Rethinking Internet Traffic Management: From Multiple Decompositions to a Practical Protocol

Jiayue He, Ma'ayan Bresler, Mung Chiang, and Jennifer Rexford
Princeton University, USA
Email: {jhe, mbresler, chiangm, jrex}@princeton.edu
Princeton Tech Report TR-774-07

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 slowly evolved without a conscious design. 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 alike. Using all known optimization decomposition techniques, we generate four distributed algorithms where sources adapt their sending rates along multiple paths, based on different kinds of feedback from the links. Optimization theory guarantees that these algorithms converge to a stable and optimal point, and simulations allow us to compare rate of convergence, robustness to tunable parameters, and performance under realistic traffic. Combining the best features of the algorithms, we construct TRUMP: a traffic management protocol that is distributed, adaptive, robust, flexible and easy to manage. 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 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 set link weights through solving centralized optimization. This optimization minimizes a cost function that considers the resulting utilizations on all links given the offered traffic (*e.g.*, [1, 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 sending rates assuming routing is fixed. Second, the link-weight setting problem is NP-hard, forcing operators to resort to heuristics. 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 decompositions as a foundation. Optimization decomposition is the process of decomposing a single optimization problem into many sub-problems, each of which can then be 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 [3] as a notable exception. The barriers are two-fold. First, the mathematics leaves many important practical details unspecified. Second, recent work [4] shows that there are multiple decomposition methods, leading to multiple algorithms with potentially different practical properties, but there lacks theory to compare between them. To our best knowledge, this is the *first* work that systematically compares between *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 formulating the right problem (*i.e.*, identifying the objective function and constraints). We then derive distributed solutions using optimization decomposition techniques. After comparing performance properties of different solutions through simulations, we combine the best features of each to construct a simple traffic management protocol. Our contributions are two-fold:

- **Redesigned Traffic Management:** We introduce TRUMP, a TRaffic-management Using Multipath Protocol to replace congestion control and traffic engineering today. It is optimal, easy to manage, and robust to small-timescale traffic shifts.
- **Protocol Design using Decompositions:** We demonstrate how to create a practical network protocol by generating multiple decompositions, comparing their practical properties, and synthesizing their best features.

We start with an optimization objective of maximizing aggregate user utility, but discover that the resulting algorithm has poor convergence properties. This leads us to an alternative objective function: a weighted difference of the goals of the users (to maximize their utility) and the operators (to prevent heavily-congested links). We then apply all known decomposition techniques to generate four different distributed solutions to the new optimization problem. In all four algorithms, sources adapt their sending rates on each of the multiple paths given feedback price from the links, but differ in how the prices are computed.

While optimization theory guarantees all four algorithms to converge to the optimal solution at equilibrium, it falls short on several practical considerations. First, there is no theory for choosing the appropriate tunable parameters, which could depend on network topology, link capacities and other network properties. Second, the rate of convergence is only loosely bounded by theory, so two algorithms with the same bound could behave drastically differently in practice. Finally, under stochastic dynamics, the existing mathematical machinery in stochastic optimization can at best help characterize the stability of queues rather than transient behavior or performance metrics. In this paper, we address these issues through simulations.

From our simulations, we observe that all four algorithms indeed converge to the optimal solution, but at different rates and with different sensitivity to tunable parameters. Learning from the simulation results, we combine best parts from different decompositions to construct a TRaffic-management Using Multipath Protocol (TRUMP). TRUMP converges more quickly than the previous decompositions and has the fewest tunable parameters. It also leads to an intuitive interpretation where the feedback price is based on a combination of *packet loss* and *queuing delay*. This paper showcases that a heuristic protocol with better performance properties can be found through the process of considering multiple decompositions.

In this paper, we start with abstract formulation of a problem, which we interpret with a progressive amount of detail as the paper progresses. Since an optimization model operates only at the abstract level of "sources" and "links", the mathematics leaves much room for interpretation regarding the division of functionality. For example, "sources" could refer to edge routers or end hosts, computations could be done distributedly or centrally, and feedback could be implicit or explicit. We defer the discussions regarding *architectural implications* of TRUMP until Section 7. We do not address issues associated with incremental deployment, as our goal is to *redesign traffic management* not to retrofit a solution into today's architecture.

2. CHOOSING AN OBJECTIVE FUNCTION

In this section, we look for the best formulation of the optimization problem. Every optimization problem consists of an objective function, constraint set, variables, and constants. 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, with the topology and link capacities as constants. The objective function remains to be determined by design. 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. From our observations, we motivate a different objective function. The first few subsections serve as a preview to the rest of the paper. We introduce dual decomposition, a standard optimization technique to derive a distributed solution to a given optimization problem, which will form the basis for multiple decompositions. We also give a glimpse of the simulations which we will use to compare between multiple decomposition solutions. The key notation used in this paper is summarized in Table 1.

Symbol	Meaning
x_i	Rate of source i.
$U_i(x_i)$	Utility function for source i.
R_{li}	Fraction of traffic on link l for source i.
c_l	Capacity of link l.
z_j^i	Rate of source i on its j^{th} path.
$\mathbf{H}$	Network topology as the set of all paths.
s_l	Feedback price on link l.
β_s	Step size for update of feedback price.
t	Time.
T_p	Time it takes for feedback.
f	Cost function.
w	Weight of the cost function.
y_l	Effective capacity on link l.
β_y	Step size for effective capacity.
T_m	Time for master problem to iterate.
p_l	Consistency price on link l.
β_p	Step size for consistency price.
β_z	Step size for path rate.

Table 1: Summary of notation.

2.1 Maximizing Aggregate Utility

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 user i 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 through 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 [7, 8], this is the resulting optimization problem:

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

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. In single path routing, $R_{li} = 1$ if link l is on the path and 0 otherwise. In multipath routing, the entries of $\mathbf{R}$ can take any value between 0 and 1. To capture multipath routing, we introduce z^i_j to represent the sending rate of source i on its jth path. We also represent available paths by a matrix $\mathbf{H}$ where

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

$\mathbf{H}$ 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:

$$\begin{array}{ll} \text{maximize} & \sum_i U_i(\sum_j z^i_j) \\ \text{subject to} & \sum_i \sum_j H^i_{lj} z^i_j \leq c_l, \ \ \forall l. \end{array} \tag{2}$$

In this form, (2) is a convex optimization problem.

