Understanding Vegas: a duality model

STEVEN H. LOW
Computer Science and Electrical Engineering, California Institute of Technology, USA
slow@caltech.edu
and
LARRY L. PETERSON and LIMIN WANG
Computer Science, Princeton University, USA
{llp,lmwang}@cs.princeton.edu

We view congestion control as a distributed primal-dual algorithm carried out by sources and links over a network to solve a global optimization problem. We describe a multi-link multi-source model of the TCP Vegas congestion control mechanism. The model provides a fundamental understanding of delay, fairness and loss properties of TCP Vegas. It implies that Vegas stabilizes around a weighted proportionally fair allocation of network capacity when there is sufficient buffering in the network. It clarifies the mechanism through which persistent congestion may arise and its consequences, and suggests how we might use REM active queue management to prevent it. We present simulation results that validate our conclusions.

Categories and Subject Descriptors: C.2.5 [**Computer Systems Organization**]: Computer-communication Networks—*Local and Wide-Area Networks*; C.4 [**Computer Systems Organization**]: Performance of System

General Terms: Performance; Theory; Algorithms

Additional Key Words and Phrases: TCP congestion control, TCP Vegas, persistent congestion, REM

1. INTRODUCTION

1.1 Background

TCP uses *window-based* flow control to pace the transmission of packets. Each source maintains a "window size" variable that limits the maximum number of packets that can be outstanding: transmitted but not yet acknowledged. When a window's worth of data is outstanding the source must wait for an acknowledgment before sending a new packet. Two features of this general strategy are important. First, the algorithm is "self-clocking" meaning that TCP automatically slows down the source when the network becomes congested and acknowledgments are delayed. The second is that the window size variable determines the source rate: roughly one window's worth of packets is sent every round-trip time. This second feature is exploited in Jacobson's paper in 1988 [Jacobson 1988],

---

Partial and preliminary results have appeared in [Low et al. 2001].
The first author acknowledges the support of the Australian Research Council through Grant A49930405, NSF through Grant ANI-0113425, and the Caltech Lee Center for Advanced Networking. The second author acknowledges the support of NSF through Grant ANI-9906704, and DARPA through contract F30602–00–2–0561.

which is based on an earlier idea in [Ramakrishnan and Jain 1990]. Jacobson proposed an additive-increase-multiplicative-decrease algorithm to *dynamically* adapt the window size to network congestion. This algorithm is implemented in TCP Reno, a varient of TCP that includes this algorithm.

TCP Reno consists of three main mechanisms: *slow-start*, *congestion avoidance*, and *fast retransmit/fast recovery* [Jacobson 1988; Stevens 1999; Peterson and Davie 2000]. A source starts cautiously with a small window size of one packet and increments its window by one every time it receives an acknowledgment. This doubles the window every round-trip time and is called slow-start. When the window reaches a threshold, the source enters the congestion avoidance phase, where it increases its window more slowly by the reciprocal of the current window size every time it receives an acknowledgment. This increases the window by one packet in each round-trip time. On detecting a loss through duplicate acknowledgments, the source retransmits the lost packet, halves its window, and re-enters congestion avoidance. This is referred to as fast retransmit/fast recovery, to contrast it with the source detecting the loss through a timeout, in which case it re-enters slow-start instead of congestion avoidance.

TCP Vegas was introduced in 1994 as an alternative to TCP Reno [Brakmo and Peterson 1995]. It improves upon each of the three mechanisms of TCP Reno. The first enhancement is a more prudent way to grow the window size during the initial use of slow-start and leads to fewer losses. The second enhancement is an improved retransmission mechanism where timeout is checked on receiving the first duplicate acknowledgment, rather than waiting for the third duplicate acknowledgment (as Reno would), and leads to a more timely detection of loss. The third enhancement is a new congestion avoidance mechanism that corrects the oscillatory behavior of Reno. In contrast to the Reno algorithm, which induces congestion to learn the available network capacity, a Vegas source anticipates the onset of congestion by monitoring the difference between the rate it is expecting to see and the rate it is actually realizing. Vegas' strategy is to adjust the source's sending rate (window size) in an attempt to keep a small number of packets buffered in the routers along the path.

In this paper, we study the congestion avoidance mechanism of Vegas. It is well-known that file sizes transported over the Internet have heavy-tail. In simple terms this means that while most TCP connections are short ("mice"), most packets are generated by a few long TCP connections ("elephants"). It is these elephants, not mice, that need, and can, be effectively controlled by TCP. As will become clear later, congestion avoidance determines the bandwidth allocation among, and the quality of service experienced by, these elephants.

1.2 Motivation and Outline

Although experimental results presented in [Brakmo and Peterson 1995], and duplicated [Ahn et al. 1995], show that TCP Vegas achieves better throughput and fewer losses than TCP Reno under many scenarios, at least two concerns remained: whether Vegas is stable, and if so, whether it stabilizes to a fair distribution of resources; and whether Vegas results in persistent congestion. In short, Vegas has lacked a theoretical explanation of why it works.

This paper addresses this shortcoming by presenting a model of Vegas as a distributed optimization algorithm. The basic observation is as follows. Congestion control is a distributed algorithm to share network resources among competing sources. It consists of a source algorithm (e.g., Reno, Vegas, etc.), that dynamically adjust source rates based on congestion in their paths, and a link algorithm (e.g., DropTail, RED, REM, etc.), that up-

dates, implicitly or explicitly, a certain congestion measure at each link and feeds it back, implicitly or explicitly, to sources that use this link. Different protocols use different metrics as congestion measure. For example, Reno uses loss probability, and as it turns out, Vegas uses *queueing* delay (see below).

The key idea of the duality model in [Low and Lapsley 1999; Low 2000; Low et al. 2002] is to interpret congestion control as a distributed algorithm carried out by sources and links over the network to solve a global optimization problem, and that different protocols (Reno, Vegas, DropTail, RED, REM, etc.) are different ways to solve the same prototypical problem with different objective functions.

We show in Section 2 that the objective of Vegas is to maximize the aggregate utility of all sources subject to the capacity constraints of the network's resources. Moreover, the congestion avoidance mechanism of Vegas can be interpreted as an approximate gradient projection algorithm to solve the dual problem. This model suggests that Vegas stabilizes around a weighted proportionally fair allocation of network capacity when there is sufficient buffering in the network, that is, when the network has enough buffers to accommodate the extra packets the algorithm strives to keep in the network. If sufficient buffers are not available, equilibrium cannot be reached, and Vegas reverts to Reno. The implications on delay, loss, and fairness are explained in Section 3.

The duality model leads to a new interpretation of the Vegas algorithm, where a source sets its rate to be proportional to the ratio of its propagation delay to queueing delay in its path. Vegas estimates propagation delay by the minimum observed round-trip time. We prove in Section 4 that estimation error distorts Vegas utility function and can lead to persistent queues and unfair rate allocation. We show in Section 5 that by augmenting Vegas with appropriate active queue management (AQM) it is possible to avoid this problem. AQM serves to decouple the buffer process from the feedback required by each Vegas source to determine its optimal sending rate. In Section 6, we present simulation results that both serve to validate the model and to illustrate the effectiveness of this AQM.

Finally, we comment on related work in Section 7, and conclude in Section 8 with limitations of the current work.

2. A MODEL OF VEGAS

This section presents a model of Vegas and shows that 1) the objective of Vegas is to maximize aggregate source utility subject to capacity constraints of network resources, and 2) the Vegas algorithm is a dual method to solve the maximization problem. The goal of this effort is to better understand Vegas' stability, loss and fairness properties, which we discuss in Section 3.

2.1 Preliminaries

A network of routers is modeled by a set L of unidirectional links with transmission capacity c_l, $l \in L$, and infinite buffering space. It is shared by a set S of sources. A source s traverses a subset $L(s) \subseteq L$ of links to the destination, and attains a utility $U_s(x_s)$ when it transmits at rate x_s (e.g., in packets per second). Let d_s be the round-trip propagation delay for source s. For each link l, let $S(l) = \{s \in S \mid l \in L(s)\}$ be the set of sources that uses link l. By definition $l \in L(s)$ if and only if $s \in S(l)$.

According to one interpretation of Vegas, a source monitors the difference between its expected rate and its actual rate, and increments or decrements its window by one in the next round-trip time according to whether the difference is less or greater than a parameter

α_s. If the difference is zero, the window size is unchanged. We model this by a synchronous discrete time system. Let $w_s(t)$ be the window of source s at time t and let $D_s(t)$ be the associated round-trip time (propagation plus queueing delay). We model the change in window size of one packet per round-trip time in reality by a change of $1/D_s(t)$ per discrete time. Thus, source s adjusts its window according to:

Vegas source algorithm:

$$w_s(t+1) \;=\; \begin{cases} w_s(t) + \frac{1}{D_s(t)} & \text{if } \frac{w_s(t)}{d_s} - \frac{w_s(t)}{D_s(t)} < \alpha_s \\ w_s(t) - \frac{1}{D_s(t)} & \text{if } \frac{w_s(t)}{d_s} - \frac{w_s(t)}{D_s(t)} > \alpha_s \\ w_s(t) & \text{else} \end{cases} \tag{1}$$

In the original paper [Brakmo and Peterson 1995], $w_s(t)/d_s$ is referred to as the `Expected` rate, $w_s(t)/D_s$ as the `Actual` rate, and the difference $w_s(t)/d_s - w_s(t)/D_s(t)$ as `DIFF`. The actual implementation estimates the round-trip propagation delay d_s by the minimum round-trip time observed so far. The unit of α_s is, say, KB/s. We will explain the significance of α_s on fairness in Section 3. The algorithm in [Brakmo and Peterson 1995] adjusts window to keep `DIFF` to within α_s and β_s with $\alpha_s < \beta_s$. We assume for simplicity that $\alpha_s = \beta_s$. This captures the essence of Vegas. The effect of using $\alpha_s < \beta_s$ on fairness is discussed in [Boutremans and Boudec 2000].

Let $x_s(t) := w_s(t)/D_s(t)$ denote the bandwidth realized by source s at time t. The window size $w_s(t)$ minus the bandwidth–delay product $d_s x_s(t)$ equals the total backlog buffered in the path of s. Hence, multiplying the conditional in (1) by d_s, we see that a source increments or decrements its window according to whether the total backlog $w_s(t) - d_s x_s(t)$ is smaller or larger than $\alpha_s d_s$. This is a second interpretation of the Vegas algorithm. We will explain a third interpretation in Section 3.1.

Note that (1) only specifies the source dynamics and does not completely describe the network behavior, which also includes link dynamics that determine the round-trip delays $D_s(t)$. The delay $D_s(t)$ depends not only on the window $w_s(t)$ of source s, but also on those of other sources that are coupled through shared links.

2.2 Objective of Vegas

We now interpret the equilibrium of Vegas. Since this does not require the detailed network dynamics, we defer their complete specification to the following subsections (see (16–19) below).

