Understanding TCP Vegas: A Duality Model *

Steven H. Low
Departments of CS and EE
Caltech, USA
slow@caltech.edu

Larry Peterson
Department of CS
Princeton University, USA
llp@cs.princeton.edu

Limin Wang
Department of CS
Princeton University, USA
lmwang@cs.princeton.edu

ABSTRACT

This paper presents a model of the TCP Vegas congestion control mechanism as a distributed optimization algorithm. Doing so has three important benefits. First, it helps us gain a fundamental understanding of why TCP Vegas works, and an appreciation of its limitations. Second, it allows us to prove that Vegas stabilizes at a weighted proportionally fair allocation of network capacity when there is sufficient buffering in the network. Third, it suggests how we might use explicit feedback to allow each Vegas source to determine the optimal sending rate when there is insufficient buffering in the network. We present simulation results that validate our conclusions.

1. INTRODUCTION

TCP Vegas was introduced in 1994 as an alternative source-based congestion control mechanism for the Internet [7]. In contrast to the TCP Reno algorithm [12], 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 (congestion window) in an attempt to keep a small number of packets buffered in the routers along the path.

Although experimental results presented in [7] and [2] show that TCP Vegas achieves better throughput and fewer losses than TCP Reno under many scenarios, at least two concerns remained: is Vegas stable, and if so, does it stabilize to a fair distribution of resources; and does Vegas result in persistent congestion. In short, Vegas has lacked a theoretical explanation of why it works.

*The first author acknowledges the support of the Australian Research Council through grants S499705, A49930405 and S4005343, 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.

ACM SIGMETRICS 2001 6/01 Cambridge, MA USA

This paper addresses this shortcoming by presenting a model of Vegas as a distributed optimization algorithm. Specifically, we show that the global objective of Vegas is to maximize the aggregate utility of all sources (subject to the capacity constraints of the network's resources), and that the sources solve the dual of this maximization problem by implementing an approximate gradient projection algorithm. This model implies that Vegas stabilizes at 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.

Our analysis shows that Vegas does have the potential to induce persistent queues (up to the point that Reno-like behavior kicks in), but 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.

The paper concludes by presenting simulation results that both serve to validate the model and to illustrate the impact of this explicit feedback mechanism. All proofs are omitted and can be found in the full version of this paper [21].

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 of capacity c_l, $l \in L$. 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.[1] 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). Note that $D_s(t)$ depends not only on the window $w_s(t)$ of source s, but also on those of all other sources, possibly even those sources that do not share a link with s. We model the change in window size by one packet per round trip time in actual implementation, with a change of $1/D_s(t)$ per discrete time. Thus, source s adjusts its window according to:

Vegas Algorithm:

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

In the original paper [7], $w_s(t)/d_s$ 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.

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

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

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 Vegas.

2.2 Objective of Vegas

We now show that Vegas sources have

$$U_s(x_s) = \alpha_s d_s \log x_s \quad (3)$$

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

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

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

[1]The algorithm in [7] tries to keep this difference between α_s and β_s, with $\alpha_s < \beta_s$ to reduce oscillation. We assume for analytical simplicity that $\alpha_s = \beta_s$. This captures the essence of Vegas. The usual practice of using $\alpha_s < \beta_s$ reduces oscillation by enlarging the equilibrium rate vector from a unique point to a set.

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 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. 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).*

2.3 Vegas Algorithm

We first describe a scaled gradient projection algorithm, adapted from [20], to solve the dual problem of (4–5). Then we interpret the Vegas algorithm as a smoothed version of this algorithm. The convergence of the scaled gradient projection algorithm hence underlies the stability of Vegas.

Associated with each link l is a dual variable $p_l \geq 0$. The dual problem of (4–5) is to choose the dual vector $p = (p_l, l \in L)$ so as to (see [20]):

$$\min_{p \geq 0} \quad D(p) := \sum_s B_s(p^s) + \sum_l p_l c_l \quad (6)$$

where

$$B_s(p^s) = \max_{x_s \geq 0} \; U_s(x_s) - x_s p^s \quad (7)$$

$$p^s = \sum_{l \in L(s)} p_l \quad (8)$$

If we interpret the dual variable p_l as the price per unit bandwidth at link l, then p^s in (8) is the price per unit bandwidth in the path of s. Hence $x_s p^s$ in (7) 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. A vector $p \geq 0$ that minimizes the dual problem (6) is called *dual optimal*. Given a vector price $p = (p_l, l \in L)$ or a scalar price $p^s = \sum_{l \in L(s)} p_l$, we will abuse notation and denote the unique maximizer in (7) by $x_s(p)$ or by $x_s(p^s)$.

