mapping subgraphs representing service requests. Routing is a simple special case of the framework. We generalize shortest

path routing to finding minimum cost subgraphs, where prices are associated with computing and communication resources as well as with delay. Further, we generalize route packing under constraints on communication capacity to packing subgraphs under constraints on communication and computation capacities as well as delay. We proposed an online algorithm for sequentially processing service requests as they arrive, and we prove that after any number of requests have arrived the number of requests that the algorithm has packed is within a constant factor C of that of the maximal packing. Using multiparty audio mixing service as an example, we conduct numerous experiments, which show that compared to conventional MCU-based mixing algorithms as well as other load-balancing-based and congestion-control-based algorithms, our algorithm improves the effective capacity of the network by up to 20%.

REFERENCES

- [1] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris, "Resilient overlay networks," *ACM SOSP*, Oct. 2001.
- [2] L. Subramanian, I. Stoica, H. Balakrishnan, and R. H. Katz, "OverQos: an overlay based architecture for enhancing internet Qos," *NSDI*, Mar. 2004.
- [3] M. Szymaniak, G. Pierre and M. Van Steen, "Netairt: A DNS-based redirection system for apache," *IADIS*, Nov. 2003.
- [4] D. Karger, A. Sherman, A. Berkheimer, B. Bogstad, R. Dhanidina, K. Iwamoto, B. Kim, L. Matkins, and Y. Yerushalmi, "Web caching with consistent hashing," *Computer Networks*, vol. 31, pp. 1203-1213, 1999.
- [5] N. Chowdhury, M. R. Rahman and R. Boutaba, "Virtual network embedding with coordinated node and link mapping," *IEEE INFOCOM*, Apr. 2009.
- [6] R. Sedgewick and K. Wayne, *Algorithms*, 4th ed., Addison-Wesley, 2004.
- [7] N. Buchbinder and J. Naor, "The Design of Competitive Online Algorithms via a Primal-Dual Approach," *Foundations and Trends in Theoretical Computer Science*, vol. 3, pp. 93-263, 2009.
- [8] F. Kelly, A.K. Maulloo, and D.K.H. Tan, "Rate Control in Communication Networks: Shadow Prices, Proportional Fairness and Stability," *J. Operational Research Society*, 49, pp. 237-252, 1998.
- [9] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Co., 1979.
- [10] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-based Congestion Control for Unicast Applications," *ACM SIGCOMM*, Aug. 2000.
- [11] S. Low, L. Peterson, and L. Wang, "Understanding TCP Vegas: Theory and Practice," Princeton University CS Dept. Technical Report TR 616-00, Feb. 2000.

[12] S.M. Ross, *Introduction to Probability Models*, 5th ed., Academic Press, 1992.

[13] “PlanetLab,” <http://www.planet-lab.org/>

[14] “Multi-commodity flow problem,” Wikipedia.

[15] “GLPK,” GNU Project - Free Software Foundation (FSF)