When the algorithm converges, the *equilibrium windows* $w^* = (w_s^*, s \in S)$ and the associated *equilibrium round-trip times* $D^* = (D_s^*, s \in S)$ satisfy

$$\frac{w_s^*}{d_s} - \frac{w_s^*}{D_s^*} = \alpha_s \qquad \text{for all } s \in S \tag{2}$$

Our first result shows that Vegas sources have

$$U_s(x_s) \;=\; \alpha_s d_s \log x_s \tag{3}$$

as their utility functions. Moreover the objective of Vegas is to choose source rates $x = (x_s, s \in S)$ so as to

$$\max_{x \geq 0} \quad \sum_s U_s(x_s) \tag{4}$$

$$\text{subject to} \quad \sum_{s \in S(l)} x_s \;\leq\; c_l, \quad l \in L \tag{5}$$

The utility function U_s is strictly concave increasing, meaning that a Vegas source is greedy (utility always increasing with rate) but there is a diminishing return (concavity). Constraint (5) says that the aggregate source rate at any link l does not exceed the capacity. We will refer to (4–5) as the primal problem. A rate vector x that satisfies the constraints is called *feasible* and a feasible x that maximizes (4) is called *primal optimal* (or simply *optimal*). A unique optimal rate vector exists since the objective function is strictly concave, and hence continuous, and the feasible solution set is compact.

THEOREM 2.1. *Let $w^* = (w_s^*, s \in S)$ be the equilibrium windows of Vegas and $D^* = (D_s^*, s \in S)$ the associated equilibrium round-trip times, i.e., they satisfy (2). Suppose packets are served first-in-first-out at all links. Then the equilibrium source rates $x^* = (x_s^*, s \in S)$ defined by $x_s^* = w_s^*/D_s^*$ is the unique optimal solution of (3–5).*

Proof. By the Karush-Kuhn-Tucker theorem a feasible source rate vector $x^* \geq 0$ is optimal if and only if there exists a vector $p^* = (p_l^*, l \in L) \geq 0$ such that, for all s,

$$U_s'(x_s^*) = \frac{\alpha_s d_s}{x_s^*} = \sum_{l \in L(s)} p_l^* \tag{6}$$

and, for all l, $p_l^* = 0$ if the aggregate source rate at link l is strictly less than the capacity $\sum_{s \in S(l)} x_s^* < c_l$ (complementary slackness). We now prove that the equilibrium backlog at the links provide such a vector p^*, and hence the equilibrium rates are optimal.

Let b_l^* be the equilibrium backlog at link l. The fraction of b_l^* that belongs to source s under first-in-first-out service discipline is $\frac{x_s^*}{c_l} b_l^*$ where c_l is the link capacity. Hence source s maintains a backlog of $\sum_{l \in L(s)} \frac{x_s^*}{c_l} b_l^*$ in its path in equilibrium. Since the window size equals the bandwidth-delay product plus the total backlog in the path, we have

$$w_s^* - x_s^* d_s \;=\; \sum_{l \in L(s)} \frac{x_s^*}{c_l}\, b_l^* \tag{7}$$

Thus, from (2) we have in equilibrium (recalling $x_s^* = w_s^*/D_s^*$)

$$\alpha_s \;=\; \frac{w_s^*}{d_s} - \frac{w_s^*}{D_s^*} \;=\; \frac{1}{d_s}(w_s^* - x_s^* d_s) \;=\; \frac{1}{d_s}\left(\sum_{l \in L(s)} \frac{x_s^*}{c_l}\, b_l^*\right)$$

where the last equality follows from (7). This yields (6) upon identifying

$$p_l^* \;=\; \frac{b_l^*}{c_l}$$

and rearranging terms. Clearly, x^* must be feasible since otherwise the backlog will grow without bound, contradicting (7). Since the equilibrium backlog $b_l^* = 0$ at a link l if the aggregate source rate is strictly less than the capacity, the complementary slackness condition is also satisfied. □

2.3 Dual Problem

Theorem 2.1 asserts that if Vegas converges, then the equilibrium solves the optimization problem (3–5). The question remains whether the Vegas algorithm (1), or its more com-

plete specification in (16–19) below, indeed converges. A rigorous convergence analysis is difficult because the update function is nonlinear and discontinuous (see a heuristic analysis in [Mo et al. 1999] for two sources sharing a single link). In this subsection and the next, we interpret the Vegas algorithm (1) as *approximately* carrying out a gradient projection algorithm for the dual problem of (4–5). This gradient projection algorithm itself can be proved to converge provided that the stepsize is sufficiently small. While this does *not* imply the convergence of the approximate algorithm (1), it suggests that Vegas, in tracking the gradient projection algorithm, is likely to converge in practice to a neighborhood of the equilibrium, as confirmed by the simulation results in Section 6. Moreover, this interpretation leads to a complete model of Vegas network, explained in the next subsection.

We start by reviewing the duality approach of [Low and Lapsley 1999; Athuraliya and Low 2000b] to solve (4–5) and introduce a scaled gradient projection algorithm to solve the dual problem. In the next subsection, we interpret the Vegas algorithm (1) as an approximate version of this algorithm.

Associated with each link l is a dual variable p_l. Define the Lagrangian of (4–5) as [Bertsekas 1995]:

$$\begin{aligned} L(x,p) &= \sum_s U_s(x_s) - \sum_l p_l\Big(\sum_{s\in S(l)} x_s - c_l\Big) \\ &= \sum_s (U_s(x_s) - x_s \sum_{l\in L(s)} p_l) + \sum_l p_l c_l. \end{aligned}$$

The objective function of the dual problem of (4–5) is $D(p) := \max_{x\geq 0} L(x,p)$ [Bertsekas and Tsitsiklis 1989, Section 3.4.2]. Notice that the first term are separable in x_s, and hence

$$\max_{x_s\geq 0} \sum_s (U_s(x_s) - x_s \sum_{l\in L(s)} p_l) = \sum_s \max_{x_s\geq 0} (U_s(x_s) - x_s \sum_{l\in L(s)} p_l)$$

Hence the dual problem is to choose the dual vector $p = (p_l, l \in L)$ so as to

$$\min_{p\geq 0} \quad D(p) := \sum_s B_s(p^s) + \sum_l p_l c_l \tag{8}$$

where

$$B_s(p^s) = \max_{x_s\geq 0} U_s(x_s) - x_s p^s \tag{9}$$

$$p^s = \sum_{l\in L(s)} p_l \tag{10}$$

If we interpret the dual variable p_l as the price per unit bandwidth at link l, then p^s in (10) is the price per unit bandwidth in the path of s. Hence $x_s p^s$ in (9) represents the bandwidth cost to source s when it transmits at rate x_s, $U_s(x_s) - x_s p^s$ is the net benefit of transmitting at rate x_s, and $B_s(p^s)$ represents the maximum benefit s can achieve at the given (scalar) price p^s. Given a price vector $p \geq 0$, source s can be induced to choose the unique rate $x_s(p^s)$ that maximizes (9) based only on local information. Moreover, by duality theory [Bertsekas 1995], if the price vector p^* is *dual optimal* (i.e., minimizes (8)), then these individually optimal rates $x_s(p^{*s})$ will also be primal optimal, i.e., maximize (4–5) as well.

In the rest of the paper, we will refer to p_l as link price, $p^s = \sum_{l\in L(s)} p_l$ as path price (of source s), and the vector $p = (p_l, l \in L)$ simply as price. For Vegas, the link price p_l

turns out to be the *queueing* delay at link l; see below. An *optimal* p^* is a shadow price (Lagrange multiplier) with the interpretation that p_l^* is the marginal increment in aggregate utility $\sum_s U_s(x_s)$ for a marginal increment in link l's capacity c_l.

An iterative gradient projection algorithm to solve the dual problem is proposed in [Low and Lapsley 1999; Athuraliya and Low 2000b]. Note that the gradient $\nabla D(p(t))$ of the dual objective function D points in the direction of steepest ascent. To minimize $D(p(t))$, prices are adjusted in the opposite direction of the gradient $\nabla D(p(t))$:

$$p(t+1) \;=\; [p(t) - \gamma \Theta \nabla D(p(t))]^+$$

Here $\gamma > 0$ is a constant stepsize, Θ = diag $(\theta_l, l \in L)$ is a positive diagonal scaling matrix, and $[z]^+ = \max\{0, z\}$. The structure of the dual problem allows a decentralized and distributed implementation of the above algorithm.

Let $x_s(t)$ denote the unique source rate that maximizes (9–10) with p replaced by $p(t)$, and $x^l(t) = \sum_{s \in S(l)} x_s(t)$ denote the aggregate source rate at link l. Let $p^s(t) = \sum_{l \in L(s)} p_l(t)$ denote the path price of source s. Then it can be shown (see, e.g., [Bertsekas 1995]) that the following iteration implements the gradient projection algorithm with Vegas' utility function $U_s(x_s) = \alpha_s d_s \log x_s$:

$$p_l(t+1) \;=\; [p_l(t) + \gamma \theta_l (x^l(p(t)) - c_l)]^+ \tag{11}$$

where

$$x_s(t) \;=\; \frac{\alpha_s d_s}{p^s(t)} \tag{12}$$

To interpret, note that $x^l(t)$ represents the demand for bandwidth at link l and c_l represents the supply. Hence the price is adjusted according to the law of demand and supply: if demand exceeds the supply, raise the price; otherwise reduce it. The algorithm in [Low and Lapsley 1999] is a special case with the scaling factor $\theta_l = 1$. The scaling factor θ_l in [Athuraliya and Low 2000b] is time-varying and chosen to approximate the inverse of the Hessian matrix $\nabla^2 D(p(t))$. By (12), source s sets its rate to the unique maximizer of (9–10). This is referred to as the demand function in economics: the higher the path price $p^s(t)$ (i.e., the more congested the path), the lower the source rate. Notice the decentralized nature of the algorithm where each link and each source updates individually using only local information.

The following result says that the scaled gradient projection algorithm defined by (11–12) converges to yield the unique optimal source rates. Since Vegas can be regarded as an approximate version of this algorithm, this theorem underlies its stability. Its proof is a straightforward adaptation of the proof in [Athuraliya and Low 2000b] for the convergence of an approximate Newton-algorithm to solve the dual problem (8–10), and is omitted.

THEOREM 2.2. *Provided that the stepsize γ is sufficiently small, then starting from any initial rates $x(0) \geq 0$ and prices $p(0) \geq 0$, every limit point (x^*, p^*) of the sequence $(x(t), p(t))$ generated by algorithm (11–12) is primal-dual optimal.*

The proof in [Athuraliya and Low 2000b] also provides an explicit (conservative) bound on the stepsize γ that guarantees the optimality of (x^*, p^*). Define $\overline{L} := \max_{s \in S} |L(s)|$ and $\overline{S} := \max_{l \in L} |S(l)|$. In words $\overline{L}$ is the length of a longest path used by the sources and $\overline{S}$ is the number of sources sharing a most congested link. Suppose source rates $x_s(t)$ are upper bounded so that $\alpha_s d_s / x_s^2(t) \geq \alpha d / \overline{x}^2$ for all s, t. Specialize to the Vegas algorithm where

the scaling factor is $\theta_l = 1/c_l$ (see below); suppose link capacities are lower bounded, $c_l \geq \underline{c}$. Then the conclusion of Theorem 2.2 holds if

$$\gamma \leq \frac{2\alpha d \underline{c}}{\overline{L}\overline{S}\overline{x}^2}$$

2.4 Vegas Algorithm

We now interpret the Vegas algorithm as approximately carrying out the scaled gradient projection algorithm (11–12).

The algorithm takes the familiar form of adaptive congestion control: the link algorithm (11) computes a congestion measure $p_l(t)$, and the source algorithm (12) adapts the transmission rate to congestion feedback $p^s(t)$. In order to execute this algorithm, Vegas, a *source-based* mechanism, must address two issues: how to compute the link prices and how to feed back the path prices to individual sources for them to adjust their rates. We will see that, first, the price computation (11) is performed by the buffer process at each link. Second, the path prices are *implicitly* fed back to sources through round-trip times. Given the path price $p^s(t)$, source s carries out an approximate version of (12).