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. In case of Vegas with its particular utility function, the link price p_l turns out to be the *queueing* delay at link l; see Section 3. 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.

A scaled gradient projection algorithm to solve the dual problem takes the following form [20]. Let $p_l(t)$ be the link price at time t and $p^s(t) = \sum_{l \in L(s)} p_l(t)$ denote the path price of source s at time t. Then at time t source s sets its rate to (setting derivative of $U_s(x_s(t)) - x_s(t)p^s(t)$ to zero):

$$x_s(t) = x_s(p^s(t)) = \frac{\alpha_s d_s}{p^s(t)} \quad (9)$$

Let $x^l(p(t)) = \sum_{s \in S(l)} x_s(p(t))$ denote the aggregate source rate at link l. Link l computes $p_l(t)$ according to:

$$p_l(t+1) \quad = \quad [p_l(t) + \gamma\theta_l(x^l(p(t)) - c_l)]^+ \qquad (10)$$

where $\gamma > 0$ and $\theta_l > 0$ are constants.

The following result says that the scaled gradient projection algorithm defined by (9–10) converges to yield the unique optimal source rates, and underlies the stability of Vegas.

THEOREM 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 (10–9) is primal-dual optimal.*

We now interpret the Vegas algorithm as approximately carrying out the scaled gradient projection algorithm (9–10). 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 now show that, first, the price computation (10) is performed by the buffer process at link l. 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 a *smoothed* version of (9).

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 $x^l(t) = \sum_{s \in S} x_s(t)$, and the buffer occupancy $b_l(t)$ at link l evolves according to:

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

Dividing both sides by c_l we have

$$\frac{b_l(t+1)}{c_l} \quad = \quad \left[\frac{b_l(t)}{c_l} + \frac{1}{c_l}(x^l(t) - c_l)\right]^+ \qquad (11)$$

Identifying $p_l(t) = b_l(t)/c_l$, we see that (11) is the same as (10) 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 (9), as explained next.

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

$$w_s(t) - x_s(t)d_s < \alpha_s d_s \quad \text{or} \quad w_s(t) - x_s(t)d_s > \alpha_s d_s \qquad (12)$$

From the proof of Theorem 1, this quantity is related to the backlog, and hence the prices, in the path:

$$w_s(t) - x_s(t)d_s \quad = \quad x_s(t)\; p^s(t) \qquad (13)$$

Thus, the conditional in (12) becomes (cf. (9)):

$$x_s(t) \;<\; \frac{\alpha_s d_s}{p^s(t)} \quad \text{or} \quad x_s(t) \;>\; \frac{\alpha_s d_s}{p^s(t)} \qquad (14)$$

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 (9) sets the rate to the target rate in one step.

3. DELAY, FAIRNESS AND LOSS

3.1 Delay

The previous section developed 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 (10) and (11)):

$$p_l(t) \quad = \quad \frac{b_l(t)}{c_l} \qquad (15)$$

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 (9) 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 (9) 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:

$$x_s^* p^{*s} = \alpha_s d_s \qquad (16)$$

This is just Little's Law in queueing theory. The relation (16) then implies that queueing delay must increase 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 [7], 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, Theorem 1 implies that the equilibrium rates x^* are *weighted proportionally fair* [13, 15]: for

any other feasible rate vector x, we have

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

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. The the utility functions and the equilibrium rates are weighted proportional fair, with weights proportional to sources' propagation delays. (16) 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* [14, 16, 19]:

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

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., [8, 10, 18, 22, 5].

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 [7] and [2].

As observed in [7] and [5], 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 [11] 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 [11], 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 7.4 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) [4], in the form of the recently proposed ECN bit [9, 26].