2.2 Dual Decomposition

Now that we have a convex optimization problem formulation, we can then proceed to derive a distributed solution to (2) through *dual decomposition*. The variables which appear in the original problem *e.g.* $\mathbf{z}$ are called *primal variables* while the variables introduced to relax constraints are called *dual variables*. We relax (2) by forming the following Lagrangian:

$$L(\mathbf{z}, \mathbf{s}) = \quad \sum_i U_i(\sum_j z^i_j) + \sum_l s_l(c_l - \sum_i \sum_j H^i_{lj} z^i_j)$$

where $s_l \geq 0$ is the dual variable associated with the capacity constraint on link l. Additivity of total utility and linearity of flow constraints lead to a Lagrangian dual decomposition into the following per-source subproblem:

$$\text{argmax}_{z^i_j}(U_i(\sum_j z^i_j) - z^i_j \sum_l s_l H^i_{lj}.)$$

The Lagrangian dual function $h(\mathbf{s})$ is defined as the maximized $L(\mathbf{z}, \mathbf{s})$ over $\mathbf{z}$ for a given $\mathbf{s}$. Each source can compute an optimizer $\mathbf{z}^{i*}(\mathbf{s})$. The Lagrange dual problem of (2) is:

$$\begin{array}{ll} \text{minimize} & h(\mathbf{s}) = L(\mathbf{z}^*(\mathbf{s}), \mathbf{s}) \\ \text{subject to} & \mathbf{s} \succeq \mathbf{0}, \end{array} \tag{3}$$

and can be solved locally by each link. Note that (3) is a convex minimization. Since $h(\mathbf{s})$ may be non-differentiable, an iterative subgradient method can be used to update the dual variable $\mathbf{s}$ to solve (3):

$$s_l(t+1) = [s_l(t) - \beta_s(t)(c_l - \sum_i \sum_j H^i_{lj} z^i_j(t))]^+,$$

where $\beta_s(t)$ represents dual variable step size.

2.3 Dual-based Utility Maximizing Protocol

The distributed algorithm derived in the previous subsection can be interpreted similarly to the reverse engineering of the congestion-control protocol in [6]. The resulting Dual-based Utility Maximizing Protocol (DUMP) involving users (sources) and routers (links) is summarized in Figure 2.

Feedback price update at link l:

$$s_l(t+T_p) = \left[s_l(t) - \beta_s(t) \left(c_l(t) - \sum_i \sum_j H^i_{lj} z^i_j(t) \right) \right]^+,$$

where β_s is the feedback price step size.

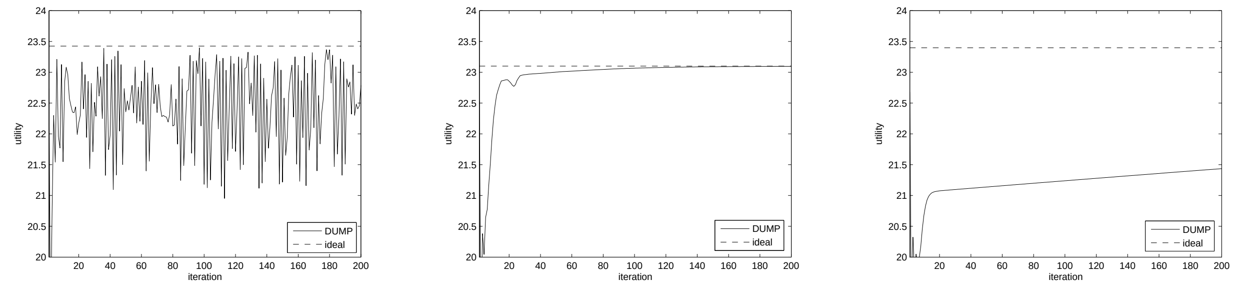
Path rate update at source i, path j:

$$z^i_j(t+T_p) = \text{maximize}_{z^i_j} \left(U_i(\sum_j z^i_j) - z^i_j \sum_l s_l(t) H^i_{lj} \right)$$

Figure 2: Dual-based utility maximizing protocol.

Here t represents the iteration number and each iteration is at the same timescale as the longest Round Trip Time (RTT) of the network. The parameter T_p represents the propagation delay for the edge router to receive the congestion-price information from the links. At each link, s_l is updated based on the difference between the link load $\sum_i \sum_j H^i_{lj} z^i_j$ 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^i_j 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.

Architecturally, one interpretation is to have the path-rate computation done at the sources and the feedback price computation done at the links. The sources would



(a) Step size = 0.0001 (b) Step size = 0.00001 (c) Step size = 0.000001

Figure 1: Plots of aggregate utility versus iteration to show the sensitivity to step size choices.

then receive s_l as explicit control messages. Alternative architectures are discussed in Section 7.

2.4 Poor Convergence Properties

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 $t \to \infty$ [9]. 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 sessions. Using MATLAB, we first examine how DUMP behaves with a constant step size under greedy flows with no stochastics. In the problem formulation, we have implicitly assumed that the flows are long lived and links do not fail. The detailed simulation set-up will be described in Section 3.2 and the experiments in this section use an access-core topology (Figure 3a).

In Figure 1, we plot the aggregate user utility against the number of iterations for three different step sizes. Each iteration corresponds to the maximum RTT in the network, so on the order of $100ms$. We observe that, when the step size is too large, such as in Figure 1a, DUMP will constantly overshoot or undershoot, never reaching the ideal utility. On the other hand, when the step size is too small, such as in Figure 1c, DUMP converges very slowly. A good fit is shown in Figure 1b, where DUMP does converge after about 100 iterations. This highlights that choosing an appropriate step size is challenging. The paper [7] uses the same objective as DUMP and their simulations on much simpler topologies show the same convergence sensitivity.

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 very fine-grained. This corresponds to the current congestion control mechanism where sources only reduce their sending rates once packets are already lost, causing the saw tooth behavior which we observe. In fact, the feedback price in DUMP has the same formulation as the congestion price in [6].

Another reason for not maximizing the aggregate utility was proposed in [10]. The authors of [10] suggest the network would be driven to a solution where some links are operating near capacity. This is an undesirable operating point which is very fragile to traffic bursts. This suggests that maximizing the aggregate user utility enhances performance of the individual users in the short term, but leaves the network as a whole fragile.

2.5 New Objective for Traffic Management

In order to avoid the poor convergence properties seen in DUMP, we look for an alternative problem formulation which takes into account the operator's objective as well. In today's traffic-engineering practices, the following optimization problem is solved with only $\mathbf{R}$ is a variable (and $\mathbf{x}$ constant):

$$\text{minimize} \quad \textstyle\sum_l f(\sum_i R_{li} x_i / c_l). \tag{4}$$

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, this roughly models M/M/1 queuing delay. Second, network operators want to penalize solutions with many links at or near capacity and don't care too much whether a link is 20% loaded or 40% loaded [1, 2]. If we solve (4) with both $\mathbf{x}$ and $\mathbf{R}$ as variables, then the solution would end up with no one sending any traffic, 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 $(\mathbf{x}, \mathbf{R})$:

$$\begin{array}{ll} \text{maximize} & \sum_i U_i(x_i) - w \sum_l f(\sum_i R_{li} x_i / c_l) \\ \text{subject to} & \mathbf{Rx} \preceq \mathbf{c},\ \mathbf{x} \succeq \mathbf{0}. \end{array} \tag{5}$$

This objective favors a solution that provides 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, then (5) is very close to (1) since the utility term dominates. When w is large, the solution would be more conservative in avoiding solutions which are close to capacity. Operators can tune w to operate their network with aggressive or conservative sending rates. Architecturally, the operator is responsible for setting U, f, w and also any tunable algorithmic parameters that may come out of the decomposition process.

