

Rethinking Internet Traffic Management: From Multiple Decompositions to a Practical 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

In the Internet today, traffic management spans congestion control (at end hosts), routing protocols (on routers), and traffic engineering (by network operators). Historically, this division of functionality evolved organically. In this paper, we perform a top-down redesign of traffic management using recent innovations in optimization theory. First, we propose an objective function that captures the goals of end users and network operators. Using all known optimization decomposition techniques, we generate four distributed algorithms that divide traffic over multiple paths based on feedback from the network links. Combining the best features of the algorithms, we construct TRUMP: a traffic management protocol that is distributed, adaptive, robust, flexible and easy to manage. Further, TRUMP can operate based on implicit feedback about packet loss and delay. We show that using optimization decompositions as a foundation, simulations as a building block, and human intuition as a guide can be a principled approach to protocol design.

1. INTRODUCTION

Traffic management is the adaptation of source rates and routing to efficiently use network resources. 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 tune link weights to minimize a cost function [1]. 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. Second, tuning link weights is an indirect way to control traffic flow through a network; further, the link-weight setting problem is NP-hard, forcing operators to resort to heuristics that can lead to highly suboptimal solutions. Finally, since this offline optimization occurs at the timescale of hours, it does not adapt to changes in the offered traffic.

In this paper, we rethink Internet traffic management using optimization theory as a foundation. Optimization decomposition is the process of decomposing a single optimization problem into many sub-problems, each of which is solved locally. While decomposition is a useful tool for deriving distributed algorithms, it has rarely been used to design a practical protocol, with FAST TCP [2] as a notable exception. The barriers 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, 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 2. In Section 3, we derive four distributed solutions where sources adapt their sending rates along multiple paths, based on different kinds of feedback from the links, using optimization decomposition techniques discussed in [3]. 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 4. We combine the best features of each algorithm to construct a simple traffic management protocol in Section 5. Our contributions are:

- **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.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CoNEXT'07, December 10-13, 2007, New York, NY, U.S.A.
Copyright 2007 ACM 978-1-59593-770-4/07/0012 ...\$5.00.

- **Redesigned Traffic Management:** We introduce TRUMP, a TRaffic-management Using Multipath Protocol to replace congestion control and traffic engineering. TRUMP is easy to manage and robust to small-timescale traffic shifts.

TRUMP converges faster than the four algorithms presented in Section 3, and has the fewest tunable parameters. As with any mathematical modeling, the TRUMP algorithm leaves many protocol details unspecified. We use our intuition to address these details. In Section 6, the TRUMP protocol is evaluated using packet-level simulations with realistic topologies and traffic patterns. The protocol still leaves room for interpretation regarding the division of functionality. We defer the discussions on deployment issues until Section 7. Finally, we discuss related work in Section 8 and conclude in Section 9.

2. CHOOSING AN OBJECTIVE FUNCTION

Every optimization problem consists of an objective function, 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. In this section, we start with an objective of maximizing aggregate user utility, but simulations reveal its solution converges slowly and is sensitive to step size. In addition, maximizing utility leads to bottlenecks in the network, making the network fragile to traffic bursts. To address practical challenges, we select an objective which balances maximizing user utility with minimizing operator’s cost function.

2.1 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 [4, 5]. 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 [6, 7], 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 prob-

lem. 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:

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

In this form, (2) is a convex optimization problem. A distributed solution to (2) can be derived using *dual decomposition* [6], where a *dual variable* is introduced to relax the capacity constraint. The resulting Dual-based Utility Maximizing Protocol (DUMP) involving sources and links is summarized in Figure 1. Similar to the reverse engineering of the congestion-control protocol in [5], 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 β_s is the feedback price step size.

Path rate update at source i , path j :

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

Figure 1: 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 [5] 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 step sizes, such as $\beta_s(t) = \beta/t$ where $\beta > 0$ is a constant, guarantee that DUMP will converge to the joint optimum as $t \rightarrow \infty$ [8]. However, such diminishing step size 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. Even under the simplest of topologies and assuming greedy flows, DUMP has poor convergence behavior [6, 9]. We observe that, when the step size is too large, DUMP will constantly overshoot or undershoot, never reaching the ideal utility. On the other hand, when the step size 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 step size for DUMP is challenging.