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 state of the network: topology, link capacities, number of sources, and their utility functions. As the number of sources increases the equilibrium prices, and hence the equilibrium backlog, increases (since $b_l^* = p_l^* c_l$). This not only necessitates large buffers in the network, but worse still, it leads to large feedback delay and possibly oscillation. Indeed, 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 [23]. 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.[2] 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

[2]A remedy is suggested for the first problem in [23] 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.

$0 < \hat{d}_s(t) \leq D_s(t)$. The next result says that the estimation error effectively changes the utility function from (3) to:

$$U_s(x_s) \quad = \quad (1+\epsilon_s)\alpha_s d_s \log x_s + \epsilon_s d_s x_s \qquad (17)$$

THEOREM 3. *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. 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 (17).*

The significance of Theorem 3 is twofold. First, it implies that incorrect propagation delay does not upset the stability of Vegas algorithm— the rates simply converge to 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.

Although Vegas can be stable in the presence of error in propagation delay estimation, the error may cause two problems. First, overestimation 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. These are illustrated in the simulations in Section 7.2 below.

4.3 Remarks

We did not see persistent congestion in our original simulations of Vegas. This is most likely due to three factors. First, 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 [25]. 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) [4] 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. It is hence essential that the active queue management scheme strives to clear the buffer *while maintaining high utilization*. 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 (9), except that path price now is no longer the round trip delay. Instead it is computed and fed back using the REM algorithm, explained below. When packet loss is detected through duplicate acknowledgments, Vegas halves its window, as Reno would. The purpose of AQM is not to replace a loss signal due to buffer overflow by a 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 queue, 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 allows sources to estimate their path prices from observed dropping or marking rate. We now summarize REM; see [4, 3] 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:

$$p_l(t+1) \quad = \quad [p_l(t) + \gamma(\mu_l b_l(t) + x^l(t) - c_l)]^+ \qquad (18)$$

where $\gamma > 0$ is a small constant and $0 < \mu_l < 1$. This price adjustment (18) leads to small backlog ($b_l^* \simeq 0$) and high utilization ($x^{l*} \simeq c_l$) in equilibrium, regardless of the equilibrium price p_l^*. 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:

$$m_l(t) \quad = \quad 1 - \phi^{-p_l(t)}$$

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.

Source s estimates the 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:

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

where $\log_\phi$ is logarithm to base ϕ. It then adjusts its rate using marginal utility (cf. (9)):

$$x_s(t) \quad = \quad \frac{\alpha_s d_s}{\hat{p}^s(t)} \quad = \quad \frac{\alpha_s d_s}{-\log_\phi(1 - \hat{m}^s(t))} \qquad (19)$$

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 (19)), and decrementing it slightly otherwise, in the spirit of the original Vegas algorithm (1):

Vegas with REM:

$$w_s(t+1) \quad = \quad \begin{cases} w_s(t) + \frac{1}{D_s(t)} & \text{if} \quad -\frac{w_s(t)}{D_s(t)} < \frac{\alpha_s d_s}{\hat{p}^s(t)} \\ w_s(t) - \frac{1}{D_s(t)} & \text{if} \quad -\frac{w_s(t)}{D_s(t)} > \frac{\alpha_s d_s}{\hat{p}^s(t)} \\ w_s(t) & \text{else} \end{cases}$$

6. RELATED WORK

The optimal flow control problem (4–5) is formulated in [13]. It is solved using a penalty function approach in [15, 16] and extended in [24] and [17]. The problem is solved using a duality approach in [20] leading to an abstract algorithm whose convergence has been proved in asynchronous environment.

A practical implementation of this algorithm is studied in [4]. 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 [19] to other schemes, such as Reno/DropTail, Reno/RED, and Reno/REM. The utility functions of these schemes are derived.

In [24] 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 [24] 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 1 is first proved in [24]; our proof has a somewhat more intuitive presentation.

A single-link dynamical system model of Vegas is used in [5] 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 can only decrease over time.

A model of Vegas with a single link, two sources and $\alpha < \beta$ is used in [23] 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 PF 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 [6] to investigate the effect of persistent congestion on fairness both in the case when $\alpha = \beta$ and $\alpha < \beta$. They conclude that overestimation 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.