The number of non-degenerate alternative decompositions is dependent on the complexity of the objective function and the number of constraints. Since (1) is simpler, it only had two different known decompositions (the other one with equally poor convergence behavior), whereas (5) has four.

3. MULTIPLE DECOMPOSITIONS

In this section, we describe the distributed algorithms generated from all known optimization decompositions of (5), [4, 5]. As in Section 2.1, we first transform (5) to a convex optimization problem using $\mathbf{z}$ and $\mathbf{H}$:

$$\begin{array}{ll} \text{maximize} & \sum_i U_i(\sum_j z_j^i) - w \sum_l f(y_l/c_l) \\ \text{subject to} & \mathbf{y} \preceq \mathbf{c}, \\ & y_l = \sum_i \sum_j H_{lj}^i z_j^i, \ \ \forall l. \end{array} \tag{6}$$

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.

In all four algorithms, the resulting solutions update their path rates based on feedback prices from links. Optimization decomposition leads us to three general notions that make the algorithms effective:

- *Effective capacity* (y_l) arises out of decoupling the cost function f in the objective from U. Its is to provide feedback before link load exceed actual capacity.
- *Consistency price* arises from relaxing the constraint $\mathbf{y} \preceq \mathbf{c}$. It ensures that effective capacity is below actual capacity at the equilibrium point.
- *Direct path-rate update* is possible when the capacity constraint is relaxed with a penalty function.

There are several 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 local optimization and 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 constant per source.

Decomposition	Tunable Parameters	Storage Per Link	Storage Per Source
Partial Dual	1	1	0
Primal Dual	3	1	1
Full Dual	2	2	0
Primal Driven	2	0	1

Table 2: Summary of tunable parameters and storage overhead.

The four algorithms do differ in the number of tunable parameters and the amount of memory required, as summarized in Table 2. The primal-dual algorithm contains the most number of tunable parameters while partial-dual algorithm contains the fewest. In terms of memory consumption, any iterative subgradient update requires storage of the previous value. The primal-driven algorithm consumes the least memory as there are fewer sources than links in any connected graph.

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 2. Similar to Section 2.2, a single dual variable $\mathbf{s}$ is introduced to relax the constraint $y_l = \sum_{i,j} H_{lj}^i z_j^i$. The feedback price is updated as follows:

$$s_l(t+T_p) = s_l(t) - \beta_s \left(y_l(t) - \sum_i \sum_j H_{lj}^i z_j^i(t) \right). \tag{7}$$

In considering constant step sizes in this section, we remove the t argument from all the step sizes. Equation (7) is identical to Figure 2, except for using the effective capacity rather than actual capacity. This is one way to provide feedback to the source before the links reach actual capacity.

3.1.1 Local Optimization: Partial-dual

The derivation process for the **partial-dual** algorithm is identical to Section 2.2 except with *effective capacity* $\mathbf{y}$ as an additional primal variable. The constraint $\mathbf{y} \preceq \mathbf{c}$ is directly enforced resulting in the following effective capacity update:

$$y_l(t+T_p) = \text{minimize}_{(y_l \le c_l)} w f(y_l/c_l) - s_l(t) y_l. \tag{8}$$

In (8) 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). Note that the effect of the cost function is proportional to the size of w.

3.1.2 *Subgradient Update: Primal-Dual*

The **primal-dual** decomposition first decomposes (6) into two layers, one responsible for each primal variable. The master problem solves for $\mathbf{y}$ assuming a given $\mathbf{x}^*$, while the subproblem that solves for $\mathbf{x}$ assuming a fixed $\mathbf{y}$. The master problem is as follows:

$$\begin{array}{ll}\text{maximize} & \sum_i U_i(\mathbf{x}^*) - w\sum_l f(y_l/c_l) \\ \text{subject to} & \mathbf{y} \preceq \mathbf{c}.\end{array} \tag{9}$$

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

$$\begin{array}{ll}\text{maximize} & \sum_i U_i(x_i) \\ \text{subject to} & \mathbf{Rx} \preceq \mathbf{y}.\end{array} \tag{10}$$

Note that (10) 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 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 :

$$y_l(t+T_m) = \min(c_l, y_l(t)+\beta_y(s_l(t)-wf'(y_l(t)))) \tag{11}$$

where β_y is the effective capacity step size. Taking a closer look at (11), the minimization ensures effective capacity stays below the actual capacity. The parameter T_m is a multiple of T_p since (9) is updated less frequently than (10). 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 (11) 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:

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

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:

$$y_l(t+T_p) = \text{minimize}_{y_l} wf(y_l/c_l) - (s_l(t)+p_l(t))y_l.$$

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

3.3 Direct Path Rate Update: Primal Driven

In all the previous algorithms, auxiliary dual variables were introduced to relax the constraints. In this **primal-driven** decomposition, we find a direct solution by introducing a penalty function. Let the penalty function $g_l(\sum_i\sum_j H^i_{lj}z^i_j)$ replace the capacity constraint $\mathbf{Hz} \preceq \mathbf{c}$. 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 [12]. If we combine g and the cost function f to get a penalty-cost function $P_l(\sum_i\sum_j H^i_{lj}z^i_j)$, then (6) can be transformed into the following:

$$\text{maximize} \sum_i U_i(\sum_j z^i_j) - w\sum_l P_l(\sum_i\sum_j H^i_{lj}z^i_j). \tag{12}$$

The derivative (12) is:

$$\frac{dz_i}{dt} = \beta_z\frac{\partial U_i}{\partial z^i_j}(x_i(t)) - w\sum_l P_l'(\sum_i\sum_j H^i_{lj}z^i_j(t))). \tag{13}$$

Converting (13) into a subgradient update form and separate link information from source information, then we obtain the algorithm in Figure 4.

Path rate update:

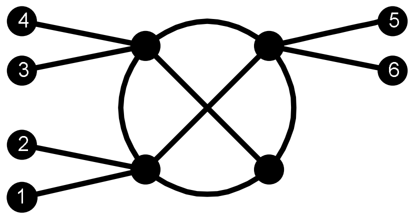
$$z^i_j(t+T_p) = z^i_j(t) + \beta_z z^i_j(t)(\frac{\partial U_i}{\partial z^i_j}(x_i(t)) - \sum_l H^i_{lj}s_l(t))$$

Feedback price update:

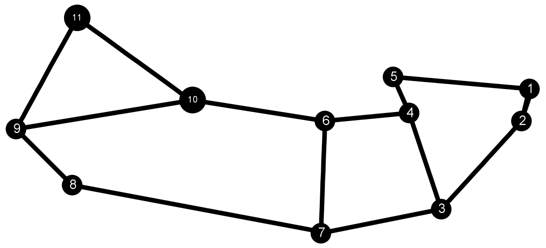
$$s_l(t+T_p) = wP_l'(\sum_i\sum_j H^i_{lj}z^i_j(t)),$$

Figure 4: The Primal-driven 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



(a) Access-Core topology



(b) Abilene topology

Figure 3: Two realistic topologies.

function is changing at a given link load. The primal-driven algorithm in Figure 4 differs significantly from the first three decompositions in two ways. First, it is the only algorithm with 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 convergence rate and aggregate utility achieved. Second, we find algorithms which use local minimizations instead of iterative updates converge faster. Third, we find consistency price can aid convergence, but only for small w.