2.2 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 current congestion control mechanism 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 [5]. In addition, utility is only based on throughput, while having low delay is also important to traffic management. In addition, the authors of [10] suggest the network would be driven to a solution where some links are operating near capacity if 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 [1]. 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 [10, 11], though without w . Here w is a tuning 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 in avoiding solutions which are close to capacity. Today, operators tune link weights today depending on the given traffic, in this case, they can tune w depending on the given traffic. Fairness is another important consideration. From a theoretical perspective, the solution to (4) is α -fair as $w \rightarrow 0$, where α -fairness is defined in [12]. While this does not hold for general values of w , our experimental results in Section 6.4 are encouraging.

Before generating distributed solutions in Section 3, 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 actual capacity.

3. MULTIPLE DECOMPOSITIONS

In this section, we describe the distributed algorithms generated from optimization decompositions of (4) (the decomposition techniques are surveyed in [3, 4]). In all four resulting algorithms, the solutions update their path rates based on feedback prices from links. Optimization decomposition leads us to three useful notions: effective capacity, consistency price and direct path-rate update. Practically, there are several other similarities between the four algorithms. They only impose a small amount of overhead on the links including measuring the link load. They also incur the same amount of message passing overhead, *i.e.*, passing one link price to the sources. While computations can involve solving a local optimization problem and taking derivatives, U and f are twice differentiable, therefore closed-form solutions

are available and they are just simple function evaluations. The computational complexity for all four algorithms are constant per link and linear per source. The main difference is the number of tunable parameters of each algorithm, which varies from one to three.

3.1 Effective Capacity

The first three algorithms 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 1. The feedback price is similar to that in Figure 1, except 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 2.1, we consider constant stepsize for practical reasons, thus we remove the t argument from all the step sizes.

3.1.1 Local Optimization: Partial-Dual

The derivation process for the **partial-dual** algorithm is identical to Section 2.1 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 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 .

3.1.2 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 1 except for the feedback price update uses the effective capacity \mathbf{y} rather than actual capacity \mathbf{c} .

The master problem can be solved through an iterative update on 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 β_y is the effective capacity step size. 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 ($f'_l(y_l)$). In a nutshell, the primal-dual decomposition is identical to the partial-dual decomposition in Section 3.1.1 except that the effective capacity is updated iteratively through (10) rather than by solving a local minimization problem.

3.2 Consistency Price: Full Dual

The **full-dual** decomposition is quite similar to the partial-dual decomposition in Section 3.1.1, 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 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 β_p is the step size 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:

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

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 [10], though our objective contains w as a weighing factor. Appendix 2 of [10] also shows a complete derivation of the full-dual algorithm.

3.3 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 appendix of [13]. 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 [14]. 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 β_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 2.

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:

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

Figure 2: 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 2 differs significantly from the first three decompositions in two ways. First, it uses direct subgradient update on the path rates. Second, it does not use the concept of effective capacity.

4. CONVERGENCE PROPERTIES

In this section, we study convergence properties of the four algorithms, and make three observations which will guide our design of a new protocol in Section 5. 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 .

4.1 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 [12]. 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 [1].

We study two realistic topologies as shown in Figure 3. On the left is a tree-mesh topology, which is representative of a common access-core network structure. On the right is the Abilene backbone network [15]. We select six source-destination pairs for access core and four pairs for Abilene. For each source-destination pair, we choose three minimum-hop paths as possible paths for access-core and four minimum-hop paths as possible paths 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 routing conditions. Due to space constraints, we omit extra graphs when the same trends are observed across algorithms, topologies and values of w , more graphs can be found in [9].

4.2 Weighing User Utility and Operator Cost

In this section, we illustrate *a trade-off between aggregate utility and convergence time*. In Figure 4, we plot the number of iterations before convergence against step-size for three values of w for the partial-dual algorithm from Section 3.1.1. For each step-size, 10 random capacity distributions are chosen and the average number of iterations before convergence is highlighted in a solid line. Comparing across Figure 4 from left to right, we see that as w shrinks, the convergence time at the optimal step size grows and the range of step sizes with a good convergence time shrinks. This helps understand why DUMP ($w = 0$) is hard to tune.