Specifically, suppose the input rate at link l from source s is $x_s(t)$ at time t. Then the aggregate input rate at link l is $x^l(t) = \sum_{s \in S(l)} x_s(t)$, and the buffer occupancy $b_l(t)$ at link l evolves according to:

$$b_l(t+1) \;=\; \big[b_l(t) + x^l(t) - c_l\big]^+$$

Dividing both sides by c_l we have

$$\frac{b_l(t+1)}{c_l} \;=\; \left[\frac{b_l(t)}{c_l} + \frac{1}{c_l}(x^l(t) - c_l)\right]^+ \tag{13}$$

Identifying $p_l(t) = b_l(t)/c_l$, we see that (13) is the same as (11) with stepsize $\gamma = 1$ and scaling factor $\theta_l = 1/c_l$, except that the source rates $x_s(t)$ in $x^l(t)$ are updated slightly differently from (12), as explained next.

Recall from (1) that the Vegas algorithm updates the window $w_s(t)$ based on whether

$$w_s(t) - x_s(t)d_s \;<\; \alpha_s d_s \quad \text{or} \quad w_s(t) - x_s(t)d_s \;>\; \alpha_s d_s \tag{14}$$

As in the proof of Theorem 2.1, this quantity is related to the backlog, and hence the prices, in the path:

$$w_s(t) - x_s(t)d_s \;=\; x_s(t) \sum_{l \in L(s)} \frac{b_l(t)}{c_l} \;=\; x_s(t) \sum_{l \in L(s)} p_l(t) \;=\; x_s(t)\, p^s(t) \tag{15}$$

Thus, the conditional in (14) becomes (cf. (12)):

$$x_s(t) \;<\; \frac{\alpha_s d_s}{p^s(t)} \quad \text{or} \quad x_s(t) \;>\; \frac{\alpha_s d_s}{p^s(t)}$$

Hence, a Vegas source compares the current source rate $x_s(t)$ with the target rate $\alpha_s d_s / p^s(t)$. The window is incremented or decremented by $1/D_s(t)$ in the next period according as the current source rate $x_s(t)$ is smaller or greater than the target rate $\alpha_s d_s / p^s(t)$. In contrast, the algorithm (12) sets the rate to the target rate in one step.

In summary, we have shown that the Vegas algorithm is described by the following nonlinear system:

Vegas Network Model:

$$p_l(t+1) \;=\; \left[p_l(t) + \frac{1}{c_l}\left(x^l(t) - c_l\right)\right]^+ \qquad \text{for all links } l \tag{16}$$

$$w_s(t+1) \;=\; [w_s(t) + v_s(t)]^+ \qquad \text{for all sources } s \tag{17}$$

where

$$v_s(t) \;:=\; \frac{1}{d_s + p^s(t)}\;\{\mathbf{1}(x_s(t)p^s(t) < \alpha_s d_s) - \mathbf{1}(x_s(t)p^s(t) > \alpha_s d_s)\} \tag{18}$$

$$x_s(t) \;:=\; \frac{w_s(t)}{d_s + p^s(t)} \tag{19}$$

Here $D_s(t) = d_s + p^s(t)$ is the round-trip time of s, $[z]^+ = \max\{0, z\}$, and the indicator function $\mathbf{1}(A)$ is 1 if A is true and 0 otherwise. This nonlinear system can be interpreted as an approximate version of the gradient projection algorithm (11–12) for the dual problem (8–10). The equilibrium is given by the fixed point of (16–19) and satisfies

$$\begin{aligned} x_s^* p^{*s} \;&=\; \alpha_s d_s \qquad \text{since } v_s^* \;=\; 0, \text{ and} \\ x^{*l} \;&\leq\; c_l \qquad \text{with equality if } p_l^* > 0 \end{aligned}$$

These are precisely the Karush-Kuhn-Tucker condition for the primal problem (3–5). Hence the equilibrium maximizes aggregate utility, as explained in Theorem 2.1.

2.5 Remarks

The sufficient condition in Theorem 2.2 for stability requires that the stepsize $\gamma > 0$ be sufficiently small. The original Vegas algorithm however assumes that $\gamma = 1$ (compare (11) and (16)). We now describe a way to re-introduce γ into the Vegas algorithm which can then be adjusted to ensure convergence. Multiplying both sides of (13) by $\gamma > 0$ and identifying $p_l(t) = \gamma\,\frac{b_l(t)}{c_l}$, we obtain

$$p_l(t+1) \;=\; [p_l(t) + \gamma\frac{1}{c_l}(x^l(p(t)) - c_l)]^+$$

that is, by using *weighted* queueing delays as prices, the price update is carried out with a stepsize of γ that is not necessarily one. Then (15) is modified to

$$\gamma(w_s(t) - x_s(t)d_s) \;=\; x_s(t)\sum_{l\in L(s)} \gamma\frac{b_l(t)}{c_l} \;\;=\;\; x_s(t)\,p^s(t) \tag{20}$$

Since the modification should not alter the utility functions nor the equilibrium rates, $w_s(t)$ should still be adjusted according to (18) so that, in equilibrium, $p^{*s} = \alpha_s d_s / x_s^*$ as for $\gamma = 1$. This requirement together with (20) modifies the conditional in Vegas algorithm from (14) to:

$$w_s(t) - x_s(t)d_s \;<\; \frac{\alpha_s}{\gamma}d_s \quad \text{or} \quad w_s(t) - x_s(t)d_s \;>\; \frac{\alpha_s}{\gamma}d_s$$

This amounts to using an α_s that is $1/\gamma$ times larger, i.e., using a unit of 10KBps (say) instead of KBps for α_s. Note that γ (or unit of α_s) should be the same at all sources.

Smaller γ ensures convergence of source rates, but the convergence will be slower. The tension between stability and responsiveness is present in any feedback control system.

A smaller γ also leads to a larger equilibrium backlog since $b_l(t) = c_l p_l(t)/\gamma$. This difficulty can be overcome by introducing marking to decouple the buffer process from price computation; see Section 5.

3. DELAY, FAIRNESS AND LOSS

3.1 Delay

The previous section describes two equivalent interpretations of the Vegas algorithm. The first is that a Vegas source adjusts its rate so as to maintain its actual rate to be between α_s and β_s KB/s lower than its expected rate, where α_s (typically $1/d_s$) and β_s (typically $3/d_s$) are parameters of the Vegas algorithm. The expected rate is the maximum possible for the current window size, realized if and only if there is no queueing in the path. The rationale is that a rate that is too close to the maximum underutilizes the network, and one that is too far indicates congestion. The second interpretation is that a Vegas source adjusts its rate so as to maintain between $\alpha_s d_s$ (typically 1) and $\beta_s d_s$ (typically 3) number of packets buffered in its path, so as to take advantage of extra capacity when it becomes available.

The duality model suggests a third interpretation. The dynamics of the buffer process at link l implies the relation (comparing (11) and (13)):

$$p_l(t) = \frac{b_l(t)}{c_l} \tag{21}$$

It says that the link price $p_l(t)$ is the queueing delay at link l faced by a packet arrival at time t. The path price $p^s(t) = \sum_{l \in L(s)} p_l(t)$ is thus the *end–to–end* queueing delay (without propagation delay). It is the congestion signal a source needs to adjust its rate, and the source computes it by taking the difference between the round-trip time and the (estimated) propagation delay. Then (12) implies that a Vegas source sets its (target) rate to be proportional to the ratio of propagation to queueing delay, the proportionality constant being between α_s and β_s. Hence the larger the queueing delay, the more severe the congestion and the lower the rate. This interpretation of Vegas will be used to modify Vegas when used with REM; see Section 5 below.

It also follows from (12) that in equilibrium the bandwidth–*queueing*–delay product of a source is equal to the extra packets $\alpha_s d_s$ buffered in its path:

$$x_s^* p^{*s} = \alpha_s d_s \tag{22}$$

This is Little's Law in queueing theory. The relation (22) then implies that queueing delay p^{*s} must increase, since x_s^* must decrease, with the number of sources. This is just a restatement that every source attempts to keep some extra packets buffered in its path.

3.2 Fairness

Although we did not recognize it at the time, there are two equally valid implementations of Vegas, each springing from a different interpretation of an ambiguity in the algorithm. The first, which corresponds to the actual code, defines the α_s and β_s parameters in terms of bytes (packets) per *round-trip time*, while the second, which corresponds to the prose in [Brakmo and Peterson 1995], defines α_s and β_s in terms of bytes (or packets) per *second*. These two implementations have an obvious impact on fairness: the second favors sources with a large propagation delay,

In terms of our model, the $\log$ utility function (Theorem 2.1) implies that the equilibrium rates x^* are *weighted proportionally fair* [Kelly 1997; Kelly et al. 1998]: for any other

feasible rate vector x, we have

$$\sum_s \alpha_s d_s \frac{x_s - x_s^*}{x^*} \le 0$$

The first implementation has $\alpha_s = \alpha/d_s$ inversely proportional to the source's propagation delay. Then the utility functions $U_s(x_s) = \alpha_s d_s \log x_s = \alpha \log x_s$ are identical for all sources, and the equilibrium rates are *proportionally fair* and are independent of *propagation* delays. We call this implementation *proportionally fair* (PF).

The second implementation has identical $\alpha_s = \alpha$ for all sources. Then the utility functions and the equilibrium rates are weighted proportional fair, with weights proportional to sources' propagation delays. (22) implies that if two sources r and s face the same path price, e.g., in a network with a single congested link, then their equilibrium rates are proportional to their propagation delays:

$$\frac{x_r^*}{d_r} = \frac{x_s^*}{d_s}$$

In a network with multiple congested links, weighting the utility by propagation delay has a balancing effect to the "beat down" phenomenon, if the propagation delay is proportional to the number of congested links in a source's path. We call the second implementation *weighted proportionally fair* (WPF).

This contrasts with TCP Reno which attempts to equalize *window* [Kelly 1999; Kunniyur and Srikant 2000; Low 2000]:

$$x_r^* D_r^* = x_s^* D_s^*$$

and hence a source with twice the (round-trip) delay receives half as much bandwidth. This discrimination against connections with high propagation delay is well known in the literature, e.g., [Floyd 1991; Floyd and Jacobson 1993; Lakshman and Madhow 1997; Mathis et al. 1997; Bonald 1998]. Indeed the same methodology developed here has been applied in [Low 2000; Low et al. 2002] to Reno, where, by interpreting loss probability as the dual variable, Reno is shown to have a utility function

$$U_s(x_s) = \frac{\sqrt{2}}{D_s} \tan^{-1} \frac{x_s D_s}{\sqrt{2}}$$

3.3 Loss

Provided that buffers at links l are large enough to accommodate the equilibrium backlog $b_l^* = p_l^* c_l$, a Vegas source will not suffer any loss in equilibrium owing to the feasibility condition (5). This is in contrast to TCP Reno which constantly probes the network for spare capacity by linearly increasing its window until packets are lost, upon which the window is multiplicatively decreased. Thus, by carefully extracting congestion information from observed round-trip time and intelligently reacting to it, Vegas avoids the perpetual cycle of sinking into and recovering from congestion. This is confirmed by the experimental results of [Brakmo and Peterson 1995] and [Ahn et al. 1995].

As observed in [Brakmo and Peterson 1995] and [Bonald 1998], if the buffers are not sufficiently large, equilibrium cannot be reached, loss cannot be avoided, and Vegas reverts to Reno. This is because, in attempting to reach equilibrium, Vegas sources all attempt to place $\alpha_s d_s$ number of packets in their paths, overflowing the buffers in the network.