4.1 Experimental Set-up

We chose MATLAB as our simulation environment since we are proposing a new traffic management system and do not need models of existing congestion control or routing protocols. We do consider stochastic perturbations and feedback delay in our simulations. We present simulation with topology changes and traffic stochastics in Section 6. For all algorithms, we update the source and link variables at each iteration based on link load from the previous iteration. Each iteration represents the total time it takes for all sources to receive feedback from all links, this is on the order of 100ms. Since all four algorithms have the same feedback mechanism, it is likely that they will all be impacted equally by variabilities in feedback delay.

For the utility function U, we use a logarithmic function commonly associated with proportional fairness and TCP Reno today [13]. 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, 2].

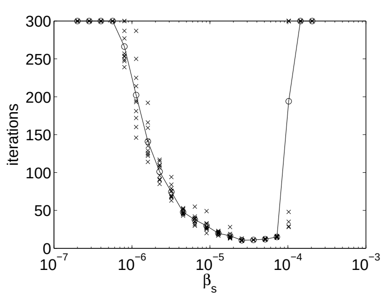
We study two realistic topologies as shown in Figure 3. On the left is a tree-mesh topology, which is representative of a common access-core network structure. On the right is the Abilene backbone network [14]. For tractability of manual setting of paths between two nodes, we choose six source-destination pairs for access core and four pairs for Abilene to be active. For each source-destination pair, we choose three minimum-hop paths as possible paths for access-core and the four minimum-hop paths as possible paths for Abilene. In reality, the Abilene topology has a completely homogenous capacity distribution, but we wanted to simulate the potential variability in real networks. Therefore, 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 (5). 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.

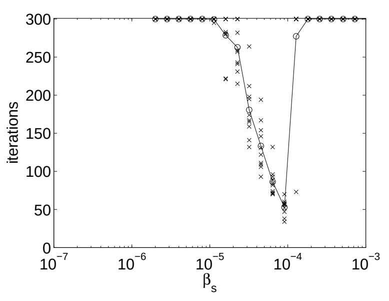
4.2 Weighing User Utility and Operator Cost

In this section, we illustrate *a trade-off between aggregate utility and convergence time*. In Figure 5, 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 value, 10 random capacity distributions are chosen and the average number of iterations before convergence is highlighted in a solid line. Comparing across Figure 5 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.

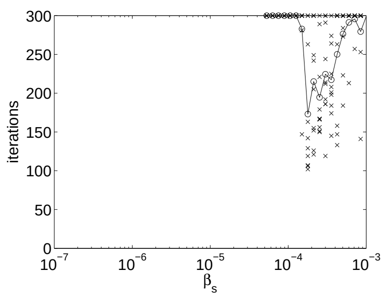
In Figure 7, we plot the percentage of maximal aggregate utility achieved by maximizing the new objective in (5) for a range of w values. From the graph, we observe that there is a knee region for both topologies. For the access-core topology, this knee region is from $w = 1/4$ to $w = 1/6$; for the Abilene topology, this knee region is between $w = 1/6$ and $w = 1/10$. Below this knee region, the algorithm achieves nearly 100% of maximal aggregate utility, since the cost function f is weighed sufficiently lightly to not change the solution. Above this knee region, the percentage of maximal aggregate



(a) $w = 1$

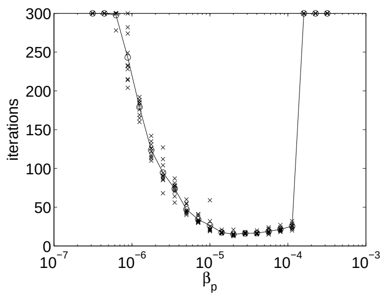


(b) $w = 1/6$

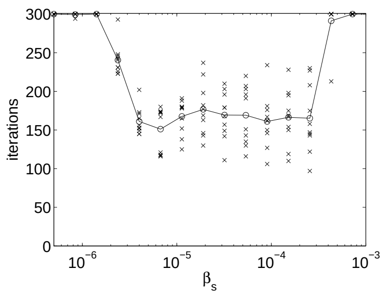


(c) $w = 1/36$

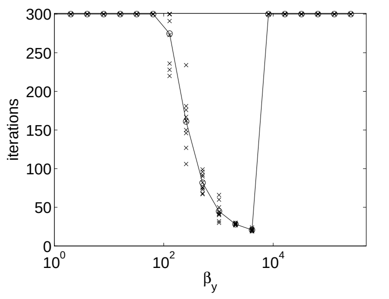
Figure 5: Plots of partial-dual algorithm showing dependence of convergence time on step-size. Access-core topology.



(a) $w = 1, \beta_y = 5000$



(b) $w = 1, \beta_y = 500$



(c) $w = 1, \beta_s = 0.0005$

Figure 6: Plots of primal-dual algorithm showing dependence of convergence time on step-size. Access-core topology.

utility decreases, as f start to impose new constraints beyond the capacity constraints.

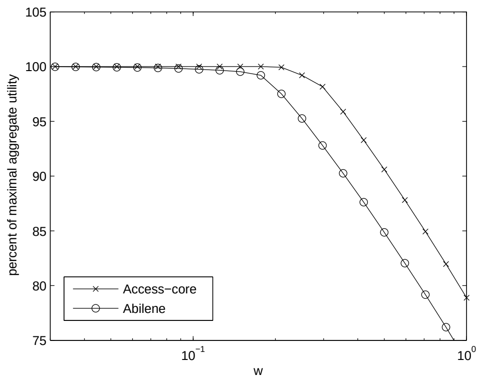


Figure 7: Plot of w versus percentage of maximal utility achieved.

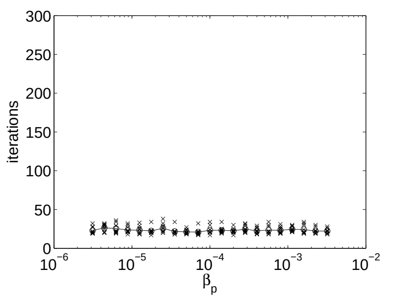
Below the knee region, the gains in aggregate utility does not offset the gain in convergence time. Otherwise, there is a trade-off between aggregate utility and convergence time. Depending on the conditions of their network, operators can choose their desired operation point.