In Figure 5, 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 both topologies. For the access-core topology, this knee region is $w = [1/4, 1/6]$; for the Abilene topology, this knee region is $w = [1/6, 1/10]$. Below this knee region, the algorithm achieves near maximal aggregate utility, since the cost function f is weighed sufficiently lightly to not change the solution. Above this knee region, the aggregate utility achieved decreases, as the cost function f becomes a significant part of the objective.

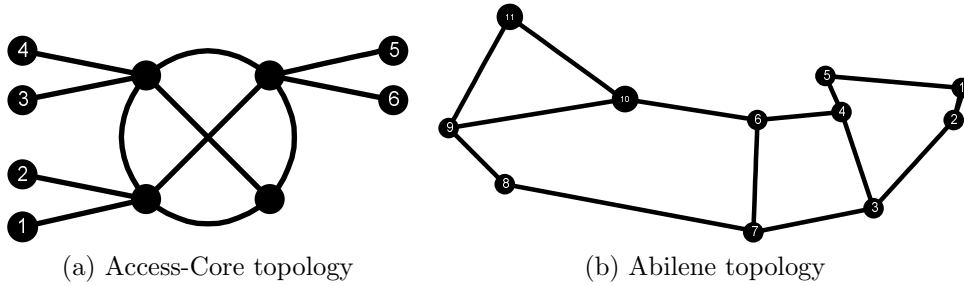


Figure 3: Two topologies.

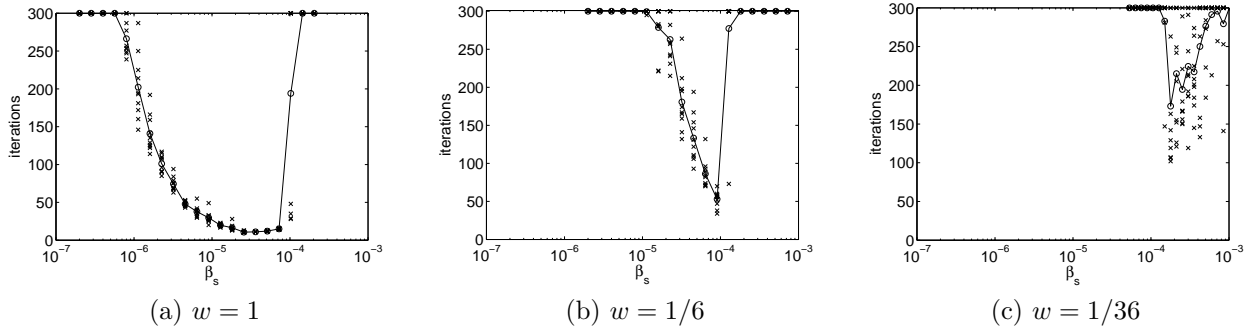


Figure 4: Plots of partial-dual algorithm showing dependence of convergence time on step-size. 'x' represent the actual data points and 'o' represent the average value. Access-core topology.

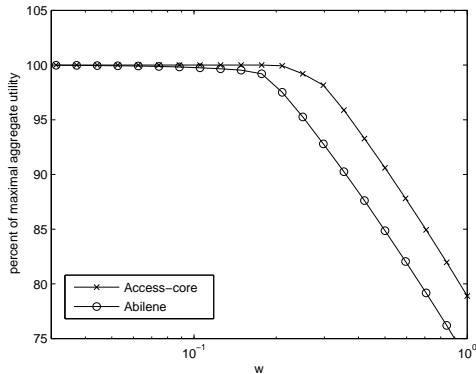


Figure 5: Plot of w versus percentage of maximal utility achieved.

Looking at Figure 4 and 5 together, one can observe there are some values of w that have faster convergence and less sensitivity to step size without much sacrifice in utility. Below the knee region of Figure 5, the gains in aggregate utility do not offset the gain in convergence time. Otherwise, there is a trade-off between aggregate utility and convergence time. Depending on the conditions of their network, operators can choose their desired operation point.