This plausibly explains an intriguing observation in [Hengartner et al. 2000] where a detailed set of experiments assess the relative contribution of various mechanisms in Vegas to its performance improvement over Reno. The study observes that the loss recovery mechanism, not the congestion avoidance mechanism, of Vegas makes the greatest contribution. This is exactly what should be expected if the buffers are so small as to prevent Vegas from reaching an equilibrium. In [Hengartner et al. 2000], the router buffer size is 10 segments; with background traffic, it can be easily filled up, leaving little space for Vegas' backlog. The effect of buffer size on the throughput and retransmission of Vegas is illustrated through simulations in Section 6 below.

4. PERSISTENT CONGESTION

This section examines the phenomenon of persistent congestion, as a consequence of both Vegas' exploitation of buffer process for price computation and of its need to estimate propagation delay. The next section explains how this can be overcome by Random Exponential Marking (REM) [Athuraliya and Low 2000a], in the form of the recently proposed ECN bit [Floyd 1994; Ramakrishnan and Floyd 1999].

4.1 Coupling Backlog and Price

Vegas relies on the buffer process to compute its price $p_l(t) = b_l(t)/c_l$. The *equilibrium* prices depend not on the congestion control algorithm but *solely* on the problem instance: network topology, link capacities, number of sources, their routing and utility functions. As the number of sources increases, so do the equilibrium prices and backlog ($b_l^* = p_l^* c_l$). If every source keeps $\alpha_s d_s = \alpha$ packets buffered in the network, the equilibrium backlog will be αN packets, linear in the number N of sources.

4.2 Propagation Delay Estimation

We have been assuming in our model that a source knows its round trip propagation delay d_s. In practice it sets this value to the minimum round trip time observed so far. Error may arise when there is route change, or when a new connection starts [Mo et al. 1999]. First, when the route is changed to one that has a longer propagation delay than the current route, the new propagation delay will be taken as increased round trip time, an indication of congestion. The source then reduces its window, while it should have increased it. Second, when a source starts, its observed round trip time includes queueing delay due to packets in its path from existing sources. It hence overestimates its propagation delay d_s and attempts to put more than $\alpha_s d_s$ packets in its path, leading to persistent congestion.[1] We now look at the effect of estimation error on stability and fairness.

Suppose each source s uses an estimate $\hat{d}_s(t) := (1 + \epsilon_s) d_s(t)$ of its round trip propagation delay d_s in the Vegas algorithm (1), where ϵ_s is the percentage error that can be different for different sources. Naturally we assume $-1 < \epsilon_s \leq D_s(t)/d_s(t) - 1$ for all t so that the estimate satisfies $0 < \hat{d}_s(t) \leq D_s(t)$. The equilibrium windows $w^* = (w_s^*, s \in S)$

[1] A remedy is suggested for the first problem in [Mo et al. 1999] where a source keeps a record of the round trip times of the last $L \cdot N$ packets. When their minimum is much larger than the current estimate of propagation delay, this is taken as an indication of route change, and the estimate is set to the minimum round trip time of the last N packets. However, persistent congestion may interfere with this scheme. The use of Random Exponential Marking (REM) eliminates persistent congestion and facilitates the proposed modification.

and the associated equilibrium round trip times $D^* = (D_s^*, s \in S)$ then satisfy

$$\frac{w_s^*}{\hat{d}_s} - \frac{w_s^*}{D_s^*} = \alpha_s \qquad \text{for all } s \in S \tag{23}$$

The next result says that the estimation error effectively changes the utility function from (3) to:

$$U_s(x_s) \;=\; (1+\epsilon_s)\alpha_s d_s \log x_s + \epsilon_s d_s x_s \tag{24}$$

THEOREM 4.1. *Let ϵ_s be the percentage error in propagation delay estimation. Let $w^* = (w_s^*, s \in S)$ be the equilibrium windows of Vegas and $D^* = (D_s^*, s \in S)$ the associated equilibrium round trip times, i.e., they satisfy (23). Suppose packets are served first-in-first-out at all links. Then the equilibrium source rates $x^* = (x_s^*, s \in S)$ defined by $x_s^* = w_s^*/D_s^*$ is the unique optimal solution of (4–5) with utility functions given by (24).*

Proof. The argument follows the proof of Theorem 2.1, except that (6) is replaced by

$$U_s'(x_s^*) \;=\; \frac{(1+\epsilon_s)\alpha_s d_s}{x_s^*} + \epsilon_s d_s \;=\; \sum_{l \in L(s)} p_l^* \tag{25}$$

To show that the equilibrium backlog at the links provides such a vector p^*, and hence the equilibrium rates are optimal, substitute the estimated propagation delay $\hat{d}_s^* = (1+\epsilon_s)d_s^*$ for the true value d_s^* in (23) to get

$$\alpha_s \;=\; \frac{w_s^*}{(1+\epsilon_s)d_s} - \frac{w_s^*}{D_s^*}$$

Using $w_s^* - x_s^* d_s = x_s^* \sum_{l \in L(s)} b_l^*/c_l$ we thus have

$$(1+\epsilon_s)\alpha_s d_s \;=\; (w_s^* - d_s x_s^*) - \epsilon_s d_s x_s^* \;=\; \left(\sum_{l \in L(s)} \frac{b_l^*}{c_l} - \epsilon_s d_s \right) x_s^*$$

This yields (25) upon identifying $p_l^* = \frac{b_l^*}{c_l}$ and rearranging terms. As in the proof of Theorem 2.1, x^* must be feasible and since otherwise the backlog will grow without bound, contradicting the complementary slackness condition must be satisfied. Hence the proof is complete. □

The significance of Theorem 4.1 is twofold. First, it implies that incorrect propagation delay does not upset the stability of the Vegas algorithm—the rates simply pursue a different equilibrium. Second, it allows us to compute the new equilibrium rates, and hence assess the fairness, when we know the relative error in propagation delay estimation. It provides a qualitative assessment of the effect of estimation error when such knowledge is not available.

For example, suppose sources r and s see the same path price. If there is zero estimation error then their equilibrium rates are proportional to their weights:

$$\frac{\alpha_r d_r}{x_r^*} \;=\; \frac{\alpha_s d_s}{x_s^*}$$

With error, their rates are related by

$$\frac{(1+\epsilon_r)\alpha_r d_r}{x_r^*} + \epsilon_r d_r \;=\; \frac{(1+\epsilon_s)\alpha_s d_s}{x_s^*} + \epsilon_s d_s \tag{26}$$

Hence, a large positive error generally leads to a higher equilibrium rate to the detriment of other sources. For PF implementation where $\alpha_r d_r = \alpha_s d_s$, if sources have identical absolute error, $\epsilon_r d_r = \epsilon_s d_s$, then source rates are proportional to $1 + \epsilon_s$.

Although Vegas can be stable in the presence of error in propagation delay estimation, the error may cause two problems. First, over-estimation increases the equilibrium source rate. This pushes up prices and hence buffer backlogs, leading to persistent congestion. Second, error distorts the utility function of the sources, leading to an unfair network equilibrium in favor of newer sources. We illustrate this with a simple example. The example also demonstrates an application of Theorem 4.1 and is confirmed by simulations in Section 6.2 below.

Example: Persistent Congestion
Consider a single link with capacity c pkts/ms and an infinite buffer shared by N sources, all with a common round trip propagation delay of d ms and parameter α pkts/ms. Hence if all sources know the propagation delay exactly, then each will keep αd pkts in its path in equilibrium.

Now suppose the sources activate successively. Source t, $t = 1, \ldots, N$, becomes active at the beginning of period t after sources $1, \ldots, t-1$ have reached equilibrium. Then the estimated propagation delay d_t of source t includes the queueing delay $p(t)$ due to sources $1, \ldots, t-1$. In period 1, only source 1 is active, so source 1 estimates its propagation delay correctly, $d_1 = d$, and generates a queueing delay of $p(1) = \alpha d/c$ pkts. This is the error in propagation delay estimation by source 2, i.e., $d_2 = d + p(1)$. Sources 1 and 2 generate an equilibrium queueing delay of $p(2)$ in period 2 that satisfies (25) in the proof of Theorem 4.1. Since $x_1^* + x_2^* = c$, we have

$$\frac{\alpha d}{p(2)} + \frac{\alpha(d + p(1))}{p(2) - p(1)} \;=\; c$$

By induction, the equilibrium queueing delays in successive periods can be computed recursively by

$$\frac{\alpha d}{p(t)} + \frac{\alpha(d + p(1))}{p(t) - p_1} + \frac{\alpha(d + p(2))}{p(t) - p(2)} + \cdots + \frac{\alpha(d + p(t-1))}{p(t) - p(t-1)} \;=\; c, \quad t = 2, \ldots, N \tag{27}$$

$$p(1) \;=\; \frac{\alpha d}{c} \tag{28}$$

Then equilibrium queue length in period t is $cp(t)$ pkts. The equilibrium rate $x_n(t)$ for source n in period t, $n = 1, \ldots, t$, is given by (from (25))

$$x_n(t) \;=\; \frac{\alpha(d + p(n-1))}{p(t) - p(n-1)} \tag{29}$$

□

4.3 Remarks

We did not see persistent congestion in our original simulations of Vegas. This is most likely due to three factors. First, and most importantly, the original implementation of Vegas reverts to Reno-like behavior when there is insufficient buffer capacity in the network. Second, our simulations did not take the possibility of route changes into consideration, but on the other hand, evidence suggests that route changes are not likely to be a problem

in practice [Paxson 1996]. Finally, the situation of connections starting up serially is pathological. In practice, connections continually come and go; hence all sources are likely to measure a baseRTT that represents the propagation delay plus the average queuing delay.

5. VEGAS WITH REM

Persistent congestion is a consequence of Vegas' reliance on *queueing* delay as a congestion measure, which makes backlog indispensable in conveying congestion to the sources. This section shows how REM (Random Exponential Marking) introduced in [Athuraliya and Low 2000a; Athuraliya et al. 2001] can be used to correct this situation.

Our goal is to preserve the equilibrium rate allocation of Vegas without the danger of persistent congestion described in the last section. This preserves the $\log$ utility function and proportional fairness of Vegas. Recall the three interpretations of Vegas discussed in Section 3.1. To preserve the equilibrium rate allocation, we use the third interpretation that a Vegas source sets its rate to be proportional to the ratio of propagation delay to path price as expressed by (12), except that path price is no longer the round-trip delay. Instead it is computed and fed back using the REM algorithm, explained below. The purpose of AQM is *not* to replace a loss signal due to buffer overflow by probabilistic dropping or marking, but rather to feed back the path price.

There are two ideas in REM that suit our purpose. First, REM strives to match rate and clear buffer, leading to high utilization and low queue. With small queues, minimum round-trip time would be an accurate approximation to propagation delay. Round trip times however no longer convey price information to a source. The second idea of REM allows sources to estimate their path prices from observed dropping or marking rate. We now summarize REM; see [Athuraliya and Low 2000a; Athuraliya et al. 2001] for design rationale, performance evaluation, and parameter setting.