4.3 Comparing Between the Algorithms

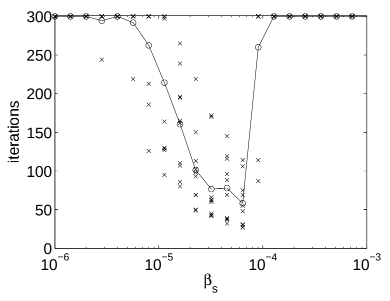
In this subsection, we do a series of comparisons between convergence time of different algorithms, and find partial-dual in Figure 5 is the best overall, with a good convergence profile and fewest tunable parameters.

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.* In Figure 6, we plot convergence time versus step-size for the primal-dual with $T_m = 3T_p$. Since the primal-dual algorithm contains two step sizes (β_y, β_s), the plots in Figure 6 shows convergence time as a function of one step size for a particular value of the other step size. Comparing Figure 6a to Figure 5a, the convergence times of primal-dual algorithm and partial-dual algorithms are almost identical for well-chosen β_y and T_m. For other values of β_y, however, we find the primal-dual algorithm converges more slowly than the partial-dual algorithm, as seen by comparing Figure 6b to Figure 5a. Figure 6c further highlights the convergence properties are quite sensitive to β_y.

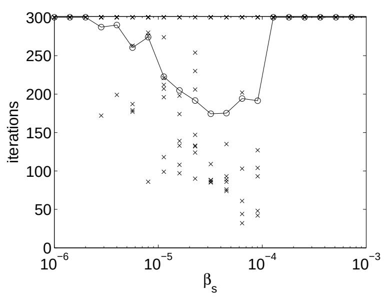
Comparing the full-dual algorithm in Section 3.2 to



(a) $w = 1, \beta_s = 0.000064$, full-dual



(b) $w = 1/6, \beta_p = 0.0001$, full-dual



(c) $w = 1/6$, partial-dual

Figure 8: Plots of full-dual and partial-dual algorithm showing dependence of convergence time on step-size. Abilene topology.

the partial-dual algorithm, we find *consistency price can improve convergence properties*. We plot convergence time versus step size for the full-dual algorithm in Figure 8a and 8b. If we look at Figure 8a, 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 7), we find that the full-dual algorithm converges faster than the partial-dual algorithm, as seen by comparing Figure 8b to Figure 8c. This is because if we allow the capacity constraint to be violated during transients, the algorithm can take more aggressive steps and potentially converge faster.

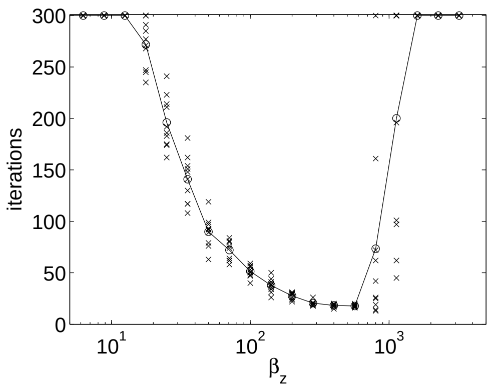


Figure 9: Plot of primal-driven algorithm showing dependence of convergence time on step-size for $w = 1$. Access-core topology.

Comparing the primal-driven algorithm in Section 3.3 to the partial-dual algorithm, we find *local minimization update has better convergence properties than subgradient update.* We plot convergence time versus step size in Figure 9 for the primal-driven algorithm. Compared to Figure 5a, the primal-driven algorithm takes longer to converge at the optimal step size (25 iterations versus 15 iterations) and the convergence time is more sensitive to the choice of step size. In addition, the primal-driven algorithm also requires operators to tune a second parameter (the penalty function g).

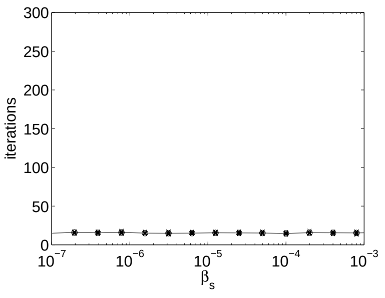
5. TRUMP

While the algorithms introduced in Section 3 converge faster than DUMP for well-tuned parameters, they still require explicit feedback. In this section, we introduce a simpler Traffic-management Using Multipath Protocol (TRUMP) which can be implemented using only *implicit* feedback at the sources.

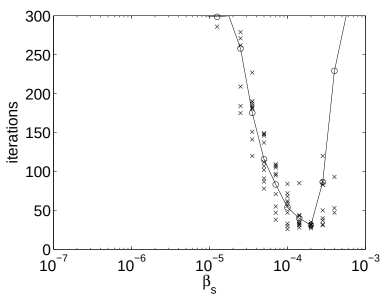
5.1 The TRUMP Algorithm

Our simulations in the previous section suggested 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 best parts of all four algorithms* to construct the TRaffic-management Using Multipath Protocol (TRUMP) described in Figure 11. 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-driven* algorithm in Figure 4 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 prices p_l and q_l also have intuitive interpretations. A closer look at p_l reveals that it is closely related to packet loss: the portion $c_l - \sum_i \sum_j H^i_{lj} z^i_j(t)$ is the amount of overflow on a link, β_p moderates how much to react to the overflow and how much to rely on its old value. If we use the most common interpretation of f *i.e.* queuing delay, then q_l summed along a path is just the sum of the rate of change of queuing delay. This



(a) $w = 1$



(b) $w = 1/6$

Figure 10: Plots of TRUMP algorithm showing dependence of convergence time on step-size. Access-core topology.

Feedback price update:

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

Loss price update:

$$p_l(t+T_p) = [p_l(t) - \beta_p(c_l - \sum_i \sum_j H^i_{lj} z^i_j(t))]^+,$$

Delay price update:

$$q_l(t+T_p) = wf'\left(\sum_i \sum_j H^i_{lj} z^i_j(t)\right),$$

Path rate update:

$$z^i_j(t+T_p) = \text{maximize}_{z^i_j} U_i(\sum_j z^i_j) - \sum_l (s_l(t)) \sum_j H^i_{lj} z^i_j$$

Figure 11: TRaffic-management Using Multipath Protocol.

is captured as difference in Round Trip Times (RTTs). Current congestion control reacts mostly to packet loss [6], which tends to cause the saw-toothed windowing behavior seen today. Proposed TCP variants primarily react to queuing delay so that feedback regarding the link conditions can be received before packets are lost *e.g.* [3, 15, 16]. TRUMP is similar to Compound TCP [17] which reacts to both delay and loss, though TRUMP also handles routing. The intuition is that queuing delay warns TRUMP when a link is getting heavily loaded to avoid packet loss, but if a packet is lost, TRUMP will take that into consideration. Algorithmically, TRUMP has the same measurement and computational overhead as the four algorithms in Section 3. It requires just one variable to be stored per link and has only one tunable parameter. If TRUMP is implemented based on implicit feedback, then the measurement, computational and storage at the links are no longer required.

5.2 TRUMP Convergence Properties

In all our simulations, we find that it indeed converges to the optimal values of (5) across a large range of w values, both topologies and different variations in link capacity. So when we plot its achieved aggregate utility at equilibrium versus w, we would get an identical plot to Figure 7. While we believe that a convergence proof is important, it is perhaps even more important to trade-off the simplicity and practical properties of TRUMP with the lack of a convergence proof.