4.3 Comparing Between the Algorithms

In this subsection, we do a series of comparisons between convergence time and step-size sensitivity of the four algorithms, and find partial-dual in Figure 4 is the best overall, with a good convergence profile and fewest tunable parameters. Due to space constraints, we summarize our observations in Table 1, but more details can be found in [9].

Comparing the primal-dual algorithm in Section 3.1.2 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 β_y and k . For other values of β_y , however, we find the primal-dual algorithm converges more slowly than the partial-dual algorithm.

Comparing the full-dual algorithm in Section 3.2 to the partial-dual algorithm, we find *consistency price may improve convergence properties*. From Table 1, we note that β_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 step size plays no role. For $w = 1/6$ (which is the edge of the knee region seen in Figure 5), 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 vio-

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 1: Summary of average number of iterations to convergence for best chosen tuning parameters. Here * denotes sensitivity to step-size variation and ** denotes extra sensitivity to step-size variation.

lated during transient periods, the algorithm can take more aggressive steps and potentially converge faster.

Comparing the primal algorithm in Section 3.3 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 step-size is constrained to react with the same strength each time, while local minimization can react more flexibly. From Table 1, the primal algorithm takes longer to converge at the optimal step size (25 iterations versus 15 iterations). In addition, the primal algorithm also requires operators to tune a second parameter g .

5. TRUMP

While the algorithms introduced in Section 3 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.

5.1 TRUMP: A Mathematical Algorithm

Our simulations in the previous section suggest that simpler algorithms with fewer tunable parameters converged 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 6. 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 2 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 step size.

By a similar argument, we use a local optimization for the path rate update as in the dual-based algorithms. The w -value is only known at the sources where the z 's are computed, and there is only a single value for the network. The exact value is a judgment call based on an understanding (from measurements) of how stochastic the traffic is. Simulations such as those performed in Section 6.2 could reveal the right trade-off between the level of stochasticity and the value of w .

Through simulations, we find that TRUMP indeed

Feedback price update:

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

Loss price update:

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

Delay price update:

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

Path-rate update:

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

Figure 6: The TRUMP algorithm.

converges to the optimum of (4) for both topologies and a range of w values. When we plot its achieved aggregate utility at equilibrium versus w , we obtain a plot identical to Figure 5. In Figure 7, we plot convergence time versus step-size for TRUMP. When the network is reacting quite strongly to delay price q ($w = 1$ and the traffic engineering part is dominating), the loss price p is unnecessary as observed in Figure 7a. In the region where the network is being less conservative ($w = 1/6$), loss price p is a more definitive indicator of performance than delay price q , and can be helpful for guiding source reactions. Comparing Figure 7 to Figure 4, we see that TRUMP has nicer convergence properties than the partial-dual algorithm, while requiring fewer parameters.

Unlike the algorithms from Section 3, TRUMP is a heuristic and does not correspond to a known decomposition. Consequently, the convergence and optimality is not automatically guaranteed by optimization theory. We were able to prove the convergence of TRUMP when the network is lightly loaded, see Appendix A of [9]. We consider the region where w is sufficiently large for $p = 0$ (as seen in Figure 7a), and find a contraction mapping on z . Overall, TRUMP is simpler than any of the al-

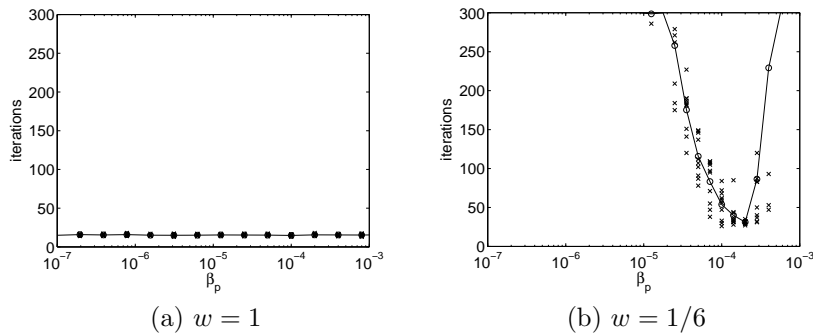


Figure 7: Plots of TRUMP algorithm showing dependence of convergence time on step-size. 'x' represent the actual data points and 'o' represent the average value. Access-core topology.

gorithms presented in Section 3, with only one tunable parameter that only needs to be tuned for small w .