Each link l updates a link price $p_l(t)$ in period t based on the *aggregate* input rate $x^l(t)$ and the buffer occupancy $b_l(t)$ at link l:

$$p_l(t+1) \;=\; [p_l(t) + \gamma(\mu_l b_l(t) + x^l(t) - c_l)]^+ \tag{30}$$

where $\gamma > 0$ is a small constant and $0 < \mu_l < 1$.[2] The parameter γ controls the rate of convergence and μ_l trades off link utilization and average backlog. Hence $p_l(t)$ is increased when the weighted sum of backlog $b_l(t)$ and mismatch in rate $x^l(t) - c_l$, weighted by μ_l, is positive, and is reduced otherwise. In equilibrium, this weighted sum is zero (at a congested link where equilibrium price $p^{*l} > 0$), which implies both $b_l^* = 0$ and $x^{*l} = c_l$. This is because if $x^{*l} \neq c_l$ then the queue length b_l^* cannot be in equilibrium. Hence $x^{*l} = c_l$. This implies $b_l^* = 0$ since the weighted sum is zero. This property leads to both high utilization and low loss and delay, as confirmed by simulation results in the next section.

To convey prices to sources, link l marks each packet arriving in period t, that is not already marked at an upstream link, with a probability $m_l(t)$ that is exponentially increasing in the congestion measure:

$$m_l(t) \;=\; 1 - \phi^{-p_l(t)} \tag{31}$$

[2]Note that if $\mu_l = 0$ then (30) reduces to (11) with scaling factor $\theta_l = 1$. Hence REM (30) can be regarded as an approximate gradient projection algorithm for the dual problem. The use of strictly positive μ_l is to drive equilibrium backlog to zero.

where $\phi > 1$ is a constant. Once a packet is marked, its mark is carried to the destination and then conveyed back to the source via acknowledgment.

The exponential form is critical for multi-link network, because the *end–to–end* probability that a packet of source s is marked after traversing a set $L(s)$ of links is then

$$m^s(t) \;=\; 1 - \prod_{l \in L(s)} (1 - m_l(t)) \;=\; 1 - \phi^{-p^s(t)} \tag{32}$$

where $p^s(t) = \sum_{l \in L(s)} p_l(t)$ is the path price. The end–to–end marking probability is high when $p^s(t)$ is large.

Source s estimates this end–to–end marking probability $m^s(t)$ by the *fraction* $\hat{m}^s(t)$ of its packets marked in period t, and estimates the path price $p^s(t)$ by inverting (32):

$$\hat{p}^s(t) \;=\; -\log_\phi(1 - \hat{m}^s(t))$$

where $\log_\phi$ is logarithm to base ϕ. It then adjusts its rate using marginal utility (cf. (12)):

$$x_s(t) \;=\; \frac{\alpha_s d_s}{\hat{p}^s(t)} \;=\; \frac{\alpha_s d_s}{-\log_\phi(1 - \hat{m}^s(t))} \tag{33}$$

In practice a source may adjust its rate more gradually by incrementing it slightly if the current rate is less than the target (the right hand side of (33)), and decrementing it slightly otherwise, in the spirit of the original Vegas algorithm (1):

Vegas with REM:

$$w_s(t+1) \;=\; \begin{cases} w_s(t) + \frac{1}{D_s(t)} & \text{if } -\frac{w_s(t)}{D_s(t)} \log_\phi\left(1 - \hat{m}^s(t)\right) < \alpha_s d_s \\ w_s(t) - \frac{1}{D_s(t)} & \text{if } -\frac{w_s(t)}{D_s(t)} \log_\phi\left(1 - \hat{m}^s(t)\right) > \alpha_s d_s \\ w_s(t) & \text{else} \end{cases}$$

6. VALIDATION

This section presents four sets of simulation results. The first set shows that source rates converge quickly under Vegas to the theoretical equilibrium, even in the presence of mice traffic, thus validating our model. The second set illustrates the phenomenon of persistent congestion discussed in Section 4. The third set shows that the source rates (windows) under Vegas/REM behave similarly to those under plain Vegas, but the buffer stays low. The last set shows that enough buffer space is necessary for plain Vegas to work properly.

We use the *ns-2* network simulator [ns], and unless stated otherwise, configure it with the topology shown in Figure 1. Each host on the left runs an FTP application that transfers a large file to its counterpart on the right. We use a packet size of 1KB. The various simulations presented in this section use different latency and bandwidth parameters, as described below.

6.1 Equilibrium and Fairness

6.1.1 *Single Link Case.* We run five classes of connections across the network shown in Figure 1. Each class has 50 flows, that is, between hosts 1a(0) and 1b(0)..., 1a(49) and 1b(49); 2a(0) and 2b(0)... and so on, for a total of 250 flows. The round trip latency for the five classes are 15ms, 15ms, 20ms, 30ms and 40ms respectively. The shared link has a bandwidth of 2.4Gbps, which is close to an OC-48 link, and all host/router links have a

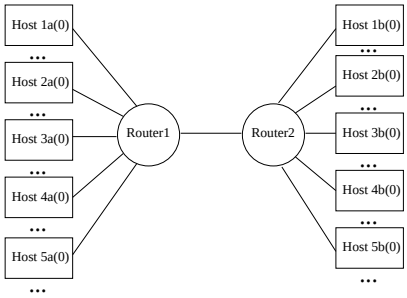


Fig. 1. Network topology

bandwidth of 100Mbps. Routers maintain a FIFO (first in first out) queue with unlimited capacity.

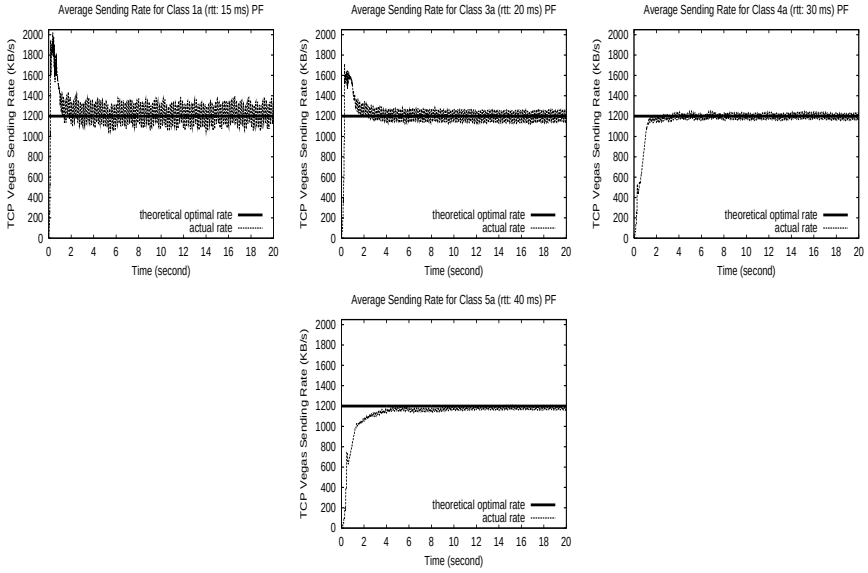


Fig. 2. Average sending rates for single link case with proportionally fair implementation. Host Class 2a is not shown, but behaves similarly to Host Class 1a.

As described in Section 3, there are two different implementations of Vegas with different fairness properties. For proportional fairness, we set $\alpha_s = 2$ packets *per RTT* and we let $\alpha_s = \beta_s$ in *ns-2*. The model predicts that all connections receive an equal share (1200KBps) of the bottleneck link and the simulations confirm this. This contrasts sharply with Reno which is well known to discriminate against connections with large propagation delays. Figure 2 plots the *average* sending rate of 50 flows in each class against the predicted rates (thick straight lines): all connections quickly converge to the predicted rate. (Individual source rates have similar behavior, so we only show their averages.) Table I

summarizes other performance values,[3] which further demonstrate how well the model predicts the simulation. All simulation numbers are averages among sources of each class in equilibrium.

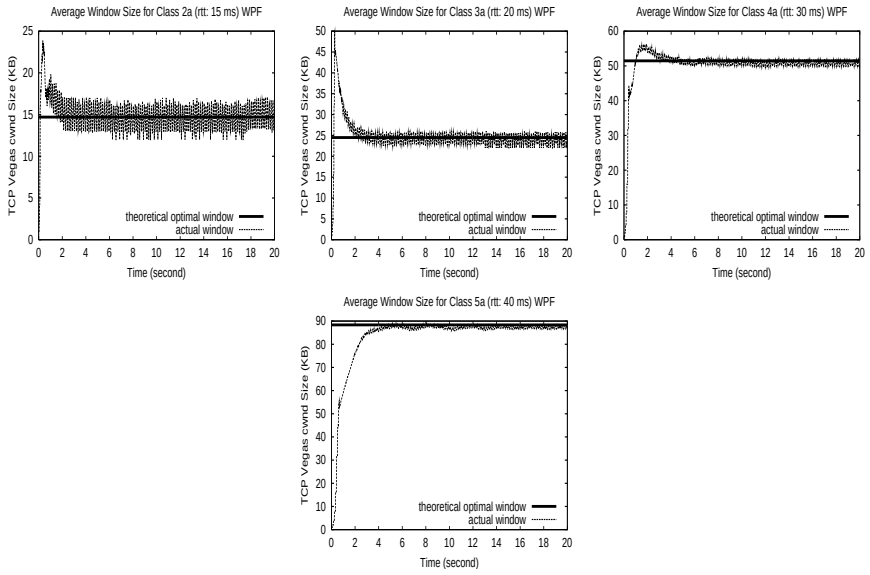


Fig. 3. Average sending rates for single link case with weighted proportionally fair implementation. Host Class 1a is not shown.

For weighted proportional fairness, we set α_s to 2 packets *per 10ms*, which means each source will have a different number of extra packets in the pipe and the optimal sending rate will be proportional to the propagation delay. The results are shown in Figure 3, except this time we show the average congestion windows instead of the sending rates. The other performance numbers are in Table II, which again show that simulations closely follow model's predictions.

Notice that in both cases, sources with smaller round-trip times converge more rapidly (see Figures 2 and 3). This scaling down of response by delay enhances stability in the face of feedback delay [Paganini et al. 2001]. Moreover, the individualized scaling of Vegas has the appealing feature that sources with low delays can respond quickly, without compromising fairness, and it is only those sources whose fast response is destabilizing (those with long delays) that must slow down.

6.1.2 *Multi-Link Case*. We also simulated a network topology with multiple links, as shown in Figure 4. This topology is almost the same as that used in [Brakmo and Peterson 1995], except that to simplify computation, we set the bandwidth of the "backbone" to be 2.4Gbps. We now have six classes of sources (1a to 6a) and six classes of sinks (1b to 6b). Each class has 50 sources or sinks; as usual, links between a source or sink and a

[3]The baseRTT, for both Model and Simulation, includes round trip propagation delay and transmission time for a packet and its acknowledgment.

Source	*1*a		*2*a		*3*a		*4*a		*5*a	
Class	M	S	M	S	M	S	M	S	M	S
baseRTT (ms)	15.17	15.17	15.17	15.17	20.17	20.17	30.17	30.17	40.17	40.17
RTT w/ queueing (ms)	16.84	16.88	16.84	16.88	21.84	21.88	31.84	31.9	41.84	41.87
Sending rate (KB/s)	1200	1221	1200	1203	1200	1193	1200	1183	1200	1169
Congestion window (pkts)	20.2	20.4	20.2	20	26.2	26	38.2	38	50.2	49.4

Queue	Model	Simulation
at Router1 (pkts)	500	515

Table I. Comparison of theoretical and simulation results for the single link case with proportionally fair implementation. M denotes model and S denotes simulation.

