

# From Multiple Decompositions to TRUMP: Traffic Management Using Multipath Protocol

Jiayue He, Martin Suchara, Ma'ayan Bresler, Jennifer Rexford, and Mung Chiang  
Princeton University, USA

Email: {jhe, msuchara, mbresler, jrex, chiangm}@princeton.edu

**Abstract**—Traffic management is the adaptation of source rates and routing to efficiently utilize network resources. Traffic management today includes congestion control, routing and traffic engineering. In this paper, we perform a top-down redesign of traffic management using recent innovations in optimization theory. First, we propose a *new objective function* that captures the goals of both end users and network operators. Second, using various optimization decomposition techniques, we *generate* four distributed algorithms that divide traffic over multiple paths based on feedback from the network links. These distributed algorithms are provably stable and optimal. Third, combining the best features of these distributed algorithms, we *construct* TRUMP: a new traffic management protocol that is distributed, adaptive, robust, flexible and easy to manage. Packet-level simulations show TRUMP behaves well with realistic topologies, feedback delays, capacities, and traffic loads. Overall, we show that using optimization decomposition as a foundation, simulations as a building block, and engineering intuition as a guide can be a principled approach to protocol design.

## I. INTRODUCTION

*Traffic management* is the adaptation of source rates and routing to achieve the goals of users and operators. Traffic management has three players: users, routers, and operators. In today's Internet, users run congestion control to adapt their sending rates at the edge of the network. Inside a single Autonomous System (AS), routers run shortest-path routing based on link weights. Operators monitor the network for congestion, and tune link weights to direct traffic away from congested links [2]. The current division of labor between the three players slowly evolved over time without any conscious design, resulting in a few shortcomings. First, operators tune link weights assuming that the traffic is inelastic, and end hosts adapt their sending rates assuming routing is fixed, leading to suboptimal interactions [3]. Second, tuning link weights is an indirect way to control traffic flow through a network; further,

This paper is an extended version of a conference paper that appeared as [1]. The key additions of this journal version are as follows. First, Section II describes several specific scenarios for the distributed traffic management protocols to be implemented in the current Internet. Second, this paper contains a proof of convergence for TRUMP in Section VI-B. Finally, this paper contains new experimental results in Section V-B, VII-B, VII-C and VII-E. In Section V-B, additional experiments on a multihoming topology provide insight on the relationship between the number of flows sharing bottleneck links, and the performance of distributed algorithms. In Section VII-B, new experiments provide insight on setting specific protocol parameters for fast convergence (for a wide range of topologies and capacities). In Section VII-C, new experiments capture both the sending rates and the actual throughput for distributed protocols, allowing for a more concrete comparison. Section VII-E studies how the number of flows and the number of paths per flow impact the performance of TRUMP.

the link-weight setting problem is NP-hard, forcing operators to resort to heuristics that can lead to highly suboptimal solutions [4]. Finally, since this offline optimization occurs at the timescale of hours, it does not adapt to changes in the offered traffic, causing an inefficient use of the underlying resources.

*In this paper, we rethink Internet traffic management using optimization theory as a foundation.* Optimization theory has been successfully used to analyze and design the various components of traffic management. TCP congestion control has been reverse engineered as implicitly solving an optimization problem [5], [6], [7], and optimization theory has been used to guide the design of new congestion control protocols (*e.g.*, [8]). In addition, traffic engineering imposes an optimization problem on the system [2], and optimization theory has been used to analyze proposed traffic-engineering protocols [9]. Although much of the existing research has focused on a single aspect of traffic management, our paper provides a holistic view.

*Optimization decomposition* is the process of decomposing a single optimization problem into many sub-problems, each of which is solved locally. The challenges of using optimization decomposition to derive protocols are two-fold. First, any mathematical modeling makes simplifying assumptions. Second, while multiple decomposition methods exist, it is unclear how to compare them. To the best of our knowledge, this is the first work that compares *multiple* decomposition solutions for traffic-management protocols, then builds a practical protocol that combines best features from each one.

In our *top-down redesign* of traffic management, we start by selecting an intuitive and practical objective function in Section III. Section II describes practical considerations in deploying of a distributed *multipath* traffic-management algorithm: where sources adapt their sending rates along multiple paths according to congestion feedback from the links. Although multipath traffic-management is not commonly deployed today, we explore several deployment scenarios that relate mathematical notions of sources and paths concretely to network elements. Using optimization decomposition techniques discussed in [10], Section IV derives four specific distributed solutions that differ in the computation of path rates and congestion feedback. Optimization theory guarantees that these algorithms converge to a stable and optimal point, while simulations allow us to compare rate of convergence and robustness to tunable parameters in Section V. Although these distributed algorithms work well, they can be sensitive to tun-

able parameters. We combine the best features of each of these algorithms to construct a simple TRAFFIC-management Using Multipath Protocol (TRUMP) in Section VI. Our contributions are two-fold:

- **Protocol Design using Decompositions:** We demonstrate how to create a practical network protocol by deriving multiple distributed algorithms, comparing their practical properties, and synthesizing their best features into a practical protocol.
- **Redesigned Traffic Management:** We introduce TRUMP: an easy to manage distributed protocol, that performs well for diverse topologies, capacities, feedback delays and traffic loads.

TRUMP converges faster than the four algorithms presented in Section IV, and has the fewest tunable parameters. Although TRUMP is not derived from a particular optimization decomposition, we are able to prove its convergence when the network is tuned to have low packet loss. As with any mathematical modeling, the TRUMP algorithm leaves many protocol details unspecified. We use engineering intuition to address these details.

In Section VII, the TRUMP protocol is evaluated using packet-level simulations with a wide range of topologies and traffic loads. We use a simple heuristic to set TRUMP's parameter such that it converges smoothly for a wide range of topologies, capacities, feedback delays, and traffic loads. When many flows share the same bottleneck link, there is a small amount of packet loss during convergence, but TRUMP still converges within a few RTTs. In contrast, the other distributed solutions become unstable when such links exist. We also study the impact of number of paths on the performance of TRUMP. First, TRUMP is more likely to split traffic over multiple paths when there are fewer concurrent source-destination pairs. Second, while TRUMP can achieve higher throughput if all sources can access two paths rather than a single path, additional paths do not provide significant gains. This paper discusses related work in Section VIII and concludes in Section IX.

## II. DISTRIBUTED MULTIPATH TRAFFIC MANAGEMENT

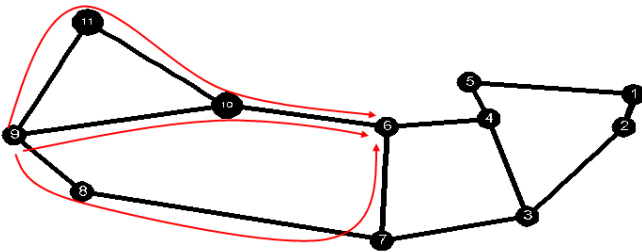


Fig. 1. Three paths between source node 9 and destination node 6.

Traffic management controls how much traffic traverses each path in a network. In the Internet today, end hosts run congestion control to adapt sending rates, and routing protocols select a *single path* between two end hosts. In this paper, we present several *distributed* traffic-management

algorithms where sources adapt sending rates on *multiple paths* to a destination. This is illustrated in Figure 1, source node 9 computes its sending rate on each of its three paths to destination node 6, based on feedback regarding the path congestion conditions. An advantage of our distributed algorithms is that they adapt at a single timescale (on the order of RTTs), and is able to respond quickly to traffic shifts.

Implementing such a distributed algorithm requires support from the network: multiple paths must exist between a source-destination pair, the sources must have knowledge of and control over the multiple paths, and data plane forwarding of packets on each of the specified paths must be possible. We address all these requirements in this section. The 'sources' and 'paths' in an algorithm can map to different network elements depending on the scenario. For example, in today's traffic-management system, sources can be end hosts (which run distributed congestion-control algorithms), or edge routers (which have been proposed to run distributed traffic-engineering protocols [11]).

**Existence of Multiple Paths:** In the Internet today, multiple end-to-end paths often exist because many stub networks are connected to multiple upstream ISPs, most ISPs have multiple paths between a pair of edge routers, and large ISPs often connect to each other in multiple locations. Today, most routing protocols only forward packets on a single path between a source and destination, though the underlying resources can be more efficiently utilized if traffic is *dynamically* balanced between multiple paths, as we propose in this paper.

**Sources See Multiple Paths:** Although path diversity exists in the Internet, sources cannot always access the multiple paths, since directing packets onto *any* end-to-end path can require cooperation between multiple networks. Still, there are several natural scenarios where cooperation between multiple networks is not required.

- **End-host Overlay:** In an end host overlay network, end hosts (possibly belonging to multiple networks) are connected in a logical topology. Although the routers between any two end hosts still select a single path between them, an end host can reach a destination through any other overlay node in the network, thus creating multiple paths.
- **Single ISP:** A single ISP can also exploit its own internal path diversity. Inside an ISP, an edge router can compute the traffic rates on each path and the end hosts connected to it can send at a rate explicitly specified by the edge router. Alternatively, an edge router can shape the incoming traffic, and the end hosts can run congestion control to adapt to the rate limits imposed by the edge routers.
- **Multihomed Stub:** A stub network connected to multiple upstream providers can split traffic over multiple upstream links. At the multihomed stub network, an edge router can compute the splitting percentages over outgoing links, or the stub network can provide the access information for the end hosts residing in the network.

Even more end-to-end paths are accessible if two or more networks cooperate, as surveyed in [12].