In Figure 10, we plot convergence time versus step-size for TRUMP. From Figure 10a, we observe that as with the full-dual algorithm, the second link price does not play a role when $w = 1$. Comparing Figure 10b to Figure 5b, we see that TRUMP has much nicer convergence properties than the partial-dual algorithm. Overall, TRUMP is simpler than any of the algorithms presented in Section 3, with just one tunable parameter that only needs to be tuned if the operator wants to operate his network with a small w.

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. We consider the region where w is sufficiently large for $p = 0$ (as seen in Figure 10a), and find a contraction mapping on z. Overall, TRUMP is simpler than any of the algorithms presented in Section 3, with only one tunable parameter that only needs to be tuned for small w.

6. PERFORMANCE UNDER STOCHASTICS

In our earlier simulations, we had implicitly assumed flows are persistent and sources are greedy. In this section, we relax those assumptions by studying how TRUMP responds under topology changes and traffic shifts. Since our goal is to understand TRUMP's behavior under stochastics, we do not explicitly model retransmissions of lost packets, just like early studies of TCP protocols such as [15, 16]. Instead we penalize the utility achieved when links are overloaded to ensure that utility is not given to packets that would ultimately be dropped. For each path rate, we calculate the maximum relative load overflow of links along the path $((y_l - c_l)/y_l)$. Then we penalize all flows using that link proportionally to the amount of load they contribute to that link. We assume the excess packets are loading the entire path since a downstream link could be a bottleneck link for a different path. Due to space constraints, we only present the results for access-core topology as the same trends are observed for Abilene. For legibility of the graphs, we only compare TRUMP to DUMP, but TRUMP also outperforms the algorithms in Section 3.

6.1 Topology Dynamics

In our first set of experiments, we consider what happens when a link fails. Link failures occur infrequently, but can be quite disruptive. We start with all links operational, then we let a random link fail or recover every 50 iterations. We ensure that even with a failed link there is still at least one available path between each source-destination pair.

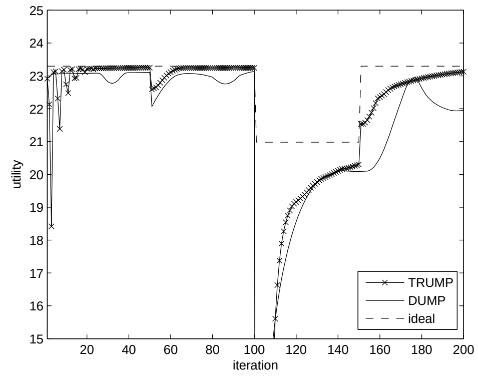


Figure 12: Plot of utility versus iteration number for $w = 1/6$. Links fail or recover every 50 iterations.

In Figure 12, we plot the aggregate utility versus time for TRUMP, DUMP and also the ideal as a benchmark. First of all, we observe that link failures can be quite disruptive. For example, at iteration 100, we see a large drop in utility for both DUMP and TRUMP. This is because a source is sending all its traffic on a single path that is no longer available due to a link failure. This means the maximum relative load overflow for that path is now 1, hence causing the utility to drop to 0 (not shown due to scale) until the traffic is moved onto a new path. Second, we note that since when a link fails or recovers the underlying set of possible paths changes and hence this is similar to just starting from a set of new initial conditions and then moving towards convergence. We observe that since TRUMP converges faster than DUMP, it also recovers faster in this case.

6.2 Traffic Dynamics

In our next set of experiments, we allow flows to arrive according to a Poisson process with rate P. For each instance of the experiment, we consider a constant flow duration d. We then repeat the experiments with different values of d to see how TRUMP reacts to flow duration. We let $Pd = 0.2$ in order to place the same average traffic load on the network.

From Figure 11, where we describe TRUMP algorithmically, we see that TRUMP adapts its path rates based on link prices of the previous iteration (which is represented by the link load placed on the system when flows arrive or depart). We take the aggressive approach of starting new flows which arrive between two iterations at the same rate as all other flows between the same source-destination pair.

In Figure 13, we plot aggregate utility versus iteration number for TRUMP, DUMP and the ideal. We observe that with $w = 1/6$, the gap between TRUMP and ideal is very small. Similar to the link failure case, we see that once again TRUMP trumps DUMP.

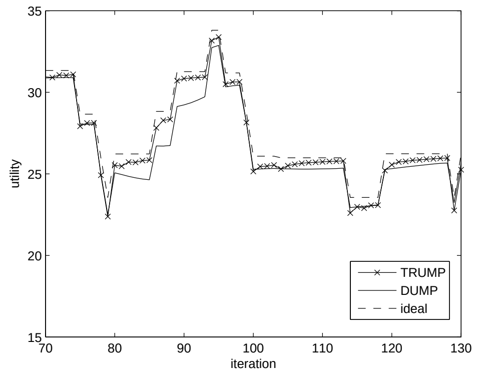


Figure 13: Plot of utility versus iteration number for $w = 1/6$. Poisson arrival with probability 0.02, flow duration of 10 iterations.

In Figure 14, we sweep a set of flow durations for TRUMP alone. We run 10 experiments for the same flow duration, each experiment for 20 times the flow duration. We calculate the average and also the standard deviation for the 10 samples associated with each duration and plot the results in Figure 14. We observe that there is very little dependency between flow duration and gap to maximal utility. There are two reasons for this observation. One, TRUMP only cares about how many flows there are between each source-destination pair, not their flow lengths when assigning path rates for a source-destination pair. Two, by assigning the same rate to all flows of a source-destination pair, TRUMP emulates processor sharing like the proposed Rate Control Protocol (RCP) [16], and hence deals equally well with long and short flows. Since TRUMP spans congestion control and routing, it has more degrees of freedom than RCP which is limited to congestion control. As a result, RCP assigns the same flow rate to all flows passing through a link (local effect), while TRUMP assigns the same flow rate to flows that have the same source-destination pair (end-to-end effect). Our preliminary results show that TRUMP also works well with a mix of short and long flows.

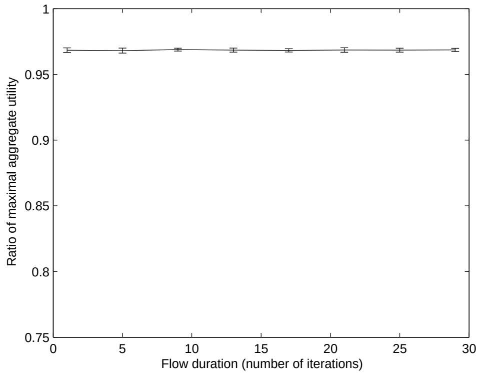


Figure 14: Plot of ratio of maximal aggregate utility versus flow duration for $w = 1/6$, TRUMP.