5.2 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 6 ignores *feedback delay*. Second, the algorithm assumes traffic flows fluidly, while real traffic consists of *packets*. Third, a *specific* U and f must be selected. We address these differences in the TRUMP protocol.

Each iteration of the TRUMP algorithm is dependent on 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:

$$\begin{aligned} p_l(t + T) &= [p_l(t) - \beta_p(c_l - \frac{N_T}{T c_l})]^+, \\ q_l(t + T) &= \frac{w}{c_l} * \exp\left(\frac{N_T}{T c_l}\right), \\ s_l(t + T) &= p_l(t + T) + q_l(t + T). \end{aligned}$$

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

$$z_j^i(t + T_j^i) = z_j^i(t) - \gamma \sum_j z_j^i(t) + \frac{\gamma}{\sum_l H_{lj}^i s_l(t)}. \quad (13)$$

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.* [2], we introduce a parameter $0 < \gamma < 1$. In general, path rates are updated

every γRTT_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 γ and T are necessary for any packet-level protocol.

6. TRUMP: PACKET-LEVEL EVALUATION

In our MATLAB simulations, we had implicitly assumed flows are greedy and persistent. Moving to packet-level simulations, we study how the TRUMP protocol performs under feedback delay, link failures and traffic shifts. In addition, we examine whether TRUMP shares bottleneck links fairly.

6.1 Experimental Set-up

We implement the TRUMP protocol in NS-2 as described in Section 5.2. In particular, the link prices are updated every 5ms and feedback from the links to the sources is piggy-backed on acknowledgment packets. 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 2 and 3 respectively. For the topologies used in our MATLAB experiments (Figure 3), 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 50 ms, a value chosen to test TRUMP under long feedback delay. Specific paths and link delays are selected in the Share topology (Figure 11a) to test the fairness of TRUMP. Links have a capacity of 200Mb/s, except for the bottleneck link from node 7 to node 8, which has a capacity of 100Mb/s.

Since TRUMP with explicit feedback is most easily deployed inside a single AS, we obtained intra-AS

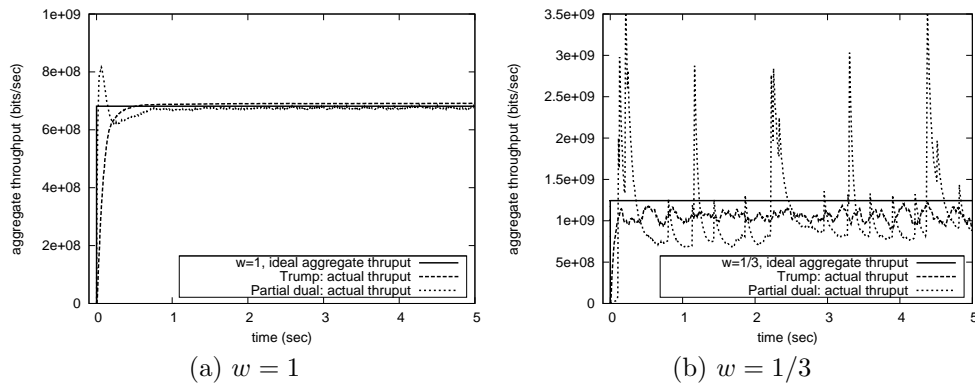


Figure 8: Ideal and actual aggregate throughput in the Sprint network with 50 greedy flows.

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

Table 2: Summary of synthetic topologies.

topologies, along with link delays from the Rocketfuel topology mapping engine [16, 17]. The link capacities are 100Mb/s if neither endpoint has degree larger than 7, and 52Mb/s otherwise. As summarized in Table 3, 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 1 to 400 ms.

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	50	1-4
Tiscali(3257)	41	174	25	1-4
Abovenet(6461)	19	68	10	1-4
AT&T(7018)	115	296	50	1-4

Table 3: Summary of ISP topologies.