**Directing Packets onto Specific Paths:** Once sources have computed the splitting percentages between multiple paths, the

data plane must ensure packets are split between the specified paths accordingly. A packet can be directed onto a specific path through *tunnels*. There are two prevalent tunneling techniques in use today: IP-in-IP tunnels and MultiProtocol Label Switching (MPLS). In both cases, establishing a tunnel involves “pushing” an extra IP header (or label) at the tunnel ingress and “popping” the IP header (or label) at the tunnel egress. In the case of MPLS, each intermediate router also stores a label-based forwarding table, so that it can direct a packet to an appropriate outgoing link based on the label. No tunneling is required in the case of a multihomed stub, where the stub network just chooses an outgoing link, rather than the entire path a packet follows. Today, routers can already split traffic equally amongst multiple paths using a variety of techniques, thus it is not difficult to extend such techniques to achieve arbitrary splitting percentages [12].

### III. CHOOSING AN OBJECTIVE FUNCTION

In this section, we use optimization as a modeling language to formalize traffic management. Every optimization problem consists of an objective function, a constraint set and variables. For traffic management, by having both routing and source rate as optimization variables, we have the most flexibility in resource allocation. In our problem, the constraint is that link load does not exceed capacity. The objective function remains to be designed. We first propose an objective that maximizes aggregate user utility, but simulations reveal the solution converges slowly and is sensitive to stepsize. In addition, maximizing user utility leads to bottlenecks in the network, making the network fragile to traffic bursts. To address these practical challenges, we design an objective which balances maximizing user utility with minimizing operator’s cost function.

#### A. Maximizing Aggregate Utility: DUMP

One natural objective for the traffic management system is to maximize aggregate user utility, where utility  $U_i(x_i)$  is a measure of “happiness” of source-destination pair  $i$  (referred to as source  $i$  in this paper) as a function of the total transmission rate  $x_i$ .  $U$  is a concave, non-negative, increasing and twice-differentiable function, *e.g.*  $\log(x_i)$ , that can also represent the elasticity of the traffic or determine fairness of resource allocation. This is the objective implicitly achieved by TCP congestion control today [5], [6]. We represent the routing by matrix  $R_{li}$  that captures the fraction of source  $i$ ’s flow that traverses link  $l$ , and we let  $c_l$  denote the capacity of link  $l$ . As proposed in [13], [14], the resulting optimization problem is:

$$\begin{aligned} & \text{maximize} && \sum_i U_i(x_i) \\ & \text{subject to} && \mathbf{R}\mathbf{x} \preceq \mathbf{c}, \mathbf{x} \succeq \mathbf{0} \end{aligned} \quad (1)$$

where both  $R$  and  $x$  are variables.

A distributed solution to (1) can be derived through dual decomposition if (1) is a convex optimization problem. In its current form, (1) has a non-convex constraint set, which can be transformed into a convex set if the routing is allowed to be multipath. To capture multipath routing, we introduce  $z_j^i$  to

represent the sending rate of source  $i$  on its  $j$ th path. We also represent available paths by a matrix  $\mathbf{H}$  where

$$H_{lj}^i = \begin{cases} 1, & \text{if path } j \text{ of source } i \text{ uses link } l \\ 0, & \text{otherwise.} \end{cases}$$

$\mathbf{H}$  does not necessarily present all possible paths in the physical topology, but a subset of paths chosen by operators or the routing protocol. Then we can rewrite (1) as:

$$\begin{aligned} & \text{maximize} && \sum_i U_i(\sum_j z_j^i) \\ & \text{subject to} && \sum_i \sum_j H_{lj}^i z_j^i \leq c_l, \forall l. \end{aligned} \quad (2)$$

In this form, (2) is a convex optimization problem. A distributed solution to (2) can be derived using *dual decomposition* [13], where a *dual variable* is introduced to relax the capacity constraint. The resulting Dual-based Utility Maximizing Protocol (DUMP) is summarized in Figure 2. Similar to the reverse engineering of the congestion-control protocol in [6],  $s$  can be interpreted as link prices.

---

#### Feedback price update at link $l$ :

$$s_l(t+1) = \left[ s_l(t) - \beta_s(t) \left( c_l(t) - \sum_i \sum_j H_{lj}^i z_j^i(t) \right) \right]^+,$$

where  $\beta_s$  is the feedback price stepsize.

#### Path rate update at source $i$ , path $j$ :

$$z_j^i(t+1) = \text{maximize}_{z_j^i} \left( U_i \left( \sum_j z_j^i \right) - z_j^i \sum_l s_l(t) H_{lj}^i \right)$$


---

Fig. 2. The DUMP algorithm.

Here  $t$  represents the iteration number and each iteration is at the same timescale as the longest Round Trip Time (RTT) of the network. At each link,  $s_l$  is updated based on the difference between the link load  $\sum_i \sum_j H_{lj}^i z_j^i$  and the link capacity. As indicated by  $[\ ]^+$ ,  $s_l$  is only positive when the link load exceeds the link capacity, *i.e.* when the network is congested. Each source updates  $z_j^i$  based on explicit feedback from the links, in the form of feedback prices  $s_l$ . In particular, each source maximizes its own utility, while balancing the price of using path  $j$ . The path price is the product of the source rate with the price per load for path  $j$  (computed by summing  $s_l$  over the links in the path). DUMP is similar to the TCP dual algorithm in [6] except the local maximization is conducted over a *vector*  $\mathbf{z}^i$ , as opposed to only a scalar  $x_i$ , to capture the multipath nature of DUMP.

From optimization theory, certain choices of stepsizes, such as  $\beta_s(t) = \beta/t$  where  $\beta > 0$  is a constant, guarantee that DUMP will converge to the joint optimum as  $t \rightarrow \infty$  [15]. However, such diminishing stepsize is difficult to implement in practice as it requires synchronization of time across the nodes, and particularly difficult to do with dynamic arrivals of new flows. Previous work indicates that even under the simplest of topologies and assuming greedy flows, DUMP has poor convergence behavior [13]; our own Matlab experiments [16]

confirm this. When the stepsize is too large, DUMP will constantly overshoot or undershoot, never reaching the ideal utility. On the other hand, when the stepsize is too small, DUMP converges very slowly. Even at the optimal stepsize, DUMP only converges after about 100 iterations. This highlights that choosing an appropriate stepsize for DUMP is challenging.

### B. New Objective for Traffic Management

Let us reflect for a moment on why DUMP has poor convergence behavior. If we look at the form for feedback price, we see it is only nonzero when links are overloaded, therefore, the feedback from the links is not fine-grained. This corresponds to the congestion control mechanism of TCP Reno where sources only reduce their sending rates once packets are already lost, causing the sawtooth behavior. In fact, the feedback price in DUMP has the same formulation as the congestion price in [6]. In addition, utility is only based on throughput, while having low delay is also important to traffic management. In addition, the authors of [3] suggest the network is driven to a solution where some links are operating near capacity when only utility is maximized. This is an undesirable operating point which is very fragile to traffic bursts. This indicates that maximizing the aggregate utility enhances performance of the individual users, but leaves the network as a whole fragile.

To avoid the poor convergence properties of DUMP, we look for an alternative problem formulation which also takes into account the operator's objective. Today, traffic engineering solves the following optimization problem with only  $\mathbf{R}$  as a variable (and  $\mathbf{x}$  constant):

$$\text{minimize } \sum_l f(\sum_i R_{li}x_i/c_l). \quad (3)$$

$f$  is a convex, non-decreasing, and twice-differentiable function that gives increasingly heavier penalty as link load increases, e.g.  $e^{\sum_i R_{li}x_i/c_l}$ . The intuition behind choosing this  $f$  is two-fold. First,  $f$  can be selected to model M/M/1 queuing delay. Second, network operators want to penalize solutions with many links at or near capacity and do not care too much whether a link is 20% loaded or 40% loaded [2]. If we solve (3) with both  $\mathbf{x}$  and  $\mathbf{R}$  as variables, then the solution would end up with zero throughput, which is also undesirable.

A better traffic management objective could be to combine performance metrics (users' objective) with network robustness (operator's objective), leading to the following formulation as a joint optimization over  $(\mathbf{x}, \mathbf{R})$ :

$$\begin{aligned} &\text{maximize } \sum_i U_i(x_i) - w \sum_l f(\sum_i R_{li}x_i/c_l) \\ &\text{subject to } \mathbf{R}\mathbf{x} \preceq \mathbf{c}, \mathbf{x} \succeq \mathbf{0}. \end{aligned} \quad (4)$$

This objective favors a solution that strikes a trade-off between high aggregate utility and a low overall network congestion, to satisfy the need for performance and robustness. Similar problem formulations were proposed in [3], [17], though without  $w$ . Here  $w$  is a parameter which adjusts the balance between the utility function and the cost function. When  $w$  is small, (4) is very close to (1) since the utility term dominates. When

$w$  is large, the solution is more conservative and avoids high link utilization. Today, operators perform traffic engineering by adjusting link weights depending on the instantaneous traffic load. In our case, they can adjust a single parameter  $w$ .

Aside from performance, fairness is another important consideration. From a theoretical perspective, the solution to (4) is  $\alpha$ -fair as  $w \rightarrow 0$ , where  $\alpha$ -fairness is defined in [18]. While this does not hold for general values of  $w$ , our experimental results in Section VII-F are encouraging.

Before generating distributed solutions in Section IV, we first transform (4) to a convex optimization problem:

$$\begin{aligned} &\text{maximize } \sum_i U_i(\sum_j z_j^i) - w \sum_l f(y_l/c_l) \\ &\text{subject to } \mathbf{y} \preceq \mathbf{c}, \\ & \quad y_l = \sum_i \sum_j H_{lj}^i z_j^i, \forall l. \end{aligned} \quad (5)$$

Note that to decouple the objective which contains  $U$  (a per-source function) and  $f$  (a per-link function), we introduce an extra variable  $y_l$  to provide feedback before link load exceeds the actual capacity.