As true for any traffic management protocol (proposed or current), it is difficult for TRUMP to adapt immediately when there is a sudden burst in traffic. A conservative approach of starting new flows at very low sending rate can waste bandwidth, while an aggressive approach as we model here can overload the network and cause losses. When we test TRUMP under the Weibull arrival rates which contains more traffic bursts, we find that when multiple flows arrive at once at iteration 71, there is a bigger gap to the ideal aggregate utility due to overloading of some links. Still TRUMP's fast reaction time is an advantage for handling shifts in traffic.

7. ARCHITECTURAL IMPLICATIONS

Architecturally, we find that the algorithms specified by the mathematics still leave many questions unanswered such as: which network elements correspond to the sources, perform computations and determine **H**. This is actually a blessing as Traffic Management is only part of network architecture, so it is good to have some flexibility. In this section, we explore how TRUMP can fit into different architectures.

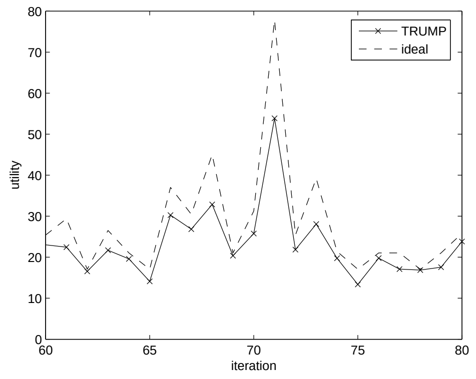


Figure 15: Plot of aggregate utility versus iteration number for $w = 1/6$. Weibull arrivals with parameter 0.5, flow duration of 1.

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 abide by the rate limits. If end hosts are unaware of the multiple paths, then edge routers could send end hosts an aggregate rate over all paths instead of feedback prices from the links. In this case, the edge routers would split traffic amongst the multiple paths. Even if the end hosts are aware of multiple paths, they might not be trusted to not send too aggressively. In this case, edge routers should perform the same calculations and perform policing or shaping to enforce the path rate by dropping excess packets. In terms of computational impact, if sources are end hosts, then the computational load of updating path rates is distributed across end systems and is scalable. If sources are edge routers, then they should keep state on traffic aggregates to avoid overloading the edge routers with flow-level state.

Are the computations done centrally or distributedly? In the algorithms, it is unclear which component takes the link feedback and computes the resulting path rates. One natural division of labor would be to have the links compute their prices, then feed back the appropriate price to the sources, the sources would then compute the path rates. An alternative solution is for sources to periodically probe along the path and sum up the price from link to link, similar to the approach in [18]. Another option is for the management system to centrally collect the link load measurements, calculate the feedback prices and the path rates. Then the edge routers would receive the path rate information and use it to police the incoming traffic.

Is H determined by the routers or the management system? In the algorithms, the matrix **H**

is considered to be a constant, but there is no hint as to where it comes from. In reality, **H** can be computed by routers or set manually by the network-management system. **H** is a chosen subset of all possible paths. This subset could be the shortest paths (determined by hop-count, RTT or link weights), the K-shortest paths or all possible paths. The number of paths introduces a trade-off between flexibility of distributing the traffic and the overhead of computing and using the routes. One way to find **H** is through an underlying routing protocol, *e.g.*, where routers compute all of the shortest paths, or the K shortest paths. It is also possible for the management system to pin a set of tunnels between each source-destination pair centrally, similar to the approach in [18]. For scalability, this can just be the ingress-egress pairs or pairs between points-of-presence.

Is the feedback explicit or implicit? While the mathematics suggest there is explicit feedback from the links to the sources for all algorithms, TRUMP can be implemented based on implicit feedback from the links. As we mentioned in Section 5.1, the feedback price has two components, which can be interpreted as packet loss and RTT changes, respectively. This interpretation has the added benefit of not requiring any explicit feedback.

What are the challenges for Multi-AS? Although most of the earlier discussion implicitly assumes all of the links and nodes belong to a single institution, many of our results are relevant to a network like the Internet that consists of many Autonomous Systems (ASes). Still, the multi-AS scenario does present some unique challenges for deploying and running the proposed traffic-management protocols. For example, the ASes would need to agree to provide explicit feedback from the links to the end hosts or edge routers, and trust that the feedback is an honest reflection of network conditions. This makes the use of implicit feedback, based on observations of packet losses and changes in RTTs, especially attractive for a multi-AS network. In addition, the sources would need the flexibility to direct traffic over multiple paths. Although already possible today through multi-homing, extending the Border Gateway Protocol (BGP) to a multipath protocol as proposed by [19], would give sources additional flexibility. Combining implicit feedback with a multipath interdomain routing protocol for TRUMP would be an exciting avenue for future research.

8. RELATED WORK

Mathematics has been used to aid research on traffic management in four ways: analysis of existing protocols *e.g.* [5, 6], tuning configuration parameters of existing protocols *e.g.* [1, 2], analysis of proposed protocols *e.g.* [15, 16] and guiding the design of new protocols *e.g.* [3]. In this section, we will focus our attention on the body of research that looks at new protocol design. Most of the proposed new traffic management protocols only consider congestion control or traffic engineering alone. Design of congestion control has been an extremely active area of research, so we will only mention a small selection of papers which share similarities with our work. FAST TCP [3] also uses optimization theory to guide protocol design, but it only considers one single decomposition. Many of the other protocols, such as XCP [15], RCP [16] and Compound TCP [17], prove the stability of their algorithms using control theory, but do not use theory to guide their design process from the beginning. Similarly, mathematics has helped to analyze dynamic distributed traffic engineering protocols that do not consider congestion control, [18, 20, 21]. In particular, tools from control theory have been used to analyze the stability of [18, 20], and [21] was developed using algorithms from game theory. These traffic-engineering protocols do not start with an objective function as we do for TRUMP and only considers part of the traffic management problem.

Some research tries to look at traffic management as a whole, but still has a mathematical flavor. Such research either tries to analyze the current architecture or propose new architectures. Our work is quite different from [10, 22], which attempt to analyze whether congestion-control and traffic-engineering practices today interact well together. Some of the research proposed traffic-management protocols with poor practical properties. For example, [8] proposes a system where link weights are based on congestion price and then traffic is routed on the path with lowest link weights, such a system turns out to be unstable. Yet others analyze stability of joint congestion control and routing algorithms using control theory, *e.g.* [7, 11, 23]. The paper [7] considers a different objective function than TRUMP. [23] is an analysis of whether joint-congestion control and routing algorithms can be stable. In [11], a similar objective function is considered, but they advocate a very different architecture involving overlays. Overall, TRUMP uses a more complete design process starting with justifying the objective function to comparing between multiple decompositions to evaluating performance under stochastic perturbations.