6.2 Effect of w

We confirm our MATLAB results from Section 5.1: TRUMP’s converges quickly for $w = 1$, under heterogeneous feedback delay. In Figure 8, we plot the ideal and actual aggregate throughput in the Sprint network with 50 greedy flows. Ideal throughput is the sum of the rates that optimizes (4), and actual throughput is the sum of the rates that our implementation of TRUMP achieved. 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, when $w = 1/3$, the actual throughput of TRUMP and partial-dual oscillates. The reason the ideal-valued throughput is not reachable is the same thing we observed in DUMP. The theory says it works with diminishing step-size, but there’s no guarantee for constant step-size. 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. The partial-dual oscillates slightly around the equilibrium point. The same convergence properties are observed across both synthetic and realistic topologies. Therefore, the TRUMP protocol is *stable* under realistic feedback delay for sufficiently large w .

6.3 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 9, 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.

We study the performance of TRUMP under realistic traffic loads by using 10 stochastic ON-OFF flows in the Abovenet network. As suggested by [18], 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 10, we plot the average file size against the *efficiency*: fraction of the actual throughput over the ideal throughput for a 10s period. First, TRUMP’s behavior

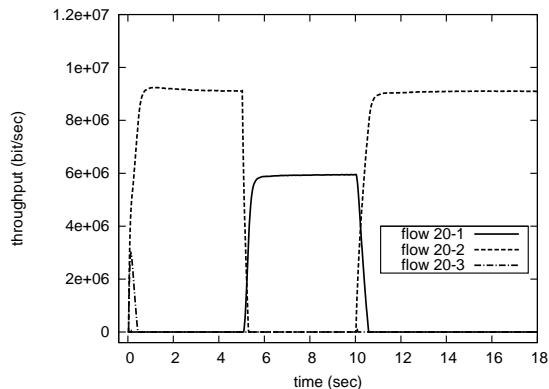


Figure 9: Plot of affected path rates for a link failure in the Sprint network.

is *independent* of the variance of the file-size distribution, since all three curves overlap. For all three distributions, 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 small flows.

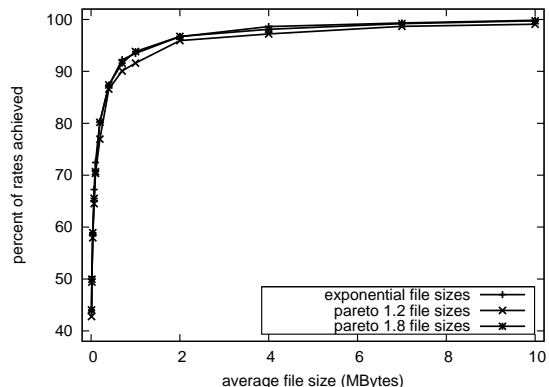


Figure 10: Plot of average file size versus efficiency for three distributions.

6.4 Fairness of Bandwidth Sharing

As mentioned in Section 2.2, TRUMP is α -fair as $w \rightarrow 0$, but its fairness for general w values is unknown. For $w = 1$, we construct a simple topology (Figure 11a) to illustrate whether the bottleneck link is shared fairly. In Figure 11b, 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 (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 operators 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 hops.

7. PRACTICAL DEPLOYMENT ISSUES

The TRUMP protocol still leaves many questions unanswered such as: which network elements correspond to the sources, and how the multiple paths are determined. TRUMP being under-specified enables flexible integration with the network architecture. In this section, we explore several deployment options.

Are the “sources” end hosts or edge routers?

The mathematics does not specify whether the sources refer to the end hosts or the edge routers. The interpretation of the sources depends on two factors: whether the end host has control over and access to the multiple paths and whether the network can trust the end hosts to send at the prescribed rate. If end hosts are unaware of the multiple paths, then edge routers rate limit the end hosts, and split traffic amongst the multiple paths. Even if the end hosts are aware of the multiple paths, they might send too aggressively. In this case, edge routers should perform the same calculations and perform policing or shaping to enforce the path rates by dropping excess packets.