## IV. MULTIPLE DECOMPOSITIONS

In this section, we describe the distributed algorithms generated from optimization decompositions of (4) (the decomposition techniques are surveyed in [10], [5]). All four resulting algorithms update the path rates based on feedback prices from links. There are a number of other similarities between the four algorithms. First, the operations performed by links including measuring the link load present only a small overhead. Second, all four algorithms incur the same small message passing overhead: only the sum of the link prices on the end-to-end path needs to be communicated. Third, while computations can involve solving a local optimization problem and taking derivatives,  $U$  and  $f$  are twice differentiable, and therefore closed-form solutions exist and they are just simple function evaluations. Finally, the computational complexity of all four algorithms is constant per link and linear per source. The main difference, then, is the number of tunable parameters of each algorithm, which varies from one to three. Optimization decomposition leads us to three constructs that are generally applicable: effective capacity, consistency price and direct path-rate update.

### A. Effective Capacity

The first three algorithms (partial-dual, primal-dual, and full-dual) prevent link loads from reaching link capacity by providing feedback based on *effective capacity* rather than actual capacity. In the resulting algorithms, the sources update the path rates based on feedback price just as in Figure 2. The feedback price is similar to that in Figure 2, except it is based on effective capacity  $y_l$ :

$$s_l(t+1) = s_l(t) - \beta_s \left( y_l(t) - \sum_i \sum_j H_{lj}^i z_j^i(t) \right). \quad (6)$$

As in Section III-A, we consider constant stepsize for practical reasons, thus we remove the dependence on  $t$  from all stepsizes.

1) *Local Optimization: Partial-Dual*: The derivation process for the **partial-dual** algorithm is identical to Section III-A except with *effective capacity*  $\mathbf{y}$  as an additional primal variable. The constraint  $\mathbf{y} \preceq \mathbf{c}$  is enforced, resulting in the following equation for updating effective capacity:

$$y_l(t+1) = \underset{(y_l \leq c_l)}{\text{minimize}} w f(y_l/c_l) - s_l(t)y_l. \quad (7)$$

In (7),  $y_l$  is updated by solving a local optimization using information from the feedback price and the cost function  $f$ . An economic interpretation is that the effective capacity balances the cost of using a link (represented by  $f$ ) and revenue from traffic transmission (represented by the product of feedback price with the effective capacity). There is an explicit solution to (7). Note that the effect of the cost function is proportional to  $w$ .

2) *Subgradient Update: Primal-Dual*: The **primal-dual** decomposition first decomposes (5) into two subproblems, one responsible for each primal variable. The master problem solves for  $\mathbf{y}$  assuming a given  $\mathbf{x}^*$ , while the subproblem solves for  $\mathbf{x}$  assuming a fixed  $\mathbf{y}$ . The master problem is as follows:

$$\begin{aligned} & \text{maximize} && \sum_i U_i(\mathbf{x}^*) - w \sum_l f(y_l/c_l) \\ & \text{subject to} && \mathbf{y} \preceq \mathbf{c}, \end{aligned} \quad (8)$$

where  $\mathbf{x}^*$  is a solution to the following subproblem:

$$\begin{aligned} & \text{maximize} && \sum_i U_i(x_i) \\ & \text{subject to} && \mathbf{R}\mathbf{x} \preceq \mathbf{y}. \end{aligned} \quad (9)$$

Note that (9) is identical to (2) except the constraint is on  $\mathbf{y}$  rather than  $\mathbf{c}$ . The solution to the subproblem is then identical to that presented in Figure 2 except for the feedback price update which uses the effective capacity  $\mathbf{y}$  rather than actual capacity  $\mathbf{c}$ .

The master problem can be solved through an iterative update of effective capacity :

$$y_l(t+k) = \min(c_l, y_l(t) + \beta_y(s_l(t) - w f'(y_l(t))))), \quad (10)$$

where  $\beta_y$  is the effective capacity stepsize. Taking a closer look at (10), the minimization ensures effective capacity stays below the actual capacity. The parameter  $k$  is an integer greater than 1 since (8) is updated less frequently than (9). The subgradient update itself consists of balancing the price the link can charge ( $s_l$ ), and the cost that link must pay ( $f'(y_l)$ ). In a nutshell, the primal-dual decomposition is identical to the partial-dual decomposition except that the effective capacity is updated iteratively through (10) rather than by solving a local minimization problem.

### B. Consistency Price: Full Dual

The **full-dual** decomposition is quite similar to the partial-dual decomposition in Section IV-A1, but a second dual variable  $\mathbf{p}$  is introduced to relax the constraint  $\mathbf{y} \preceq \mathbf{c}$ . This dual variable can be interpreted as *consistency price* as it ensures *consistency* between the effective capacity and the capacity constraint at the equilibrium point. As with the feedback price,

the consistency price is updated over time using a subgradient method:

$$p_l(t+1) = [p_l(t) - \beta_p(c_l - y_l(t))]^+,$$

where  $\beta_p$  is the stepsize for consistency price. Consistency price only comes into play when the capacity constraint is violated, therefore, it is mapped to a non-negative value. The effective capacity update is based on both link prices:

$$y_l(t+1) = \underset{y_l}{\text{minimize}} w f(y_l/c_l) - (s_l(t) + p_l(t))y_l.$$

The path rate update and feedback price update are identical to that of the previous two algorithms. The full-dual algorithm closely resembles an algorithm presented in [3], though our objective contains  $w$  as a weighing factor. Appendix 2 of [3] also shows a complete derivation of the full-dual algorithm.

### C. Direct Path Rate Update: Primal

In all the previous algorithms, auxiliary dual variables were introduced to relax the constraints. In this **primal** decomposition, we find a direct solution by introducing a penalty function, as in the appendix of [19]. Let the penalty function  $g_l(\sum_i \sum_j H_{lj}^i z_j^i)$  replace the capacity constraint  $\mathbf{H}\mathbf{z} \preceq \mathbf{c}$ . The penalty function is a continuous, increasing, differentiable and convex function that is sufficiently steep such that link loads will not overshoot capacity. If it is also sufficiently close to zero for values less than capacity, it will not affect the optimal point [20]. If we add  $g$  and the cost function  $f$  to get a penalty-cost function  $P_l(\sum_i \sum_j H_{lj}^i z_j^i)$ , then (5) can be transformed into the following:

$$\text{maximize} \sum_i U_i(\sum_j z_j^i) - w \sum_l P_l(\sum_i \sum_j H_{lj}^i z_j^i). \quad (11)$$

The derivative of (11) is:

$$\frac{dz_i}{dt} = \beta_z \frac{\partial U_i}{\partial z_j^i}(x_i(t)) - w \sum_l P_l'(\sum_i \sum_j H_{lj}^i z_j^i(t)), \quad (12)$$

where  $\beta_z$  is the stepsize for path rate. Converting (12) into a subgradient update form and separating link information from source information, we obtain the algorithm in Figure 3.

---

#### Path rate update:

$$z_j^i(t+1) = z_j^i(t) + \beta_z z_j^i(t) \left( \frac{\partial U_i}{\partial z_j^i}(x_i(t)) - \sum_l H_{lj}^i s_l(t) \right)$$

#### Feedback price update:

$$s_l(t+1) = w P_l'(\sum_i \sum_j H_{lj}^i z_j^i(t))$$


---

Fig. 3. The Primal algorithm.

The path rates are iteratively updated based on the difference between the rate of change of the utility function and the associated path feedback price. The feedback price here directly

represents how quickly the penalty function is changing at a given link load. The primal algorithm in Figure 3 differs significantly from the first three decompositions. First, it uses direct subgradient update on the path rates. Second, it does not use the concept of effective capacity.

## V. CONVERGENCE PROPERTIES

In this section, we study convergence properties of the four algorithms, and make key observations which will guide our design of a new protocol in Section VI. First, we find that there is a trade-off between the speed of convergence and the achievable aggregate utility. Second, we find algorithms which use local minimizations instead of iterative updates converge faster. Third, we find consistency price can aid convergence for small  $w$ .

### A. Set-up of MATLAB Experiments

Due to the multitude of tuning parameters, finding the optimal values requires fine-grained sweeping of the parameter space. Thus we use MATLAB simulations along with simple topologies and simple traffic patterns to identify the key properties that improve convergence. For all algorithms, we update the source and link variables at each iteration based on link load from the previous iteration. For the utility function  $U$ , we use a logarithmic function commonly associated with proportional fairness and TCP Reno today [18]. For the cost-function  $f$ , we use an exponential function, which is the continuous version of the function used in various studies of traffic engineering [2].

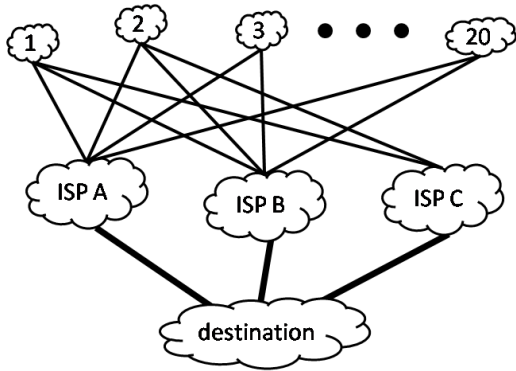


Fig. 5. Topology capturing 20 stub networks connected to 3 ISPs. All networks are connected to ISP A, networks 1 through 10 are connected to ISP B and networks 1, 2, 3, 11, 12, 13 are connected to ISP C. The links connecting the ISPs to the destination have feedback delay from 30ms to 80ms.