7. VALIDATION

This section presents four sets of simulation results. The first set shows that source rate converges quickly under Vegas to the theoretical equilibrium, 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 Vegas to work properly.

We use the *ns-2* network simulator [1] configured 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. Simulations with multiple links are presented in [21].

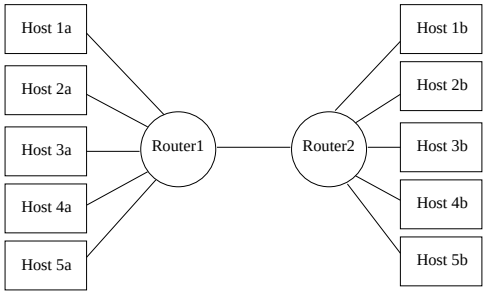


Figure 1: Network Topology

7.1 Experiment 1: Equilibrium and Fairness

We run five connections across the network (i.e., between Host1a and Host1b, 2a and 2b etc.) to understand how they compete for bandwidth. The round trip latency for the connections are 15ms, 15ms, 20ms, 30ms and 40ms respectively. The shared link has a bandwidth of 48Mbps and all host–router links have a bandwidth of 100Mbps. Routers maintain a FIFO queue with unlimited capacity.

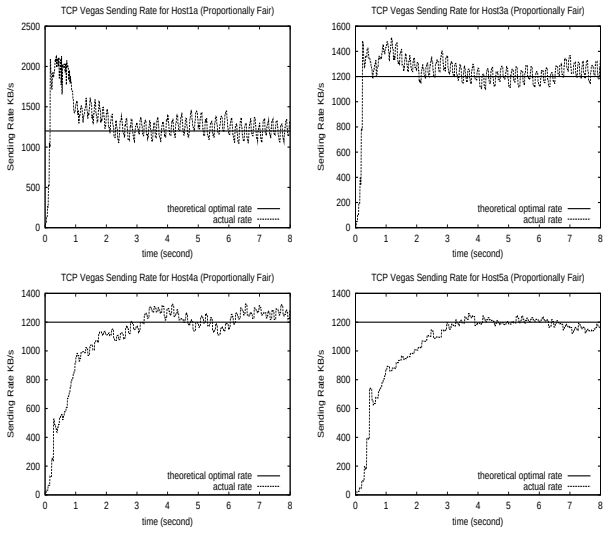
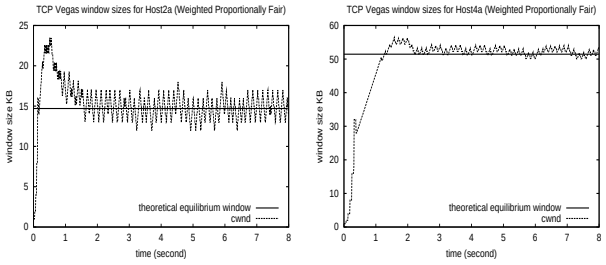


Figure 2: Experiment 1(a), PF: sending rates (Host2a, not shown here, behaved similarly to Host1a.)

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 sending rate against the predicted rates (straight lines): all connections quickly converge to the predicted rate. Table 1 summarizes other performance values,[3] which further demonstrate how well the model predicts the simulation.



(a) Host2a delay=15ms (b) Host4a delay=30ms

Figure 3: Experiment 1(b), (WPF): congestion window size for two (of the five) connections

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 for the two (of the five) connections are shown in Figure 3, except this time we show the congestion windows instead of the sending rates. The other performance numbers are in Table 2, which again show that the simulations closely follow the model's predictions. All simulation numbers are averages in equilibrium.

We repeated these simulations with multiple links using the topology in [7] and the measurements agree well with model predictions; see [21].

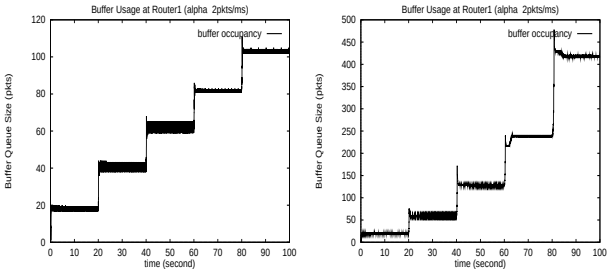
7.2 Experiment 2: Persistent Congestion

We next validate that Vegas leads to persistent congestion under pathological conditions. We set the round trip latency 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 4(a), the buffer occupancy increases linearly in the number of sources.