Source	*1*a		*2*a		*3*a		*4*a		*5*a	
Class	M	S	M	S	M	S	M	S	M	S
baseRTT (ms)	15.17	15.17	15.17	15.17	20.17	20.17	30.17	30.17	40.17	40.17
RTT w/ queueing (ms)	19.2	19.25	19.2	19.25	24.2	24.3	34.2	34.3	44.2	44.24
Sending rate (KB/s)	753.2	772	753.2	762	1001	992	1498	1475	1994	1957
Congestion window (pkts)	14.5	15	14.5	14.9	24.2	24	51.2	50.9	88.2	87.4

Queue	Model	Simulation
at Router1 (pkts)	1208.85	1200.5

Table II. Comparison of theoretical and simulation results for the single link case with weighted proportionally fair implementation. M denotes model and S denotes simulation.

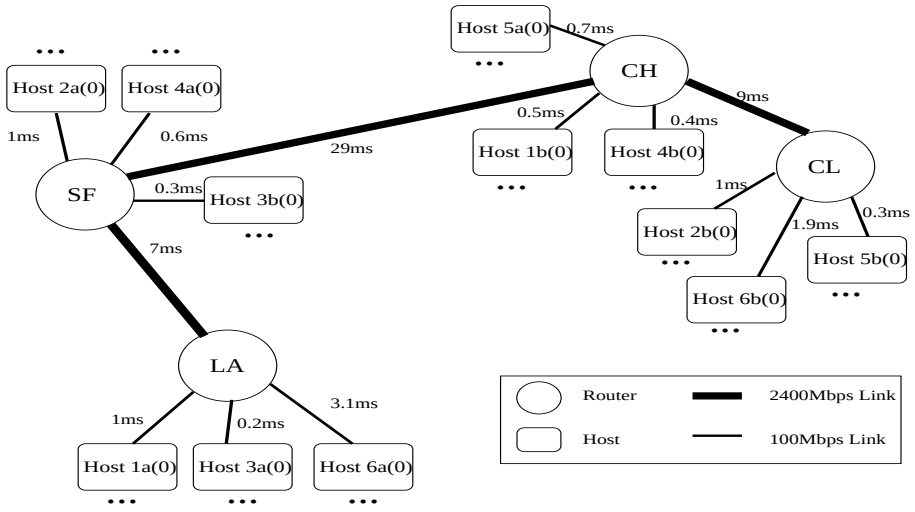


Fig. 4. Network topology for multi-link simulations.

router have a bandwidth of 100Mbps. Similar to previous simulations, an FTP application on each "a" Host transfers a large file to its counterpart sink on the "b" Host using a packet size of 1KB (e.g., 1a(3) to 1b(3), 5a(10) to 5b(10)). Altogether there are 300 flows, which

have round trip propagation delays from 20ms to 100ms.

Source	*1*a		*2*a		*3*a		*4*a		*5*a		*6*a	
Class	M	S	M	S	M	S	M	S	M	S	M	S
baseRTT (ms)	75.17	75.17	80.17	80.17	15.17	15.17	60.17	60.17	20.17	20.17	100.18	100.18
RTT w/ queueing (ms)	76.96	76.64	81.96	81.62	15.89	15.77	61.23	61	20.89	20.76	102.69	102.24
Sending rate (KB/s)	1382	1363	1382	1378	3618	3625	2236	2237	3618	3601	1000	968
Congestion window (pkts)	106.35	105.4	113.27	113.8	57.5	57.2	136.93	138.1	75.6	75.3	102.69	100.7

Queue	LA		SF		CH	
(pkts)	M	S	M	S	M	S
	166.0	160.7	268.5	259.2	166.0	159.6

Table III. Comparison of theoretical and simulation results for the multi-link case with proportionally fair implementation. M denotes model and S denotes simulation.

Source	*1*a		*2*a		*3*a		*4*a		*5*a		*6*a	
Class	M	S	M	S	M	S	M	S	M	S	M	S
baseRTT (ms)	75.17	75.17	80.17	80.17	15.17	15.17	60.17	60.17	20.17	20.17	100.18	100.18
RTT w/ queueing (ms)	85.74	85.5	91.12	90.8	16.4	16.3	69.5	69.41	21.78	21.68	112.35	111.9
Sending rate (KB/s)	1463	1448	1509	1495	2819	2804	1310	1316	2775	2766	1714	1699
Congestion window (pkts)	125.47	125.2	137.5	137.3	46.24	45.76	91.04	92.21	60.44	60.34	192.6	192.13

Queue	LA		SF		CH	
(pkts)	M	S	M	S	M	S
	319.5	336.4	2748.0	2738	432.5	442.8

Table IV. Comparison of theoretical and simulation results for the multi-link case with weighted proportionally fair implementation. M denotes model and S denotes simulation.

We repeated simulations for Proportionally Fair and Weighed Proportionally Fair implementations under this new setup for 20 seconds. Tables III and IV summarize various (average) performance values predicted from duality model and measured from simulation. Again, the simulation measurements match our predictions very well.

6.1.3 *Mice Traffic.* As mentioned at the end of Section 1.1, even though most TCP connections are mice, most packets belong to elephants. TCP aims to control elephants, not mice. The deterministic fluid model in this paper ignores mice traffic. The next experiment validates the model in the presence of mice. It shows that, indeed, the equilibrium properties of the network are largely determined by elephants, as heavy-tail file sizes imply.

We use the same simulation configuration as in the first set of experiments, with 5 classes of sources. Each class has 50 sources all with the same propagation delay. Of the 50 sources in each class, 10 sources are persistent FTP sources (elephants) and 40 are random on-off sources (mice). When a random source is on, it generates a file of size that is exponentially distributed with mean 1MB; the file is then transported over TCP Vegas. When the transfer is complete, it turns off for a random period that is exponentially distributed with mean 3 sec. With these parameters, about 20% of packets are generated by the 200 random on-off sources and 80% packets are generated by the 50 persistent FTP sources. The Vegas parameter is $\alpha_s = 2$ packets per RTT, so that all persistent sources should have the same equilibrium rate of $3 \times 10^5 \times 80\%/50$ = 4800 KB/s. Figure 5 shows that average sending rate of the 10 persistent FTP sources in each class (individual sending rates behave similarly as the average) and the instantaneous queue. The elephants can be effectively controlled with 20% mice traffic and converge to their equilibrium rates, although

the convergence is slower than that in earier experiments. Again, sources with smaller delays converge faster. The propagation delays (not shown here) are accurately estimated in the presence of mice traffic. The queue oscillates rather severely because of the random on-off sources.

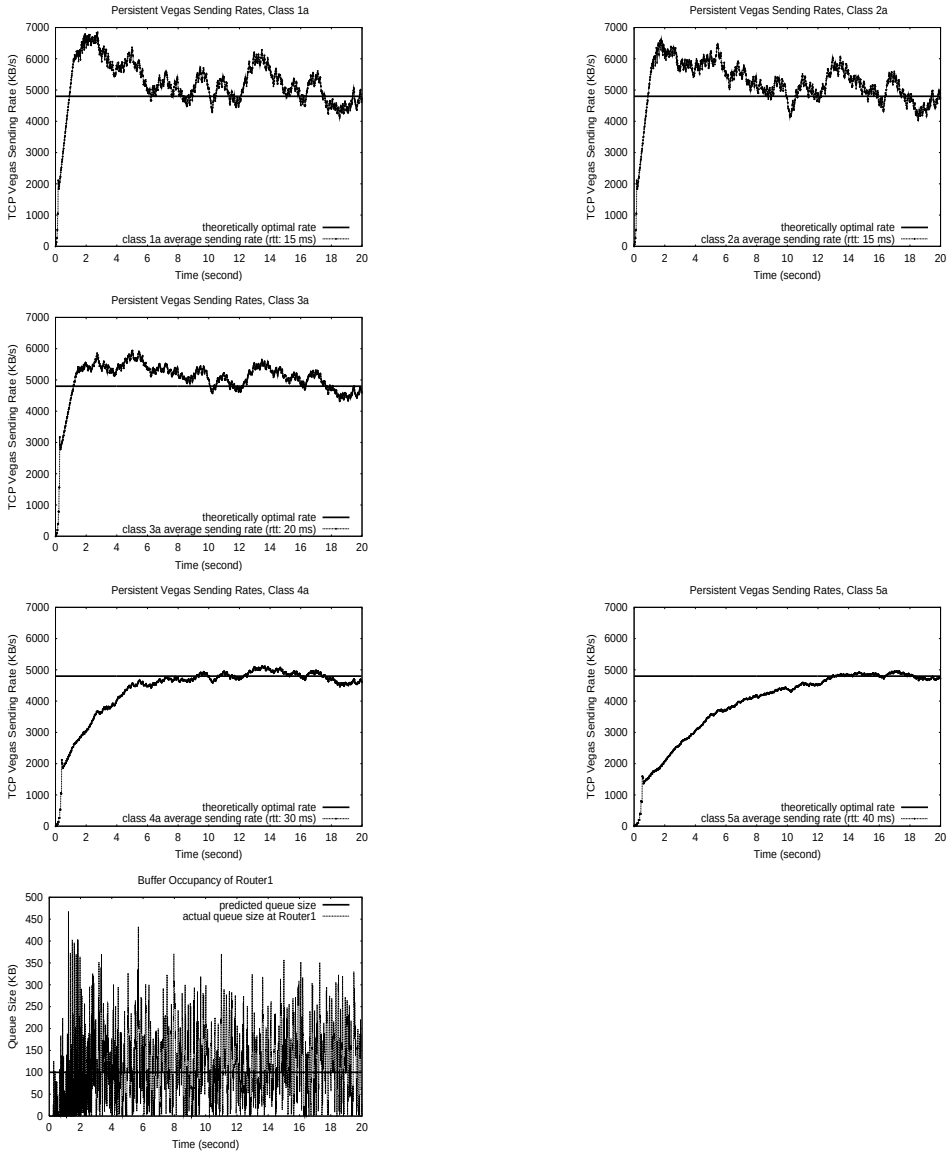
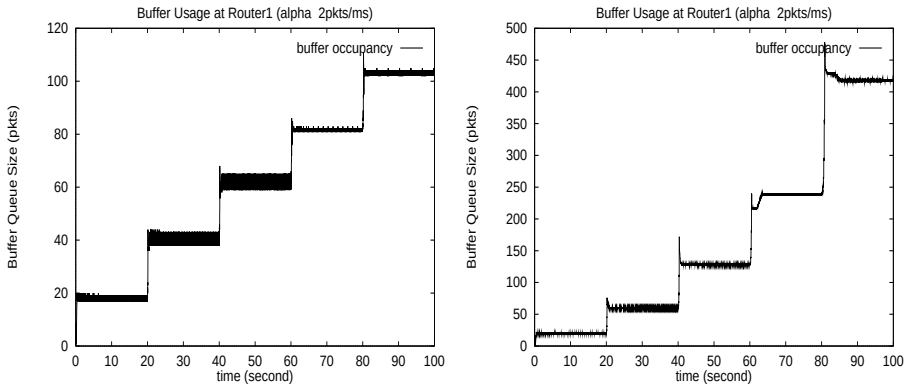


Fig. 5. Sending rates and queue sizes for persistent flows with mice traffic.

6.2 Persistent Congestion

We next validate that Vegas can lead to persistent congestion under pathological conditions. We choose *only* 5 source-sink pairs in topology of Figure 1, namely Host 1a-5a and Host 1b-5b; and set the round trip propagation delay to 10ms for *all* connections, the host-router links are all 1600 Mbps, and the bottleneck link has a bandwidth of 48 Mbps. We set α_s to 2 packets-per-ms, so each source strives to maintain 20 packets in their path. We assume the routers have infinite buffer capacity.