We study three realistic topologies as shown in Figures 4 and 5. Figure 4a is a tree-mesh topology, which is representative of a common access-core network structure. Figure 4b is the Abilene backbone network [21]. Finally, Figure 5 represents a multihoming topology where many multihomed stubs are all trying to reach the same destination through three ISPs. We select six source-destination pairs for access core and four pairs for Abilene. For each of these communicating pairs, three minimum-hop paths are available for access-core and four

minimum-hop paths are available for Abilene. The simulations assume the link capacities follow a truncated (to avoid negative values) Gaussian distribution, with an average of 100 and a standard deviation of 10. For this set of experiments, we define convergence as reaching 99.9% of the optimal aggregate utility of (4). We found the convergence rates to be independent of initial rate assignments. We omit extra graphs when the same trends are observed across algorithms, topologies and values of  $w$ , more detailed results can be found in [16].

### B. Weighing User Utility and Operator Cost

In this subsection, we study the *trade-off between aggregate utility and convergence time*. In Figure 6, we plot the number of iterations before convergence against stepsize  $\beta_s$  for three values of  $w$  for the partial-dual algorithm. For each stepsize, each point corresponds to a set of capacity values, and the average number of iterations before convergence is highlighted in a solid line. Comparing across Figure 6 from left to right, we see that as  $w$  decreases, the convergence time at the optimal stepsize increases and the range of stepsizes with a good convergence time shrinks.

In Figure 7, we plot the aggregate utility achieved by solving (4) as a percentage of maximal aggregate utility achieved by solving (1), for a range of  $w$  values. From the graph, we observe that there is a knee region for all three topologies. For the Abilene topology, this knee region is  $w = [1/6, 1/10]$ ; for the access-core topology, this knee region is  $w = [1/4, 1/6]$ ; for the multihoming topology, the knee region is  $w = [5/4, 3/2]$ . For  $w$  values smaller than the knee region, the algorithm achieves near maximal aggregate utility, since the cost function  $f$  is weighed sufficiently lightly to not change achieved aggregate utility. For  $w$  values larger than the knee region, the aggregate utility achieved decreases, as the cost function  $f$  becomes a significant part of the objective. The location of the knee region depends on whether bottleneck links are shared by many source-destination pairs, which is dependent on the topology, associated capacities, and the source-destination pairs chosen.

In this paper, we define a *flow* to be the aggregate connections between a source-destination pair. In the multihoming topology, the three links connecting the ISPs to the destination are shared by many flows. If all links have equal capacities, then those three links are bottleneck links. Since the  $f$  function is a sum over all links, when a bottleneck link is shared by many flows, the penalty associated with pushing that link to full capacity is compensated by driving all sending rates higher. Consequently, for a given  $w$  value, the gap to maximal achievable utility is smaller when there is a single bottleneck link shared by multiple flows, than when there are many bottleneck links. So, in Figure 7, we observe the aggregate utility is at 100% even at  $w = 1$  for the multihoming topology. Similarly, the access-core topology has a knee at a larger  $w$  than the Abilene topology, since several flows share the mesh at the center in the access-core topology.

Looking at Figures 6 and 7 together, at  $w = 1/36$ , the maximal aggregate utility is achieved, but the partial-dual algorithm converges slowly and is very sensitive to stepsize

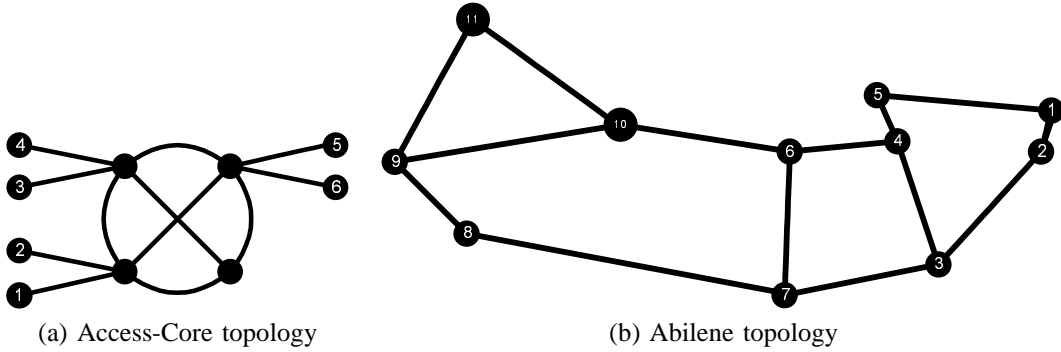


Fig. 4. Two topologies.

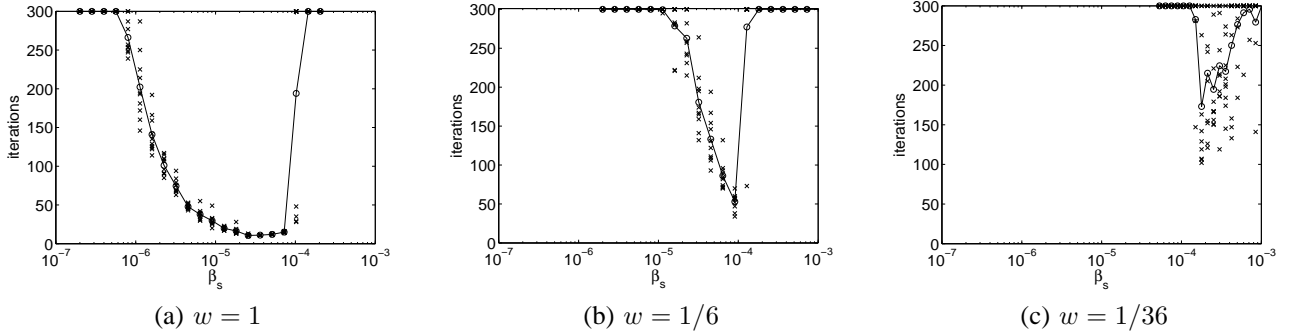
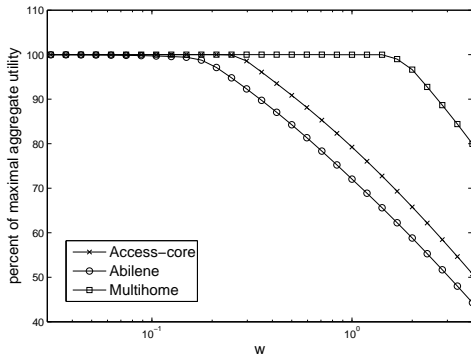


Fig. 6. Plots of partial-dual algorithm showing dependence of convergence time on stepsize. 'x' represent the actual data points and 'o' represent the average value. Access-core topology was used.

Fig. 7. Plot of  $w$  versus percentage of maximal utility achieved.

choice. In comparison, for a mere 3% reduction in aggregate utility, the convergence properties improve significantly at  $w = 1/6$ . As  $w$  increases even more, there is a clear trade-off between aggregate utility achieved, the rate of convergence and sensitivity to stepsize. At  $w = 1$ , the convergence properties are much nicer than at  $w = 1/6$ , but there is a 20% drop in utility. For a given  $w$  value, if the topology, capacities and traffic pattern causes multiple flows to share a single bottleneck link (e.g., multihoming topology instead of access-core topology), then there is a higher utility achieved, but a slower rate of convergence.

### C. Comparing the Algorithms

In this subsection, we do a series of comparisons between convergence time and stepsize sensitivity of the four algorithms, and find partial-dual in Figure 6 is the best overall, with a good convergence profile and fewest tunable parameters. We summarize our observations in Table I.

Comparing the primal-dual algorithm to the partial-dual algorithm, we find *the two extra tunable parameters do not improve the convergence properties*. The convergence times of primal-dual and partial-dual algorithms are almost identical for well-chosen  $\beta_y$  and  $k$ . For other values of  $\beta_y$ , however, we find the primal-dual algorithm converges more slowly than the partial-dual algorithm.

Comparing the full-dual algorithm in Section IV-B to the partial-dual algorithm, we find *consistency price may improve convergence properties*. From Table I, we note that  $\beta_p$  has no effect on the convergence time when  $w = 1$ . This is because the effective capacity stays far below actual capacity when  $w$  is high, so consistency price  $p_l$  stays at 0 and its stepsize plays no role. For  $w = 1/6$  (which is the edge of the knee region seen in Figure 7), we find that the full-dual algorithm can converge faster than the partial-dual algorithm. This is because if we allow the capacity constraint to be violated during transient periods, the algorithm can take more aggressive steps and potentially converge faster.

Comparing the primal algorithm in Section IV-C to the partial-dual algorithm, we find *local minimization update has better convergence properties than subgradient update*. This is intuitive as the subgradient update with a constant stepsize

Algorithm	Partial-Dual	Primal-Dual	Full-Dual	Primal
$w = 1$ , Access-Core	15	25*	15	25
$w = 1/6$ , Access-Core	50*	75**	125*	150*
$w = 1$ , Abilene	15	25*	15	25
$w = 1/6$ , Abilene	125*	100**	50*	150*

TABLE I

SUMMARY OF AVERAGE NUMBER OF ITERATIONS TO CONVERGENCE FOR BEST CHOSEN TUNING PARAMETERS. HERE \* DENOTES SENSITIVITY TO STEPSIZE VARIATION AND \*\* DENOTES EXTRA SENSITIVITY TO STEPSIZE VARIATION.

is constrained to react with the same strength each time, while local minimization can react more flexibly. From Table I, the primal algorithm takes longer to converge at the optimal stepsize (25 iterations versus 15 iterations). In addition, the primal algorithm also requires operators to tune a second parameter  $g$ .

## VI. TRUMP

While the algorithms introduced in Section IV converge faster than DUMP, we seek an algorithm with even better convergence properties. In this section, we introduce Traffic-management Using Multipath Protocol (TRUMP) with only one easy to tune parameter.