Next, we take propagation delay estimation error into account by letting the Vegas sources discover the propagation

[3]The reported `baseRTT` includes both the round trip latency and transmit time.



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

Figure 4: Experiment 2: buffer occupancy at router

baseRTT (ms)	$Host1a$	$Host2a$	$Host3a$	$Host4a$	$Host5a$
no error	10.18	10.18	10.18	10.18	10.18
w/ error (S)	10.18	13.36	20.17	31.5	49.86
w/ error (M)	10.18	13.51	20.18	31.2	49.80

Table 4: Experiment 2: error in propagation delay estimation

delay for themselves. As shown in Figure 4(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 3 to calculate the equilibrium rates, queue size, and `baseRTT` (estimated propagation delay). The predicted and measured numbers are shown in Tables 3 and 4. They match very well, further verifying our model.

As Table 3 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, $x_1 : x_2 : x_3 : x_4 : x_5 = 1 : 1.4 : 2.3 : 4.5 : 11.6$.

7.3 Experiment 3: Vegas + REM

Finally, we implement REM at Router1, which updates link price every 1ms according to (18). 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 7.2. 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: $\phi = 1.1$, $\mu_l = 0.5$, $\gamma = 0.005$.

We start 5 connections with an inter-start interval of 20s in order to test our claim that REM reduces the estimation error in Vegas' propagation delay. Figure 5 plots the congestion window size of the five connections and buffer occu-

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

Host	*1*a		*2*a		*3*a		*4*a		*5*a	
	M	S	M	S	M	S	M	S	M	S
baseRTT (ms)	15.34	15.34	15.34	15.34	20.34	20.34	30.34	30.34	40.34	40.34
RTT w/ queueing (ms)	17	17.1	17	17.1	22	21.9	32	31.9	42	41.9
Sending rate (KB/s)	1200	1205	1200	1183	1200	1228	1200	1247	1200	1161
Congestion window (pkts)	20.4	20.5	20.4	20.2	26.4	27	38.4	39.9	50.4	49.8
Buffer occupancy at Router1 (pkts)	Model				Simulation					
	10				9.8					

Table 1: Experiment 1(a), (PF): comparison of theoretical and simulation results. M-Model, S-Simulation.

Host	*1*a		*2*a		*3*a		*4*a		*5*a	
	M	S	M	S	M	S	M	S	M	S
baseRTT (ms)	15.34	15.34	15.34	15.34	20.34	20.34	30.34	30.34	40.34	40.34
RTT w/ queueing (ms)	19.4	19.55	19.4	19.58	24.4	24.4	34.4	34.3	44.4	44.3
Sending rate (KB/s)	756.3	781	756.3	774	1003	994	1496	1495	1990	1975
Congestion window (pkts)	14.7	15.1	14.7	14.9	24.5	24.6	51.5	51.7	88.4	88.6
Buffer occupancy at Router1 (pkts)	Model				Simulation					
	24.34				24.24					

Table 2: Experiment 1(b), (WPF): comparison of theoretical and simulation results. M-Model, S-Simulation.

pancy at Router1. As expected, each of the five connections converges to its appropriate bandwidth share. When the link is not congested, source rate oscillates more severely, as seen from Host1a during time 0 - 20s. This is a consequence of the log utility function; see [4]. As more sources become active (40 - 100s), oscillation becomes smaller and convergence faster. REM eliminates the superlinear growth in queue length of Figure 4(b) while maintaining high link utilization (90% to 96%). As a result propagation delay can be accurately estimated, as shown in Table 5.

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 5: Experiment 3: comparison of `baseRTT` in Vegas+REM.

7.4 Experiment 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 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.