Our work is the most inspired by [4, 10, 7, 24], though they did not try to bridge the gap between mathematics and protocol design. In [4], multiple decomposition methods were presented, but not applied to the design of new approaches to traffic management, and no machinery was offered to compare between the multiple decompositions. The work in [10] briefly describes an algorithm similar to the full-dual algorithm in Section 3.2, but had very limited evaluation of the algorithm. In the Appendix of [24], an algorithm similar to the primal-driven algorithm we described in Section 3.3 was presented, but not evaluated. Neither [10] nor [24] considered a range of design alternatives, or tried to compare between them, as we have in this paper.

9. CONCLUSIONS

In this paper, we searched for a traffic-management protocol which was distributed, adaptive, robust, flexible and easy to manage. We followed a top-down design process which started with finding the correct problem formulation. We generated four provably optimal distributed solutions using known decompositions. We then compared the practical properties of four different distributed solutions through simulation. We gleaned several insights regarding how algorithmic structure can affect the rate of convergence and sensitivity to step size. Using those insights, we combined the best parts of each algorithm to construct TRUMP: a new and simpler traffic management protocol. By adapting to both packet loss and queuing delay, TRUMP is effective in reacting to topology changes and traffic shifts on a small timescale. By emulating processor sharing for flows traversing the same source-destination pair, TRUMP works well with short and long flows. TRUMP is flexible and fits into a range of architectures. In addition, TRUMP is easy to manage, with just one optional tunable parameter. The process of considering multiple decompositions has led us to a more practical, though heuristic, protocol. This suggests that while theory can be a good guide, better protocols may be found by looking past the boundaries of theory.

Now that we have constructed a promising protocol, we plan to follow up with packet-level simulations. Using NS-2, we can test how well TRUMP behaves using only implicit feedback. We can also test how it behaves under more realistic workload models as in [25] and variable feedback delay. We are optimistic TRUMP will perform well under heterogeneous feedback delay, since [23] establishes a stable joint congestion control and routing protocol is achievable at the timescale of RTTs. We plan to add in practical details such as handling retransmission of lost packets at this stage. Another important question is how to achieve perfect splitting of the traffic. Flow-level splitting is not quite accurate, but packet-level splitting can cause out-of-order packets. Recent work on flowlet switching shows promise as a way to get near perfect traffic splitting ratios without reordered packets [26]. To make TRUMP truly scalable for a large network, we must also explore practical details such as the level of traffic aggregation and how to provision buffers properly. In our ongoing work, we have planned a series of experiments to complete the whole design cycle from mathematics to a full-fledged protocol.

10. REFERENCES

[1] 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.

[2] J. Rexford, "Route optimization in IP networks," in *Handbook of Optimization in Telecommunications*, Springer Science + Business Media, February 2006.

[3] D. X. Wei, C. Jin, S. H. Low, and S. Hegde, "FAST TCP: Motivation, architecture, algorithms, performance," *IEEE/ACM Trans. Networking*, December 2006.

[4] 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.

[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] X. Lin and N. B. Shroff, "Utility Maximization for Communication Networks with Multi-path Routing," *IEEE Trans. Automatic Control*, vol. 51, May 2006.

[8] 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.

[9] D. P. Bersekas, *Nonlinear Programming*. Athena Scientific, second ed., 1999.

[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] D. P. Bersekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Athena Scientific, second ed., 1997.

[13] 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.

[14] Abilene Backbone. `http://abilene.internet2.edu/`.

[15] D. Katabi, M. Handley, and C. Rohrs, "Congestion Control for High Bandwidth-Delay Product Networks," in *Proc. ACM SIGCOMM*, August 2002.

[16] 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/`.

[17] 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.

[18] S. Kandula, D. Katabi, B. Davie, and A. Charny, "Walking the Tightrope: Responsive Yet Stable Traffic Engineering," in *Proc. ACM SIGCOMM*, August 2005.

[19] W. Xu and J. Rexford, "MIRO: Multipath Interdomain ROuting," in *Proc. ACM SIGCOMM*, August 2006.

[20] A. Elwalid, C. Jin, S. H. Low, and I. Widjaja, "MATE: MPLS Adaptive Traffic Engineering," in *Proc. IEEE INFOCOM*, April 2001.

[21] S. Fischer, N. Kammenhuber, and A. Feldmann, "REPLEX — Dynamic Traffic Engineering Based on Wardrop Routing Policies," in *Proc. CoNEXT*, December 2006.

[22] 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.

[23] 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.

[24] R. J. Gibben and F. Kelly, "On packet marking at priority queues," *IEEE Trans. Automatic Control*, vol. 47,

pp. 1016–1020, December 2002.

[25] P. Barford and M. Crovella, "Generating representative web workloads for network and server performance evaluation," in *Proc. ACM SIGMETRICS*, pp. 151–160, June 1998.

[26] S. Sinha, S. Kandula, and D. Katabi, "Harnessing TCP's Burstiness with Flowlet Switching," in *Proc. SIGCOMM Workshop on Hot Topics in Networking*, December 2004.

APPENDIX

A. CONVERGENCE OF TRUMP

Background: A particular family of widely-used utility functions is parameterized by $\alpha \geq 0$ [13]:

$$U_\alpha(x) = \begin{cases} \log x, & \alpha = 1 \\ (1-\alpha)^{-1} x^{1-\alpha}, & \alpha \neq 1. \end{cases} \tag{14}$$

Maximizing these α-fair utilities over linear flow constraints leads to rate-allocation vectors that satisfy the definitions of α-fairness in the economics literature. The notion of α-fairness from [13] led to many TCP variants with different α-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_α as $\alpha \to \infty$ in the single-link case.

Theorem 1. *TRUMP converges to the optimal value of (5) under the following conditions:*

1. $p = 0$
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 α refers to α-fair utility [13].

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

$$z_j^i = U_i'^{-1}\Big(\sum_l H_{lj}^i f_l'\big(\sum_{i,j} z_j^i H_{lj}^i / c_l\big)/c_l\Big).$$

We look for a *contraction mapping* for z as outlined in [12]. First we compute the Jacobian for z_j^i:

$$\begin{aligned} J_{ij,st} = \quad & (U_i'^{-1})'\big(\textstyle\sum_l H_{lj}^i f_l'(\sum_{i,j} z_j^i H_{lj}^i/c_l)/c_l\big) \\ & \big(\textstyle\sum_l H_{lj}^i \sum_{s,t} H_{lt}^s f_l''(\sum_{i,j} z_j^i H_{lj}^i/c_l)/(c_l)^2\big). \end{aligned}$$

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

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

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

$$||J||_\infty = \max_{ij} (U_i'^{-1})'\Big(\sum_l H_{lj}^i f_l'(u_l)\Big)\Big(\sum_l H_{lj}^i n_l f_l''(u_l)\Big).$$

For convergence of z, $||J||_\infty < 1$ is a *sufficient* condition. For α-utility, we have $(U_i'^{-1})' = -\frac{1}{\alpha} x^{-1/\alpha - 1}$ for $\alpha > 1$. For $U = \log(x), \alpha = 1$, $(U_i'^{-1})' = -x^{-2}$, so the same equation holds. So we can rewrite $||J||_\infty$ as:

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

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