### A. The TRUMP Algorithm

Our simulations in the previous section suggest that simpler algorithms with fewer tunable parameters converge faster, although having a second link price can help for small  $w$ . Using those observations, we *combine the best parts of all four algorithms* to construct the TRUMP algorithm described in Figure 8.

In TRUMP, the feedback price has two components as in the *full-dual* algorithm:  $p_l$  and  $q_l$ . Since we observed that local optimization worked better than subgradient update, we use the feedback price update from *primal* algorithm in Figure 3 as our  $q_l$ . This has the additional benefit of removing one tuning parameter from the protocol since the update of  $q_l$  involves no stepsize. By a similar argument, we use a local optimization for the path rate update as in the dual-based algorithms. The value of  $w$  is only known at the sources where the  $z$ 's are computed, and there is only a single value for the network. The packet-level simulations in Section VII-C reveal that TRUMP performs well for a large range of  $w$ -values when an appropriate stepsize is chosen.

Through simulations, we find that TRUMP indeed converges to the optimum of (4) for both topologies and a range of  $w$  values. When we plot the achieved aggregate utility at equilibrium versus  $w$ , we obtain a plot identical to Figure 7. In Figure 9, we plot convergence time versus stepsize for TRUMP. When the network sources are reacting strongly to the price  $q$  (e.g.,  $w = 1$  and the traffic engineering part is dominating), the price  $p$  is unnecessary as observed in Figure 9a. In the region where the network is being less conservative ( $w = 1/6$ ), price  $p$  is a more definitive indicator of performance than price  $q$ , and can be helpful for source rate adjustments. Comparing Figure 9 to Figure 6, we see that TRUMP has nicer convergence properties than the partial-dual algorithm, while having fewer parameters.

### Feedback price update:

$$s_l(t+1) = p_l(t+1) + q_l(t+1),$$

$$p_l(t+1) = [p_l(t) - \beta_p(c_l - \sum_i \sum_j H_{lj}^i z_j^i(t))]^+,$$

$$q_l(t+1) = w f' \left( \sum_i \sum_j H_{lj}^i z_j^i(t) / c_l \right),$$

### Path-rate update:

$$z_j^i(t+1) = \text{maximize}_{z_j^i} U_i \left( \sum_j z_j^i \right) - \sum_l s_l(t) \sum_j H_{lj}^i z_j^i$$

Fig. 8. The TRUMP algorithm.

### B. TRUMP Convergence Proof

Unlike the algorithms from Section IV, TRUMP is a heuristic and does not correspond to a known decomposition. Consequently, the convergence and optimality is not automatically guaranteed by optimization theory. Theorem 1 below guarantees convergence of TRUMP when the network is lightly loaded. We consider the region where  $w$  is sufficiently large for  $p = 0$  (as seen in Figure 9a), and find a contraction mapping on  $z$ . Overall, TRUMP is simpler than any of the algorithms presented in Section IV, with only one tunable parameter that only needs to be tuned for small  $w$ .

A particular family of widely-used utility functions is parameterized by  $\alpha \geq 0$  [18]:

$$U_\alpha(x) = \begin{cases} \log x, & \alpha = 1 \\ (1 - \alpha)^{-1} x^{1-\alpha}, & \alpha \neq 1. \end{cases} \quad (13)$$

Maximizing these  $\alpha$ -fair utilities over linear flow constraints leads to rate-allocation vectors that satisfy the definitions of  $\alpha$ -fairness in the economics literature. The notion of  $\alpha$ -fairness from [18] led to many TCP variants with different  $\alpha$ -fairness interpretations. A utility function with  $\alpha = 2$  was linked to TCP Reno. Through reverse engineering, TCP Vegas can be interpreted as  $\alpha = 1$ , as can STCP and FAST. XCP is shown to be maximizing  $U_\alpha$  as  $\alpha \rightarrow \infty$  in the single-link case.

*Theorem 1:* TRUMP converges to the optimal value of (4) under the following conditions:

- 1)  $p_l = 0, \forall l$
- 2)  $n_l < \alpha \frac{f_l'(u_l)^{(1/\alpha+1)}}{f_l'(u_l)}, \forall l$

where  $n_l$  is the number of flows sharing link  $l$  and  $\alpha$  refers to  $\alpha$ -fair utility [18].

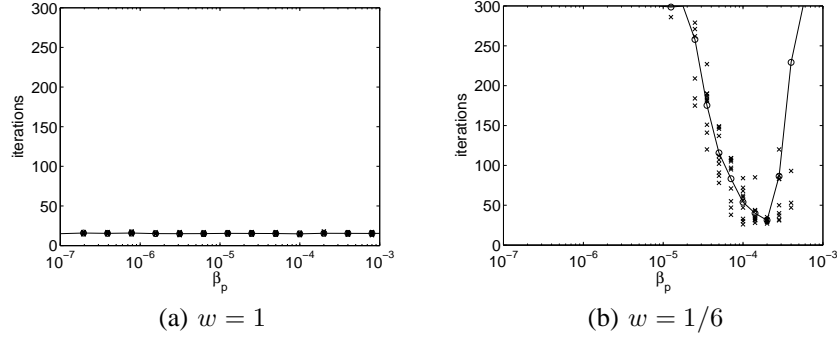


Fig. 9. Plots of TRUMP algorithm showing dependence of convergence time on stepsize. ‘x’ represent the actual data points and ‘o’ represent the average value. Access-core topology.

*Proof:* If  $p = 0$ , then the  $z$  update is:

$$z_j^i = U_i^{t-1} \left( \sum_l H_{lj}^i f_l' \left( \sum_{i,j} z_j^i H_{lj}^i / c_l \right) / c_l \right).$$

We look for a *contraction mapping* for  $z$  as outlined in [20]. First we compute the Jacobian for  $z_j^i$ :

$$J_{ij,st} = \frac{(U_i^{t-1})' \left( \sum_l H_{lj}^i f_l' \left( \sum_{i,j} z_j^i H_{lj}^i / c_l \right) / c_l \right)}{\left( \sum_l H_{lj}^i \sum_{s,t} H_{lt}^s f_l'' \left( \sum_{i,j} z_j^i H_{lj}^i / c_l \right) / (c_l)^2 \right)}.$$

Let  $u_l = \sum_{i,j} z_j^i H_{lj}^i / c_l$  be the link utilization, then:

$$\|J\|_\infty = \max_{ij} (U_i^{t-1})' \left( \sum_l H_{lj}^i f_l'(u_l) \right) \left( \sum_l H_{lj}^i \sum_{s,t} H_{lt}^s f_l''(u_l) \right).$$

Let  $n_l = \sum_{s,t} H_{lt}^s$  represent the number of flows sharing link  $l$ , then:

$$\|J\|_\infty = \max_{ij} (U_i^{t-1})' \left( \sum_l H_{lj}^i f_l'(u_l) \right) \left( \sum_l H_{lj}^i n_l f_l''(u_l) \right).$$

For convergence of  $z$ ,  $\|J\|_\infty < 1$  is a *sufficient* condition. For  $\alpha$ -utility, we have  $(U_i^{t-1})' = -\frac{1}{\alpha} x^{-1/\alpha-1}$  for  $\alpha > 1$ . For  $U = \log(x)$ ,  $\alpha = 1$ ,  $(U_i^{t-1})' = -x^{-2}$ , so the same equation holds. So we can rewrite  $\|J\|_\infty$  as:

$$\|J\|_\infty = \max_{ij} \frac{1}{\alpha} \frac{\sum_l H_{lj}^i n_l f_l''(u_l)}{\left( \sum_l H_{lj}^i f_l'(u_l) \right)^{(1/\alpha+1)}}.$$

$$\|J\|_\infty < 1 \text{ holds if } \frac{n_l}{\alpha} f_l''(u_l) < f_l'(u_l)^{(1/\alpha+1)}, \forall l. \quad \blacksquare$$

### C. TRUMP: Transition to Network Protocol

The transition from a mathematical algorithm to a network protocol requires relaxation of several simplifying assumptions. First, the algorithm in Figure 8 assumes feedback is signaled explicitly from links to sources. The explicit feedback could be piggy-backed on acknowledgment packets [22], attached to probe packets [11] or flooded throughout the network [23]. In all cases, there is delay associated with the feedback. Second, the algorithm assumes traffic flows fluidly, while real traffic consists of *packets*. Third, while an algorithm can be broadly defined with a family of functions  $U$  and  $f$ , a *specific*  $U$  and  $f$  must be selected. We address these concerns in the TRUMP protocol.

The time between iterations of the TRUMP algorithm depends on  $\text{RTT}_j^i$ , the time it takes for source  $i$  to receive feedback along all the links of path  $j$ . To transition to a packet-based protocol, the link prices are calculated based on the estimated local link load:  $N_T$  the number of bits which arrived in period  $(t, t+T)$  divided by length of the period. Choosing  $f$  as an exponential function, each link updates its prices as:

$$p_l(t+T) = [p_l(t) - \beta_p(c_l - \frac{N_T}{T})]^+, \quad (14)$$

$$q_l(t+T) = \frac{w}{c_l} * \exp\left(\frac{N_T}{T c_l}\right), \quad (15)$$

$$s_l(t+T) = p_l(t+T) + q_l(t+T). \quad (16)$$

Choosing a logarithmic function for  $U$  and solving the local minimization, we obtain the following source rate update:

$$z_j^i(t+T_j^i) = z_j^i(t) - \gamma \sum_j z_j^i(t) + \frac{\gamma}{\sum_l H_{lj}^i s_l(t)}. \quad (17)$$

At time 0, the prices are initialized to a constant before real prices are available after one RTT. New flows after time 0 are set at the calculated path rates according to the latest (delayed) price, collected by a probe before the flow starts. To control the rate of convergence for flows with varying RTTs, as commonly done in congestion control mechanisms, *e.g.* [8], we introduce a parameter  $0 < \gamma < 1$ . In general, path rates are updated every  $\gamma \text{RTT}_j^i$ , but the path rate is recalculated at most once for any given price update. Thus the path rate adaptation will happen every  $T_j^i = \max(T, \gamma \text{RTT}_j^i)$ . Note that the extra parameters  $\gamma$  and  $T$  are necessary for any packet-level protocol.