We first hard-code the round trip propagation delay to be 10 ms for each source, thus eliminating the error in propagation delay estimation. We then run five connections, each starting 20 seconds after the previous connection. That is, Host 1a starts sending at time 0, 2a starts at 20s, and so on. As shown in Figure 6(a), the buffer occupancy increases linearly in the number of sources, as expected.



(a) Without propagation delay error (b) With propagation delay error

Fig. 6. Effect of propagation delay error on queue length.

Time	*1*a (KB/s)		*2*a (KB/s)		*3*a (KB/s)		*4*a (KB/s)		*5*a (KB/s)		*Q*ueue (pkts)	
	M	S	M	S	M	S	M	S	M	S	M	S
0 – 20s	6000	5980									20	19.8
20 – 40s	2000	2050	4000	3920							60	59
40 – 60s	940	960	1490	1460	3570	3540					127	127.3
60 – 80s	500	510	730	724	1350	1340	3390	3380			238	237.5
80 – 100s	290	290	400	404	670	676	1300	1298	3340	3278	416	416.3

Table V. Equilibrium rates and queue lengths with propagation delay error. M denotes Model and S denotes Simulation.

Next, we let the sources discover their propagation delays. As shown in Figure 6(b), buffer occupancy grows much faster than linearly in the number of sources (notice the different scales in the figure). We have also applied Theorem 4.1 (see (27–29) in the Example)

baseRTT (ms)	*H*ost1a	*H*ost2a	*H*ost3a	*H*ost4a	*H*ost5a
no error	10.18	10.18	10.18	10.18	10.18
w/ error (M)	10.18	13.51	20.18	31.2	49.80
w/ error (S)	10.18	13.36	20.17	31.5	49.86

Table VI. BaseRTTs. M denotes Model and S denotes Simulation.

to calculate the equilibrium rates, queue size, and `baseRTT` (estimated propagation delay). The predicted and measured numbers are shown in Tables V and VI. They match very well, further verifying our model.

As Table V shows, distortion in utility functions not only leads to excess backlog, it also strongly favors new sources. Without estimation error, sources should equally share the bandwidth. With error, when all five sources are active, $x_1 : x_2 : x_3 : x_4 : x_5 = 1 : 1.4 : 2.3 : 4.5 : 11.6$.

6.3 Vegas + REM

We next implement REM at Router1 in Figure 1, which updates link price every 1ms according to (30). We adapt Vegas to adjust its rate (congestion window) based on estimated path prices, as described in Section 5. Vegas makes use of packet marking only in its congestion avoidance phase; its slow-start behavior stays unchanged.[4]

We use the same network setup as in Section 6.2. The bottleneck link also has a bandwidth of 48Mbps. Host-router links are 1600Mbps and α_s is 2 pkts-per-ms. In order to verify our new mechanism in different situations, this time we let sources (Host1-5a) have a round trip latency of 10ms, 10ms, 20ms, 10ms, 30ms respectively. REM parameters are: ϕ = 1.1, μ_l = 0.5, γ = 0.005.

As before, the 5 connections start serially at 20s intervals. Figure 7 plots the congestion window size of the five connections and buffer occupancy at Router1. As expected, each of the five connections converges to its equilibrium value predicted by duality model (thick straight lines). Source rates oscillate more severely when fewer sources are active (e.g., Host1a during time 0 - 20s). This is a consequence of the log utility function; see [Athuraliya and Low 2000a]. As more sources become active (40 - 100s), oscillation becomes smaller and convergence faster. REM eliminates the super-linear growth in queue length of Figure 6(b) while maintaining high link utilization (90% to 96%). As a result propagation delay can be accurately estimated, as shown in Table VII.

baseRTT (ms)	*H*ost1a	*H*ost2a	*H*ost3a	*H*ost4a	*H*ost5a
Model	10.18	10.18	20.18	10.18	30.18
Simulation	10.18	10.18	20.18	10.18	30.19

Table VII. `baseRTT` in Vegas+REM.

6.4 Effect of Buffer Capacity

Our model and all previous simulations assume an infinite buffer capacity. The next simulation studies the effect of buffer capacity on the performance of Vegas and Reno. It

[4]During slow-start, Vegas keeps updating the variable fraction $\hat{m}^s(t)$, but does not use it in window adjustment.

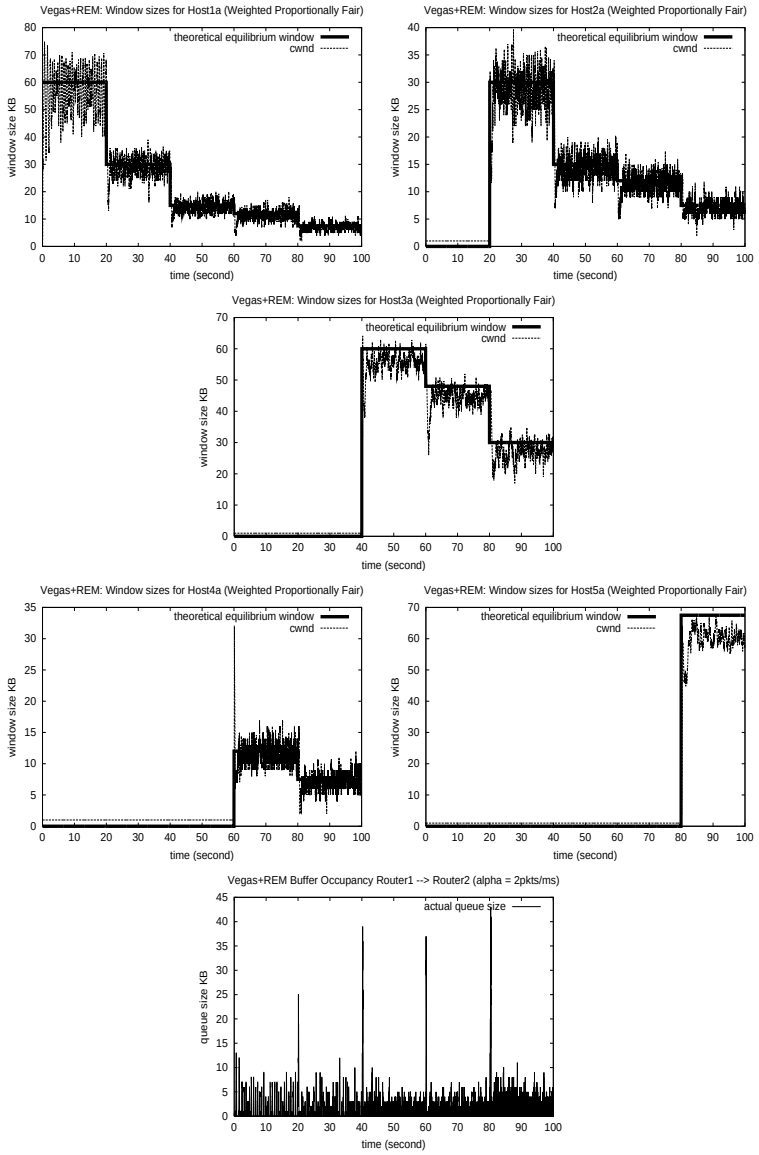
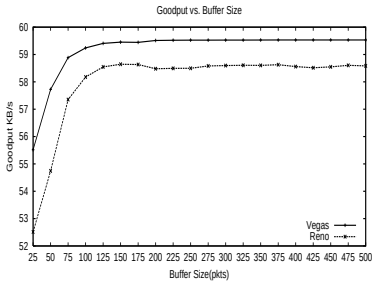


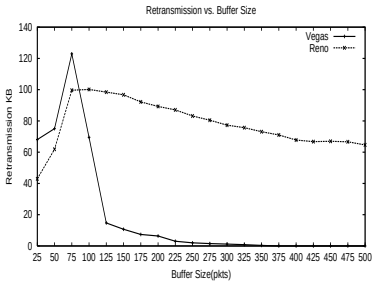
Fig. 7. Vegas+REM with link utilizations of 90%(0-20s), 96%(20-40s), 93%(40-60s), 91%(60-80s), and 90%(80-100s).

confirms our discussion in Section 3.3 and offers a plausible explanation for the intriguing observation that the congestion avoidance mechanism of Vegas contributes little to its throughput and retransmission improvement over Reno.

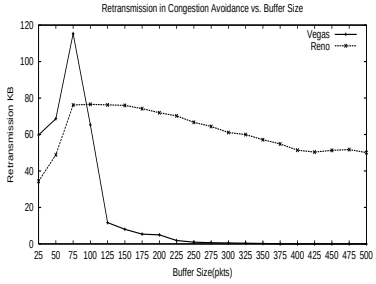
In [Hengartner et al. 2000], TCP Vegas is decomposed into several individual mechanisms and the effect of each on performance is assessed by taking the approach of a 2^k factorial design with replications. This work deploys a useful methodology and provides



(a) Goodput



(b) Retransmission



(c) Retransmission in CA

Fig. 8. Effect of buffer capacity on performance. All numbers are averages of 100 flows over a 50s period. For Vegas, $\alpha = 1$, $\beta = 3$ packet per RTT.

insights into the relative importance of different algorithms in Vegas. However, the final conclusion that Vegas' more aggressive recovery mechanism has the largest effect on performance, while its congestion avoidance mechanism contributes little, could be limited by the fact that in that setup, the bottleneck router only has a 10 packet queue, which could be easily filled up by background traffic. As a result, without enough buffer for its backlog, Vegas reverts to Reno and the changes to its recovery mechanism then stand out as the largest contributor to performance. If buffer space is enough, Vegas will maintain a steady sending rate without *any* retransmission.

To validate our claim, we simulate the same topology as in [Hengartner et al. 2000], which is similar to Figure 1, but the bottleneck link has a capacity of 6000 KB/sec. There are 100 persistent TCP flows from Host1a to Host1b with round trip propagation delay of 50ms. We run two sets of simulations, one with 100 Reno flows and the other with 100 Vegas flow, and compare the performance of Reno and Vegas at different buffer sizes. All host-router links are 10Mbps Ethernet. To isolate the effect of buffer size on the behavior of congestion avoidance mechanism, we omit the background traffic in our simulations. Such long transfers minimize the effect of other mechanisms such as slow-start on the performance and our measurements are based on the first 50 seconds of the transfer. We set $\alpha_s = 1$ and $\beta_s = 3$ pkts-per-round-trip for Vegas so that the total equilibrium queue

length should be between 100 to 300 packets. Figure 8 shows the goodput[5], retransmission and retransmission during congestion avoidance of these 100 flows as a function of buffer size at Router1. It confirms that Vegas has a steady sending rate and no retransmissions as long as the buffer sizes exceeds a threshold. In contrast, retransmission remains significant for Reno even at large buffer sizes.

This simulation illustrates that TCP Vegas' congestion avoidance mechanism will only get its full benefit when the network has enough buffer space to hold Vegas' backlog. In that case, Vegas will have a stable send rate and no retransmission, and the performance advantage, although not as pronounced, can be ascribed to Vegas' congestion avoidance mechanism. When buffer space is small, Vegas' `cwnd` behaves like Reno's and Vegas' recovery mechanism plays a larger role in the performance difference. It is worth noting, however, that there is a danger in trying to isolate the contributions of recovery mechanism and congestion avoidance. The recovery mechanism was designed to retransmit more aggressively than Reno, while the congestion avoidance mechanism was designed primarily to reduce loss (not increase throughput), and hence, provide balance against the effects of the recovery mechanism.