How is H determined? In the TRUMP protocol, there is no hint as to how a source can access multiple paths. Luckily, many options exist. Inside a single AS, routers can compute K -shortest paths, or a management system can set-up multiple tunnels. Across ASes, TRUMP is easily deployed at multihomed stub networks which can already access multiple paths to each destination. Another possibility is in the context of Content Distribution Networks, where a customer can download content from multiple servers. There are also incrementally deployable ways of accessing end-to-end multiple paths [19].

Is the feedback explicit or implicit? The TRUMP protocol described in Section 5.2 uses explicit feedback. There are several implementation options: the feedback

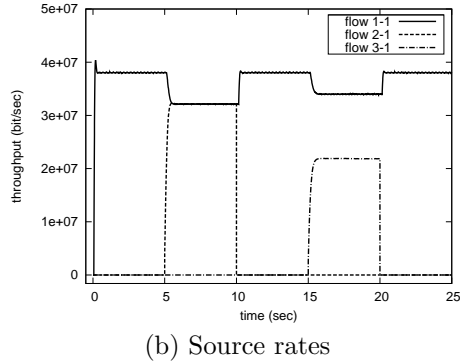
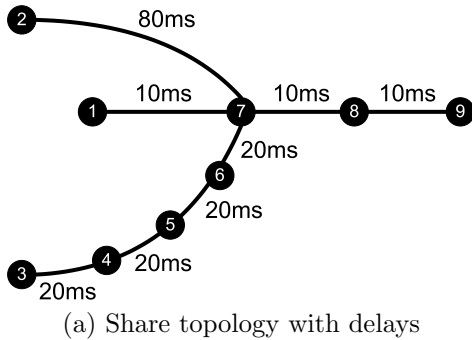


Figure 11: Fairness of bandwidth sharing.

from links to sources could be piggy-backed on acknowledgment packets. Regardless of the method, explicit feedback requires cooperation between sources and intermediate nodes, knowledge of the end-to-end multiple paths and imposes message-passing overhead. All these disadvantages can be avoided by protocol based on *implicit feedback*, *i.e.*, locally observable qualities. Fortunately, the link prices p_l and q_l correspond *loosely* to packet loss and queuing delay, which a source can easily estimate¹, though all edge routers (or sources) may still need to agree on the same w . Combining implicit feedback with flexible multipath routing for TRUMP is an exciting avenue for future research.

8. RELATED WORK

Optimization theory is used in traffic management research in areas such as reverse engineering of existing protocols [4, 5], tuning configuration parameters of existing protocols [1], and guiding the design of new protocols [2] (for more references see [20]). In turn, such a broad use has encouraged innovations in optimization theory, for example, [3] 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 [21, 22, 23], but they do not adapt the source rates. From the methodology perspective, our

¹ p_l is *loosely* related to packet loss on link l : $\max(0, \sum_i \sum_j H_{ij}^i z_j^i(t) - c_l)$ corresponds to packet loss on link l , and β_p moderates how much to react to packet loss. Thus, the sum of p_l along the path can be approximated as end-to-end packet loss. If we interpret f' as *approximating* M/M/1 queuing delay at a link, then q_l can be interpreted as being proportional to queuing delay. Thus, the sum of q_l along the path can be viewed as an approximation of the end-to-end average queuing delay.

work bears the most resemblance to FAST TCP [2]. Other congestion control protocols that use control theory to prove stability include [24, 25, 26].

According to recent research, congestion-control and traffic-engineering practices may not interact well [27, 10, 28]. In response, many new designs are proposed. Some of them start with a different objective, and find poor convergence properties [7, 6]. Algorithms similar to two of the decomposition solutions (Section 3) are described briefly in [10] and Appendix of [29], though neither consider possible design alternatives, nor bridge the gap between a mathematical algorithm and a practical protocol. Others analyze stability of joint congestion control and routing algorithms using theory, while we use optimization decomposition to guide the design of a practical protocol [11, 30, 31]. Some of our evaluation is inspired by [31, 11], which proves that multipath congestion control can be stable under heterogeneous feedback delay. In particular, [11] shares a similar problem formulation and analyze an algorithm similar to the primal-driven algorithm presented in Section 3.3, TRUMP offers extra flexibility through the tuning parameter w and faster convergence through an additional loss price. In [32], the value of coordinated path selection (over random path selection) in the context of multipath congestion control is studied, while in this paper we consider the combination of traffic engineering and congestion control.