## VII. TRUMP: PACKET-LEVEL EVALUATION

In our MATLAB simulations, we had made a number of simplifying assumptions. Moving to packet-level simulations, we study the impact of relaxing the following assumptions: homogeneous feedback delay, no flow dynamics and no packet-level burstiness. In addition, we test TRUMP under realistic traffic loads and link failures. Finally, we examine whether TRUMP shares bottleneck links fairly.

### A. Experimental Set-up

We implement the TRUMP protocol in NS-2 as described in Section VI-C. In particular, the link prices are updated every  $5ms$  and feedback is piggybacked from the links to the sources. The path rates are updated with  $\gamma = 0.1$ . Most of the experiments are performed with  $w = 1$ , where there is no packet loss. The calculated source rates are compared to the ideal rates, which are determined using MOSEK optimization software.

Our simulations use both synthetic and realistic topologies, which are summarized in Tables II and III respectively. For the topologies that were previously simulated in MATLAB (Figure 4), we use the same paths with link capacities of 100Mb/s. Link delays on the Abilene topology were selected to approximate the realistic values. Links in Access-Core topology have a one-way propagation delay of  $50ms$ , a value chosen to test TRUMP under long feedback delay. Figure 5 contains three heavily loaded links, and hence was chosen for tuning of  $\beta_p$  under varying capacities, with link delays varying from  $30ms$  to  $80ms$ . Specific paths and link delays are selected in the Share topology (Figure 15a) to test the fairness of TRUMP. Links in the Share topology have a capacity of 200Mbps, except for the bottleneck link from node 7 to node 8, which has a capacity of 100Mbps.

Topology	Nodes	Links	Flows	Paths
Abilene	11	28	4	4
Access Core	10	24	6	3
Multihome	24	40	20	1-3
Share	9	16	3	1

TABLE II  
SUMMARY OF SYNTHETIC TOPOLOGIES.

Since TRUMP with explicit feedback is most easily deployed inside a single AS, we obtained intra-AS topologies, along with link delays from the Rocketfuel topology mapping engine [24], [25]. The link capacities are 100Mbps if neither endpoint has degree larger than 7, and 52Mbps otherwise. As summarized in Table III, between 10 and 50 flows were randomly selected. For each source-destination pair, multiple paths were computed by first selecting a third transit node, then computing the shortest path containing all the three nodes, and finally removing cycles in the path. The RTTs on the paths range from  $1ms$  to  $400ms$ .

ISP(AS Number)	Cities	Links	Flows	Paths
Genuity(1)	42	110	50	1-4
Telstra(1221)	44	88	20	1-4
Sprint(1239)	52	168	500	1-4
Tiscali(3257)	41	174	25	1-4
Abovenet(6461)	19	68	10	1-4
AT&T(7018)	115	296	1000	1-4

TABLE III  
SUMMARY OF ISP TOPOLOGIES.

### B. Tuning Step size of TRUMP

We observed in section V-B that the links connecting the ISPs to the destination are fully utilized even for the conservative choice of  $w = 1$ . Previous MATLAB results indicate choosing  $\beta_p$  is challenging when there are bottleneck links, since packet loss can easily occur. In this section, we study the impact of link capacity on  $\beta_p$ , using the multihoming topology in Figure 5 where there are three bottleneck links.

	10 Mbps	100 Mbps	1000 Mbps
$\beta_p = 1 * 10^{-11}$	no	no	no
$\beta_p = 1 * 10^{-13}$	no	no	no
$\beta_p = 1 * 10^{-15}$	yes	no	no
$\beta_p = 1 * 10^{-17}$	slow	yes	no
$\beta_p = 1 * 10^{-19}$	slow	slow	yes
$\beta_p = 1 * 10^{-21}$	slow	slow	slow
$\beta_p = 1 * 10^{-23}$	slow	slow	slow

TABLE IV  
RATE OF CONVERGENCE OF TRUMP FOR DIFFERENT  $\beta_p$  VALUES AND DIFFERENT LINK CAPACITIES.

In our first set of experiments, we vary the capacity of the links *uniformly* from 10Mbps to 1000Mbps. In Table IV, we observe the best  $\beta_p$  for fast convergence is  $1 * 10^{-15}$  for 10Mbps,  $1 * 10^{-17}$  for 100Mbps and  $1 * 10^{-19}$  for 1000Mbps. More precisely, the appropriate  $\beta_p$  value decreases by two orders of magnitude when the link capacities increase by one order of magnitude. Taking a closer look at (14), we observe that for  $\beta_p = 0.1$ ,  $p_l$  is larger than  $q_l$  by  $c_l^2$ . Therefore, we let  $\beta_p$  equal  $0.05/c_l^2$  and repeat the experiments with different capacities. We find that convergence is achieved with this setting of  $\beta_p$ . To confirm our choice holds in networks with heterogeneous capacities, we repeated the experiment with topology from Figure 5 with randomly assigned capacities ranging from 10Mbps to 1000Mbps. We confirmed that  $\beta_p = 0.05/c_l^2$  results in a smooth convergence even in this challenging scenario.

### C. TRUMP versus Partial-Dual

We confirm our MATLAB results from Section VI-A: TRUMP has better convergence properties than partial-dual under heterogeneous feedback delay and for a range of  $w$  values. In Figures 10 and 11, we plot the aggregate throughput in the Sprint network with 50 greedy flows. The paths chosen had RTTs ranging from  $3ms$  to  $327ms$ , with an average of  $127ms$  and a standard deviation of  $76ms$ . Similar to the MATLAB experiments, we observe TRUMP converges slower for smaller  $w$ , though to higher aggregate rates, as shown in Figure 10. When  $w = 1$ , the TRUMP aggregate rates increase from 0 at time  $0s$  (when the flows are established), to close to the target value within  $500ms$  — about 4 times the average RTT. When  $w = 1/6$ , the TRUMP aggregate rates take longer to converge, though they still converge smoothly. Comparing Figure 10a with Figure 10b, for the first second or so, the actual throughput is lower than the sending rates for small  $w$ . This is because if TRUMP's sending rates are above the bandwidth in the network, packets are lost. TRUMP converges

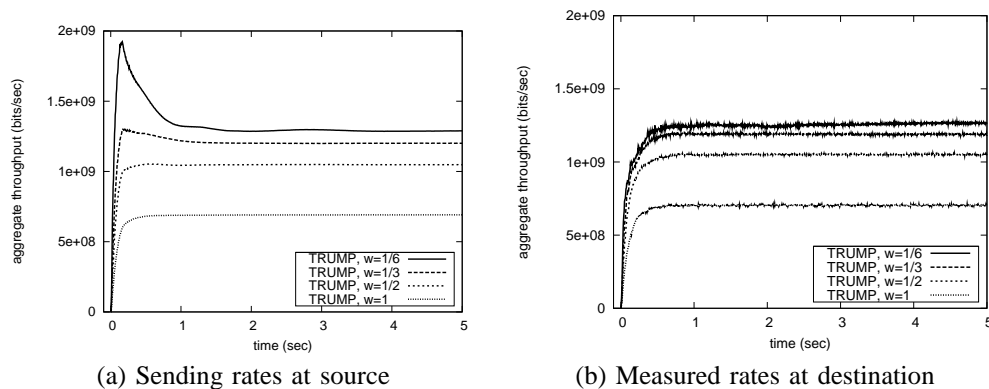


Fig. 10. Aggregate throughput of TRUMP with  $\beta_p = 0.05/c_l^2$ , in the Sprint network with 50 greedy flows.

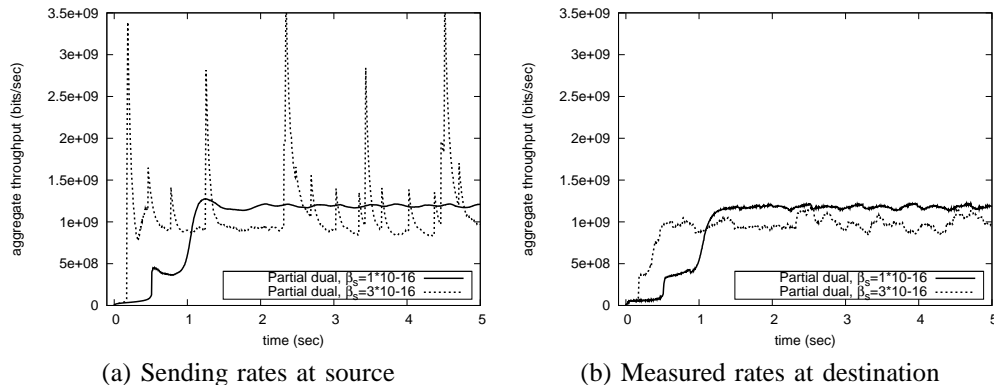


Fig. 11. Aggregate throughput of partial-dual with  $w = 1/3$ , in the Sprint network with 50 greedy flows.

for a range of  $w$  values with a single  $\beta_p$  value, chosen in the previous subsection.