7. RELATED WORK

The optimal flow control problem (4–5) is formulated in [Kelly 1997]. It is solved using a penalty function approach in [Kelly et al. 1998; Kunniyur and Srikant 2000] and extended in [Mo and Walrand 2000] and [La and Anantharam 2000]. The problem is solved using a duality approach in [Low and Lapsley 1999] leading to an abstract algorithm whose convergence has been proved in asynchronous environment. A practical implementation of this algorithm is proposed in [Athuraliya and Low 2000a]. The duality approach is extended to multirate multicast setting in [Kar et al. 2001].

The idea of treating source rates as primal variables and congestion measures (queueing delay in Vegas) as dual variables, and TCP/AQM as a distributed primal-dual algorithm to solve (4–5), with different utility functions, is extended in [Low 2000; Low et al. 2002] to other schemes, such as Reno/DropTail, Reno/RED, and Reno/REM. The utility functions of these schemes are derived.

In [Mo and Walrand 2000] a network is modeled by a set of inequalities which, in our context, are the feasibility condition (5), the Karush-Kuhn-Tucker condition for the constrained optimization (4–5), and the relation between window and backlog. One of the main results of [Mo and Walrand 2000] is a proof, using fixed point theory, that, given window sizes, there exists a unique rate vector that satisfies these inequalities. An alternative proof is to observe that the set of inequalities define the optimality condition for the *strict* convex program (3–5), implying the existence of a unique solution. Theorem 2.1 is first proved in [Mo and Walrand 2000]; our proof has a somewhat more intuitive presentation.

A single-link dynamical system model of Vegas is used in [Bonald 1998] to show that, in the case of homogeneous propagation delay, the system converges to a set where each source keeps between α_s and β_s packets in the link. The proof relies on an interesting contraction property that says that the difference between window sizes of any two sources

[5]Here, 'goodput' is the total number of non-duplicate packets acknowledged divided by simulation duration. The maximum is less than the link capacity of 60KB/s both because of retransmission and because of under-utilization during slow-start.

can only decrease over time.

A model of Vegas with a single link, two sources and $\alpha < \beta$ is used in [Mo et al. 1999] to show that, in equilibrium, each source maintains between α and β number of packets in the path, and that Vegas does not favor sources with short delays (their model corresponds to the proportionally fair model here; see Section 3.2). The problem of persistent congestion due to propagation delay estimation is discussed but no analysis is presented. We assume $\alpha = \beta$ and consider a multi-link multi-source optimization model whose solution yields an equilibrium characterization from which rates, queue sizes, delay, and fairness properties can be derived. This model also clarifies the precise mechanism through which persistent congestion can arise, its consequences and a cure.

A single link model and simulations are also used in [Boutremans and Boudec 2000] to investigate the effect of persistent congestion on fairness both in the case when $\alpha = \beta$ and $\alpha < \beta$. They conclude that over-estimation of propagation delay leads to (unfairly) larger equilibrium rate in both cases. When $\alpha = \beta$, there is a unique equilibrium rate vector, whereas when $\alpha < \beta$, the equilibrium rates can be any point in a set depending on detail dynamics such as the order of connection start times, making fairness harder to control. They hence suggest that α be set equal to β in practice, but do not propose any solution to reduce error in propagation delay estimation.

8. CONCLUSIONS

We have shown that TCP Vegas can be regarded as a distributed optimization algorithm to maximize aggregate source utility over their transmission rates. The optimization model has four implications. First it implies that Vegas measures the congestion in a path by *end–to–end queueing* delay. A source extracts this information from round trip time measurement and uses it to optimally set its rate. The equilibrium is characterized by Little's Law in queueing theory. Second, it implies that Vegas has a $\log$ utility function and hence the equilibrium rates are weighted proportionally fair. Third, it clarifies the mechanism, and consequences, of potential persistent congestion due to error in the estimation of propagation delay. Finally, it suggests a way to eliminate persistent congestion using REM that keeps buffer low while matching rate. We have presented simulation results that validate our conclusions.

Even though we have shown that the Vegas algorithm tracks a gradient projection algorithm that is stable, we have not provided a formal stability proof for the Vegas algorithm itself. The Vegas algorithm, modeled by (16–19) as a discrete-time nonlinear system with a discontinuous update function, is hard to analyze. It becomes still harder if we include feedback delays. Indeed, the discontinuity in the rate update (18) suggests that the system may converge to a neighborhood of the equilibrium point and circulate around it. It would be interesting to characterize the size of this equilibrium set in terms of protocol and network parameters. Finally, by using a static model (4–5), the formulation assumes that the duration of a flow is much larger than the timescale of algorithm convergence. It would be useful to extend it to a more dynamic model where sources of finite durations arrive and depart randomly, as in [de Veciana et al. 2001].

Acknowledgment. We are grateful to Sanjeewa Athuraliya, Nick Maxemchuk, and the anonymous reviewers for their helpful comments.

REFERENCES

NS network simulator. http://www.isi.edu/nsnam/ns/.

AHN, J. S., DANZIG, P. B., LIU, Z., AND YAN, L. 1995. Evaluation of TCP Vegas: emulation and experiment. In *Proceedings of SIGCOMM'95*.

ATHURALIYA, S., LI, V. H., LOW, S. H., AND YIN, Q. 2001. REM: active queue management. *IEEE Network*. Extended version in *Proceedings of ITC17*, Salvador, Brazil, September 2001. http://netlab.caltech.edu.

ATHURALIYA, S. AND LOW, S. H. 2000a. Optimization flow control, II: Implementation. Submitted for publication, http://netlab.caltech.edu.

ATHURALIYA, S. AND LOW, S. H. 2000b. Optimization flow control with Newton–like algorithm. *Journal of Telecommunication Systems 15*, 3/4, 345–358.

BERTSEKAS, D. 1995. *Nonlinear Programming*. Athena Scientific.

BERTSEKAS, D. P. AND TSITSIKLIS, J. N. 1989. *Parallel and distributed computation*. Prentice-Hall.

BONALD, T. 1998. Comparison of TCP Reno and TCP Vegas via fluid approximation. In *Workshop on the Modeling of TCP*. http://www.dmi.ens.fr/\%7Emistral/tcpworkshop.html.

BOUTREMANS, C. AND BOUDEC, J. Y. L. 2000. A note on the fairness of tcp vegas. In *Proceedings of International Zurich Seminar on Broadband Communications*. 163–170.

BRAKMO, L. S. AND PETERSON, L. L. 1995. TCP Vegas: end to end congestion avoidance on a global Internet. *IEEE Journal on Selected Areas in Communications 13*, 8 (October), 1465–80. http://cs.princeton.edu/nsg/papers/jsac-vegas.ps.

DE VECIANA, G., LEE, T.-J., AND KONSTANTOPOULOS, T. 2001. Stability and performance analysis of networks supporting elastic services. *IEEE/ACM Transactions on Networking 9*, 1 (February), 2–14.

FLOYD, S. 1991. Connections with multiple congested gateways in packet–switched networks, Part I: one–way traffic. *Computer Communications Review 21*, 5 (October).

FLOYD, S. 1994. TCP and Explicit Congestion Notification. *ACM Computer Communication Review 24*, 5 (October).

FLOYD, S. AND JACOBSON, V. 1993. Random early detection gateways for congestion avoidance. *IEEE/ACM Trans. on Networking 1*, 4 (August), 397–413. ftp://ftp.ee.lbl.gov/papers/early.ps.gz.

HENGARTNER, U., BOLLIGER, J., AND GROSS, T. 2000. TCP Vegas revisited. In *Proceedings of IEEE Infocom*.

JACOBSON, V. 1988. Congestion avoidance and control. *Proceedings of SIGCOMM'88, ACM*. An updated version is available via ftp://ftp.ee.lbl.gov/papers/congavoid.ps.Z.

KAR, K., SARKAR, S., AND TASSIULAS, L. 2001. Optimization based rate control for multirate multicast sessions. In *Proceedings of IEEE Infocom*.

KELLY, F. P. 1997. Charging and rate control for elastic traffic. *European Transactions on Telecommunications 8*, 33–37. http://www.statslab.cam.ac.uk/~frank/elastic.html.

KELLY, F. P. 1999. Mathematical modelling of the Internet. In *Proc. 4th International Congress on Industrial and Applied Mathematics*. http://www.statslab.cam.ac.uk/~frank/mmi.html.

KELLY, F. P., MAULLOO, A., AND TAN, D. 1998. Rate control for communication networks: Shadow prices, proportional fairness and stability. *Journal of Operations Research Society 49*, 3 (March), 237–252.

KUNNIYUR, S. AND SRIKANT, R. 2000. End–to–end congestion control schemes: utility functions, random losses and ECN marks. In *Proceedings of IEEE Infocom*. http://www.ieee-infocom.org/2000/papers/401.ps.

LA, R. AND ANANTHARAM, V. 2000. Charge-sensitive TCP and rate control in the Internet. In *Proceedings of IEEE Infocom*. http://www.ieee-infocom.org/2000/papers/401.ps.

LAKSHMAN, T. V. AND MADHOW, U. 1997. The performance of TCP/IP for networks with high bandwidth–delay products and random loss. *IEEE/ACM Transactions on Networking 5*, 3 (June), 336–350. http://www.ece.ucsb.edu/Faculty/Madhow/Publications/ton97.ps.

LOW, S. H. 2000. A duality model of TCP flow controls. In *Proceedings of ITC Specialist Seminar on IP Traffic Measurement, Modeling and Management*. http://netlab.caltech.edu.

LOW, S. H. AND LAPSLEY, D. E. 1999. Optimization flow control, I: basic algorithm and convergence. *IEEE/ACM Transactions on Networking 7*, 6 (December), 861–874. http://netlab.caltech.edu.

LOW, S. H., PAGANINI, F., AND DOYLE, J. C. 2002. Internet congestion control. *IEEE Control Systems Magazine*.

LOW, S. H., PETERSON, L., AND WANG, L. 2001. Understanding Vegas: a duality model. In *Proceedings of ACM Sigmetrics*. http://netlab.caltech.edu/pub.html.

MATHIS, M., SEMKE, J., MAHDAVI, J., AND OTT, T. 1997. The macroscopic behavior of the TCP congestion avoidance algorithm. *ACM Computer Communication Review 27*, 3 (July). http://www.psc.edu/networking/papers/model_ccr97.ps.

MO, J., LA, R., ANANTHARAM, V., AND WALRAND, J. 1999. Analysis and comparison of TCP Reno and Vegas. In *Proceedings of IEEE Infocom*.

MO, J. AND WALRAND, J. 2000. Fair end-to-end window-based congestion control. *IEEE/ACM Transactions on Networking 8*, 5 (October), 556–567.

PAGANINI, F., DOYLE, J. C., AND LOW, S. H. 2001. Scalable laws for stable network congestion control. In *Proceedings of Conference on Decision and Control*. http://www.ee.ucla.edu/~paganini.

PAXSON, V. 1996. End-to-end routing behavior in the Internet. In *Proceedings of SIGCOMM'96, ACM*.

PETERSON, L. L. AND DAVIE, B. S. 2000. *Computer Networks: A Systems Approach*, 2nd ed. Morgan Kaufmann.

RAMAKRISHNAN, K. K. AND FLOYD, S. 1999. A Proposal to add Explicit Congestion Notification (ECN) to IP. RFC 2481.

RAMAKRISHNAN, K. K. AND JAIN, R. 1990. A binary feedback scheme for congestion avoidance in computer networks. *ACM Transactions on Computer Systems 8*, 2 (May), 158–181.

STEVENS, W. 1999. *TCP/IP illustrated: the protocols*. Vol. 1. Addison–Wesley. 15th printing.

Received