9. 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 sim-

pler 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 human intuition as a guide can be a principled approach to protocol design.

Acknowledgments

We would like to thank Renata Teixeira, Wenjie Jiang, Changhoon Kim, Yaping Zhu and Dahai Xu for their feedback on earlier drafts of this paper. We also appreciate Jim Kurose, Dani Palomar and the anonymous reviewers for their insightful comments. This work has been supported in part by NSF grants CNS-0519880 and CCF-0448012, Cisco grant GH072605, and DARPA seedling W911NF-07-1-0057.

10. REFERENCES

- [1] B. Fortz and M. Thorup, "Increasing Internet capacity using local search," *Computational Optimization and Applications*, vol. 29, no. 1, pp. 13–48, 2004.
- [2] D. X. Wei, C. Jin, S. H. Low, and S. Hegde, "FAST TCP: Motivation, architecture, algorithms, performance," *IEEE/ACM Trans. Networking*, December 2006.
- [3] 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.
- [4] 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.
- [5] S. H. Low, "A duality model of TCP and queue management algorithms," *IEEE/ACM Trans. Networking*, vol. 11, pp. 525–536, August 2003.
- [6] X. Lin and N. B. Shroff, "Utility Maximization for Communication Networks with Multi-path Routing," *IEEE Trans. Automatic Control*, vol. 51, May 2006.
- [7] 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.
- [8] D. P. Bertsekas, *Nonlinear Programming*. Athena Scientific, second ed., 1999.
- [9] 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.
- [10] 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.
- [11] 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.
- [12] 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.
- [13] 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.
- [14] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Athena Scientific, second ed., 1997.
- [15] Abilene Backbone. <http://abilene.internet2.edu/>.
- [16] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP Topologies with Rocketfuel," in *Proc. ACM SIGCOMM*, August 2002.
- [17] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson, "Inferring Link Weights using End-to-End Measurements," in *Proc. Internet Measurement Workshop*, 2002.
- [18] 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.
- [19] J. He and J. Rexford, "Towards Internet-wide Multipath Routing," June 2007. Princeton University Tech. Report TR-787-07, www.cs.princeton.edu/research/techreps/TR-787-07.
- [20] M. Chiang, S. H. Low, R. A. Calderbank, and J. C. Doyle, "Layering as optimization decomposition," *Proceedings of the IEEE*, January 2007.
- [21] S. Kandula, D. Katabi, B. Davie, and A. Charny, "Walking the Tightrope: Responsive Yet Stable Traffic Engineering," in *Proc. ACM SIGCOMM*, August 2005.
- [22] A. Elwalid, C. Jin, S. H. Low, and I. Widjaja, "MATE: MPLS Adaptive Traffic Engineering," in *Proc. IEEE INFOCOM*, April 2001.
- [23] S. Fischer, N. Kammenhuber, and A. Feldmann, "REPLEX — Dynamic Traffic Engineering Based on Wardrop Routing Policies," in *Proc. CoNEXT*, December 2006.
- [24] D. Katabi, M. Handley, and C. Rohrs, "Congestion Control for High Bandwidth-Delay Product Networks," in *Proc. ACM SIGCOMM*, August 2002.
- [25] A. Lakshminantha, N. Dukkkipati, R. Srikant, N. McKeown, and C. Beck, "Performance Analysis of the Rate Control Protocol." In submission. <http://yuba.stanford.edu/rcp/>.
- [26] 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.
- [27] 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.
- [28] 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.
- [29] R. J. Gibben and F. Kelly, "On packet marking at priority queues," *IEEE Trans. Automatic Control*, vol. 47, pp. 1016–1020, December 2002.
- [30] F. Paganini, "Congestion Control with Adaptive Multipath Routing Based on Optimization," in *Proc. Conference on Information Sciences and Systems*, March 2006.
- [31] T. Voice, "Stability of Multi-Path Dual Congestion Control Algorithms." To appear in *IEEE/ACM Trans. Networking*.
- [32] P. Key, L. Massouli, and D. Towsley, "Path selection and multipath congestion control," in *Proc. IEEE INFOCOM*, May 2007.