Similar to the MATLAB experiments, we observe in Figure 11 the partial-dual is quite sensitive to the choice of stepsize at  $w = 1/3$ . In Figure 11a, we observe for a stepsize of  $10^{-16}$ , the partial-dual sending rates converge slowly; but for a stepsize of  $3 \times 10^{-16}$  (only three times larger), we find the partial-dual sending rates oscillate significantly. In Figure 11b, we observe when the sending rates oscillate, there are heavy packet losses. In fact, the actual throughput for a stepsize of  $3 \times 10^{-16}$  is lower than when the stepsize is  $10^{-16}$ . In addition, the same stepsize does not work across different values of  $w$ . By comparing Figures 10 and 11, we confirm our MATLAB results from Section VI-A: TRUMP has better convergence properties than partial-dual under heterogeneous feedback delay and for a range of  $w$  values.

#### D. Topology and Traffic Dynamics

First we consider the impact of a link failure in the Sprint Network. Path failures and recoveries are detected through active probing. All 50 greedy flows are established at 0 sec. At 5 sec the link between Pennsauken, NJ and Roachdale, IN fails, and it recovers at 10 sec. Flows 20 and 39 contain the affected link in at least one of their paths. In Figure 12, we plot the path rates of the flow 20. We observe that immediately after the failure, traffic is assigned to an alternate path unaffected by the failure. After the link is repaired at time 10 sec,

traffic returns to the original path quickly. Similar behavior is observed for flow 39.

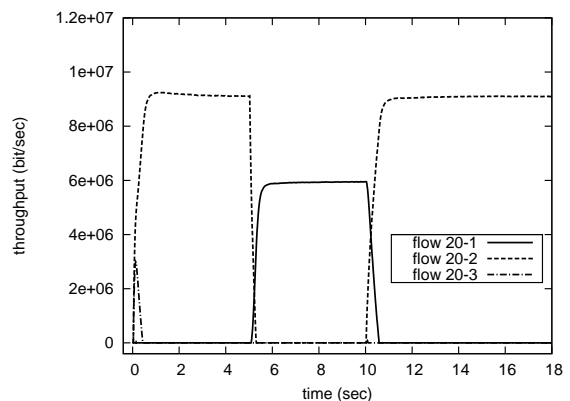


Fig. 12. Plot of affected path rates for a link failure in the Sprint network.

Second, we study the performance of TRUMP under realistic traffic loads by using 10 stochastic ON-OFF flows in the Abovenet network. As suggested by [26], the OFF periods are Pareto with shape 2.0 and average of 0.2s. We consider three file size distributions: exponential, Pareto with shape 1.2 and Pareto with shape 1.8. In Figure 13, we plot the average file size against the *efficiency*: fraction of the actual throughput over the ideal throughput for a 10s period. The ideal throughput is found by solving (5). First, TRUMP's behavior is *independent* of the variance of the

file-size distribution, since all three curves overlap. Second, TRUMP is more efficient for larger files as it takes a few RTTs to converge to the ideal throughput. On the surface, TRUMP performs poorly for small files, only achieving 50% of the ideal rate. However, given those files are transmitted within a single RTT, achieving 50% of the ideal rate is much better than TCP today. In addition, TRUMP is optimized for logarithmic utility, for example  $\log(20,000)/\log(40,000) = 0.93$ . This means TRUMP achieves close to ideal utility even for short-lived flows.

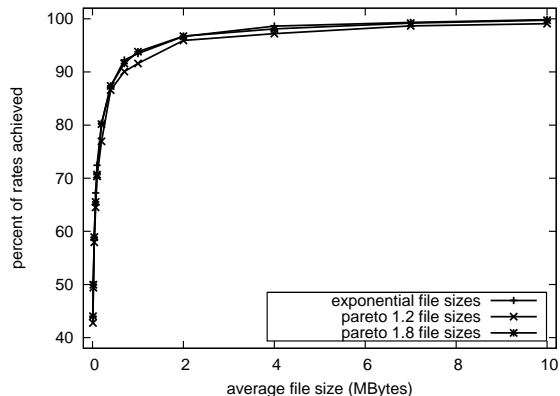


Fig. 13. Plot of average file size versus efficiency for three distributions.

### E. Selecting the Multiple Paths

There are often many paths available between each source-destination pair. In this subsection, we study how many paths to provide TRUMP, and how to select such paths.

Flows	Single path	Two paths	Three paths	Shortest path
5	56.6%	36.8%	6.6%	59.8%
10	66.8%	23.8%	6.7%	51%
25	67.2%	32.4%	0.4%	50.2%
50	76.8.4%	20.4%	2.8%	72.6%
100	70%	27.8%	2.2%	64.4%
250	88.6%	11%	0.4%	70.8%
500	91.8%	8%	0.1%	69%

TABLE V  
IMPACT OF VARYING NUMBER OF FLOWS ON THE SPRINT NETWORK.

We begin by studying how the number of flows (source-destination pairs) affects whether traffic splits over multiple paths, and whether shortest-hop paths are used. For the Sprint topology, we summarize in Table V the number of paths used by flows at equilibrium and percentage of flows using shortest-hop paths. The number of flows in the network impacts the likelihood of a flow being split amongst multiple paths. If there is a single flow in the network, it will use all the paths available to it. So when there are very few flows, a large percentage of flows place load on multiple paths simultaneously since there are many uncongested paths. As the number of flows increases, a larger percentage of flows will just select a single path, since most of the links are used by at least one flow already.

Further, we observe 50% to 73% of the flows send all their traffic on the shortest-hop path(s). Given the penalty function

$f$  is summed over all links, shorter-hop paths are generally preferred because if a longer-hop is taken, then more links are loaded. So if there are two equally loaded paths, the one with fewer hops is preferred. Longer-hop paths are more likely to be used when the network is under-utilized, because a flow might split traffic over two or three paths that are not used by any other flow. Longer-hop paths are also more likely to be utilized when the network is very congested, because a slightly longer-hop path that is much less congested is still attractive. Another advantage of shortest-hop paths is that they can be chosen a priori, while congestion levels depend on dynamic traffic patterns.

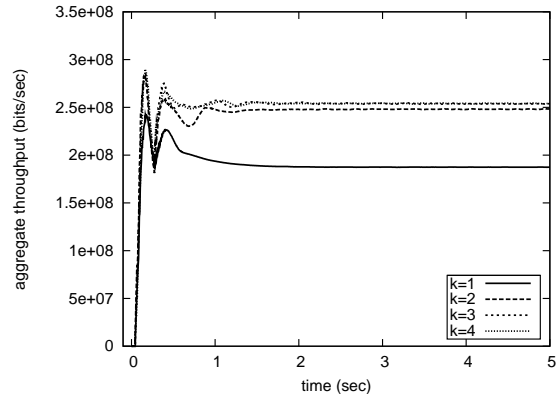


Fig. 14. Plot of aggregate throughput versus time for the Abilene topology with four flows. Different lines correspond to different number of paths available per flow.

Next, we study how the number of paths available to each flow affects the utility achieved and rate of convergence. In Figure 14, we vary the number of paths available to each flow from one to four. For the Abilene topology with four flows, the aggregate throughput increases by 25% when there is more than one path available to a flow, though the gains are much more modest when more than two paths are provided for each source. This observation is inline with research illustrating the power of two choices [27], [28]. The number of flows has no visible impact on the rate of convergence. Looking at Figure 14 and Table V together, we conclude selecting two (or three) shortest-hop paths per source-destination pair is sufficient for TRUMP to perform well.

### F. Fairness of Bandwidth Sharing

As mentioned in Section III-B, TRUMP is  $\alpha$ -fair as  $w \rightarrow 0$ , but its fairness for general  $w$  values is unknown. For  $w = 1$ , we construct a simple topology (Figure 15a) to illustrate whether the bottleneck link is shared fairly. In Figure 15b, we plot throughput of two pairs of flows which differ in RTT or hop-count. All flows have a shared destination (node 9), and the sources are nodes 1, 2 and 3 respectively. We observe that flows 1 and 2, which have very different RTT (30ms and 100ms) but the same number of hops on their paths, share bandwidth fairly. Unlike most congestion control proposals, TRUMP does not discriminate against long RTTs since (4) has no dependency on RTT. While RTTs does indeed affect the transient behavior as indicated in the distributed algorithm of

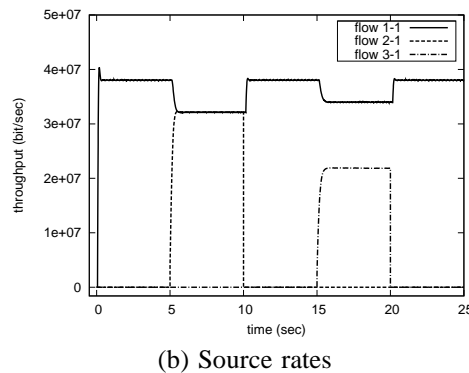
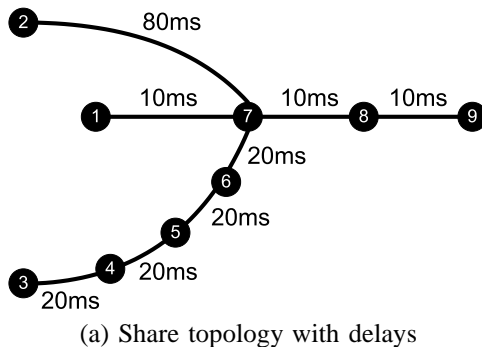


Fig. 15. Fairness of bandwidth sharing.

(13), fairness is an equilibrium property. On the other hand, flow 3 with twice as many hops receives roughly half the bandwidth of flow 1. This is inline with network operator’s goals to penalize against longer-hop paths since that would require more usage of network resources. If the unshared links are lightly loaded, the bandwidth sharing would be less unfair since the amount of penalty depends on link load. It is also possible to change the source rate adaptation for TRUMP to react to path prices normalized by hop length of that path, to ensure fair bandwidth sharing for diverse hop lengths.

## VIII. RELATED WORK

Optimization theory is used in traffic management research in areas such as reverse engineering of existing protocols [5], [6], tuning configuration parameters of existing protocols [2], and guiding the design of new protocols [8] (for more references see [29]). In turn, such a broad use has encouraged innovations in optimization theory, for example, [10] introduced multiple decomposition methods. Our paper takes advantage of the recent advancements and applies multiple decompositions to design traffic management protocols.

Most of the proposed traffic management protocols consider congestion control or traffic engineering alone. Several proposed dynamic traffic engineering protocols also load balance over multiple paths based on feedback from links [11], [9], [30], but they do not adapt the source rates. From the methodology perspective, our work bears the most resemblance to FAST TCP [8]. Other congestion control protocols that use control theory to prove stability include [22], [31], [32].

According to recent research, congestion-control and traffic-engineering practices may not interact well [33], [3], [34]. In response, many new designs are proposed. Some of them start with a different objective than this paper, and find poor convergence properties [14], [13]. Algorithms similar to two of the decomposition solutions (Section IV) are described briefly in [3] and Appendix of [35], though neither considers possible design alternatives, nor present a packet-level protocol (and associated experiments).

Some research analyzes stability of joint congestion control and routing algorithms using theory [17], [36], [37], while we use optimization decomposition to guide the design of a practical protocol. Some of our evaluation is inspired by [37], [17],

which prove that multipath congestion control can be stable under heterogeneous feedback delay. In particular, [17] shares a similar problem formulation and analyzes an algorithm similar to the primal-driven algorithm presented in Section 3.3, however, TRUMP offers extra flexibility through the tuning parameter  $w$  and faster convergence through an universal stepsize. In [28], they study coordinated path selection in the context of multipath congestion control, while in this paper we find selecting two or three shortest-hop paths is sufficient for TRUMP to perform well.

## IX. CONCLUSIONS

In this paper, we searched for a traffic-management protocol which is distributed, adaptive, robust, flexible and easy to manage. We followed a top-down design process starting with an objective which balances the goals of users and operators. We generated four provably optimal distributed solutions using known decomposition techniques. Using insight from simulations comparing the four algorithms, we combined the best parts of each algorithm to construct TRUMP: a simpler traffic management protocol. TRUMP is easy to manage, with just one optional tunable parameter. Our packet-level evaluations confirmed TRUMP is effective in reacting to topology changes and traffic shifts on a small timescale, even with realistic feedback delay. We also found TRUMP’s performance is only weakly dependent on the properties of file size distribution. In addition, our preliminary experiments show TRUMP can achieve fair bandwidth sharing for paths of diverse RTTs, but not for diverse hop count.

This paper started from an abstract model, and ended with a practical traffic management protocol based on feedback from the links along each path. In our ongoing work, we are exploring a version of TRUMP where the sources adapt the path rates based on observations of end-to-end delay and loss. We show that using optimization decompositions as a foundation, simulations as a building block, and engineering intuition as a guide can be a principled approach to protocol design.

## REFERENCES

- [1] J. He, M. Suchara, M. Bresler, M. Chiang, and J. Rexford, “Rethinking Internet Traffic Management: From Multiple Decompositions to A Practical Protocol,” in *Proc. CoNEXT*, December 2007.

- [2] B. Fortz and M. Thorup, "Increasing Internet capacity using local search," *Computational Optimization and Applications*, vol. 29, no. 1, pp. 13–48, 2004.
- [3] J. He, M. Bresler, M. Chiang, and J. Rexford, "Towards Robust Multi-layer Traffic Engineering: Optimization of Congestion Control and Routing," *IEEE J. on Selected Areas in Communications*, June 2007.
- [4] B. Fortz and M. Thorup, "Optimizing OSPF weights in a changing world," *IEEE J. on Selected Areas in Communications*, vol. 20, pp. 756–767, May 2002.
- [5] F. P. Kelly, A. Maulloo, and D. Tan, "Rate control for communication networks: Shadow prices, proportional fairness and stability," *J. of Operational Research Society*, vol. 49, pp. 237–252, March 1998.
- [6] S. H. Low, "A duality model of TCP and queue management algorithms," *IEEE/ACM Trans. Networking*, vol. 11, pp. 525–536, August 2003.
- [7] R. Srikant, *The Mathematics of Internet Congestion Control*. Birkhauser, 2004.
- [8] D. X. Wei, C. Jin, S. H. Low, and S. Hegde, "FAST TCP: Motivation, architecture, algorithms, performance," *IEEE/ACM Trans. Networking*, December 2006.
- [9] A. Elwalid, C. Jin, S. H. Low, and I. Widjaja, "MATE: MPLS Adaptive Traffic Engineering," in *Proc. IEEE INFOCOM*, April 2001.
- [10] D. Palomar and M. Chiang, "A tutorial on decomposition methods and distributed network resource allocation," *IEEE J. on Selected Areas in Communications*, vol. 24, pp. 1439–1451, August 2006.
- [11] S. Kandula, D. Katabi, B. Davie, and A. Charny, "Walking the Tightrope: Responsive Yet Stable Traffic Engineering," in *Proc. ACM SIGCOMM*, August 2005.
- [12] J. He and J. Rexford, "Towards Internet-wide Multipath Routing." To appear in IEEEENM Special Issue on Internet Scalability, March 2008.
- [13] X. Lin and N. B. Shroff, "Utility Maximization for Communication Networks with Multi-path Routing," *IEEE Trans. Automatic Control*, vol. 51, May 2006.
- [14] J. Wang, L. Li, S. H. Low, and J. C. Doyle, "Cross-layer optimization in TCP/IP networks," *IEEE/ACM Trans. Networking*, vol. 13, pp. 582–595, June 2005.
- [15] D. P. Bertsekas, *Nonlinear Programming*. Athena Scientific, second ed., 1999.
- [16] J. He, M. Bresler, M. Chiang, and J. Rexford, "Rethinking Traffic Management: From Multiple Decompositions to A Practical Protocol," March 2007. Princeton University CS Tech. Report TR-774-07. [www.cs.princeton.edu/research/techreps/TR-774-07](http://www.cs.princeton.edu/research/techreps/TR-774-07).
- [17] H. Han, S. Shakkottai, C. V. Hollot, R. Srikant, and D. Towsley, "Multi-Path TCP: A Joint Congestion Control and Routing Scheme to Exploit Path Diversity on the Internet," *IEEE/ACM Trans. Networking*, vol. 14, December 2006.
- [18] J. Mo and J. C. Walrand, "Fair End-to-end Window-based Congestion Control," *IEEE/ACM Trans. Networking*, vol. 8, pp. 556–567, October 2000.
- [19] F. Kelly and T. Voice, "Stability of end-to-end algorithms for joint routing and rate control," *ACM SIGCOMM Computer Communication Review*, vol. 35, pp. 5–12, April 2005.
- [20] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Athena Scientific, second ed., 1997.
- [21] Abilene Backbone. <http://abilene.internet2.edu/>.
- [22] D. Katabi, M. Handley, and C. Rohrs, "Congestion Control for High Bandwidth-Delay Product Networks," in *Proc. ACM SIGCOMM*, August 2002.
- [23] J. M. McQuillan and D. C. Walden, "The ARPA network design decision," *Computer Networks*, vol. 1, pp. 243–289, August 1977.
- [24] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson, "Measuring ISP Topologies with Rocketfuel," *IEEE/ACM Trans. Networking*, vol. 12, pp. 2–16, February 2004.
- [25] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson, "Inferring Link Weights using End-to-End Measurements," in *Proc. Internet Measurement Workshop*, 2002.
- [26] A. Feldmann, A. C. Gilbert, P. Huang, and W. Willinger, "Dynamics of IP traffic: A study of the role of variability and the impact of control," in *Proc. ACM SIGCOMM*, August 1999.
- [27] M. Mitzenmacher, "The power of two choices in randomized load balancing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 12, no. 10, pp. 1094–1104, 2001.
- [28] P. Key, L. Massouli, and D. Towsley, "Path selection and multipath congestion control," in *Proc. IEEE INFOCOM*, May 2007.
- [29] M. Chiang, S. H. Low, R. A. Calderbank, and J. C. Doyle, "Layering as optimization decomposition," *Proceedings of the IEEE*, January 2007.
- [30] S. Fischer, N. Kammenhuber, and A. Feldmann, "REPLEX — Dynamic Traffic Engineering Based on Wardrop Routing Policies," in *Proc. CoNEXT*, December 2006.
- [31] A. Lakshmikantha, N. Dukkipati, R. Srikant, N. McKeown, and C. Beck, "Performance Analysis of the Rate Control Protocol." In submission. <http://yuba.stanford.edu/rcp/>.
- [32] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "A Compound TCP Approach for High-speed and Long Distance Networks," in *Proc. IEEE INFOCOM*, April 2006.
- [33] E. J. Anderson and T. E. Anderson, "On the stability of adaptive routing in the presence of congestion control," in *Proc. IEEE INFOCOM*, April 2003.
- [34] R. Gao, D. Blair, C. Dovrolis, M. Morrow, and E. Zegura, "Interactions of Intelligent Route Control with TCP Congestion Control," in *Proc. of IFIP Networking*, May 2007.
- [35] R. J. Gibben and F. Kelly, "On packet marking at priority queues," *IEEE Trans. Automatic Control*, vol. 47, pp. 1016–1020, December 2002.
- [36] F. Paganini, "Congestion Control with Adaptive Multipath Routing Based on Optimization," in *Proc. Conference on Information Sciences and Systems*, March 2006.
- [37] T. Voice, "Stability of Multi-Path Dual Congestion Control Algorithms," *IEEE/ACM Trans. Networking*, vol. 15, pp. 1231–1239, December 2007.