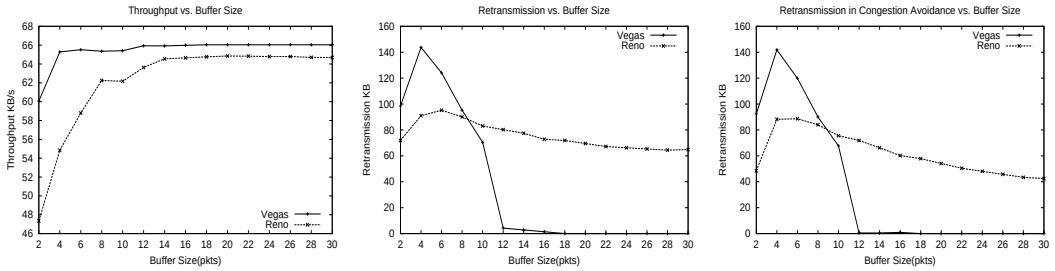
In [11], TCP Vegas is decomposed into several individual mechanisms and the effect of each on performance is assessed by taking the approach of a 2^k factorial design with replications. This work deploys a very useful methodology and gives us some insights into the relative importance of different algorithms 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 [11], which is similar to Figure 1, but the bottleneck link has a capacity of 200 KB/sec and a latency of 50ms, and host-router connections 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 and only start three persistent FTP transfers from Host1a to Host1b. 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 respectively for Vegas. Figure 6 shows the average throughput, retransmission and retransmission during congestion avoidance of the three flows as a function of buffer size at Router1. These plots confirms that Vegas has a steady send rate and no retransmissions as long as the buffer sizes exceeds a threshold. The threshold is a bit larger than the total Vegas' backlog (3–9 packets in this case) because of queue fluctuations during transient. In contrast, increasing the buffer size continuously helps the performance of Reno though retransmission remains significant even at large buffer size.

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.

Returning to the original question of whether the recovery mechanism or the congestion avoidance mechanism plays a bigger role in performance, it is worth noting that there is a danger in trying to isolate their respective contributions.

Time	1a (KB/s)		2a (KB/s)		3a (KB/s)		4a (KB/s)		5a (KB/s)		Queue (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 3: Experiment 2(b): equilibrium rates and queue lengths with propagation delay error. M-Model, S-Simulation



(a) Throughput (b) Retransmit (c) Retransmit in CA

Figure 6: Experiment 4: effect of buffer capacity on performance. All numbers are averaged on three long lasting flows over a 50 second period. For Vegas, $\alpha = 1$, $\beta = 3$ packet per RTT.

This is because 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.

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 the equilibrium rates are weighted proportionally fair. Third, it clarifies the mechanism, and consequence, 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.

Acknowledgment. We gratefully acknowledge helpful discussions with Sanjeewa Athuraliya.

9. REFERENCES

[1] NS network simulator. Available via `http://www.isi.edu/nsnam/ns/`.

[2] J. S. Ahn, P. B. Danzig, Z. Liu, and L. Yan. Evaluation of TCP Vegas: emulation and experiment. In *Proceedings of SIGCOMM'95*, 1995.

[3] Sanjeewa Athuraliya, Victor H. Li, Steven H. Low, and Qinghe Yin. REM: Active Queue Management (extended version). Submitted for publication, `http://netlab.caltech.edu`, October 2000.

[4] Sanjeewa Athuraliya and Steven H. Low. Optimization flow control, II: Implementation. Submitted for publication, `http://netlab.caltech.edu`, May 2000.

[5] Thomas Bonald. Comparison of TCP Reno and TCP Vegas via fluid approximation. In *Workshop on the Modeling of TCP*, December 1998. Available at `http://www.dmi.ens.fr/\%7Emistral/tcpworkshop.html`.

[6] C. Boutremans and J. Y. Le Boudec. A note on the fairness of tcp vegas. In *Proceedings of International Zurich Seminar on Broadband Communications*, pages 163–170, February 2000.

[7] Lawrence S. Brakmo and Larry L. Peterson. TCP Vegas: end to end congestion avoidance on a global Internet. *IEEE Journal on Selected Areas in Communications*, 13(8), October 1995. Available at `http://cs.princeton.edu/nsg/papers/jsac-vegas.ps`.

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

[9] S. Floyd. TCP and Explicit Congestion Notification. *ACM Computer Communication Review*, 24(5), October 1994.

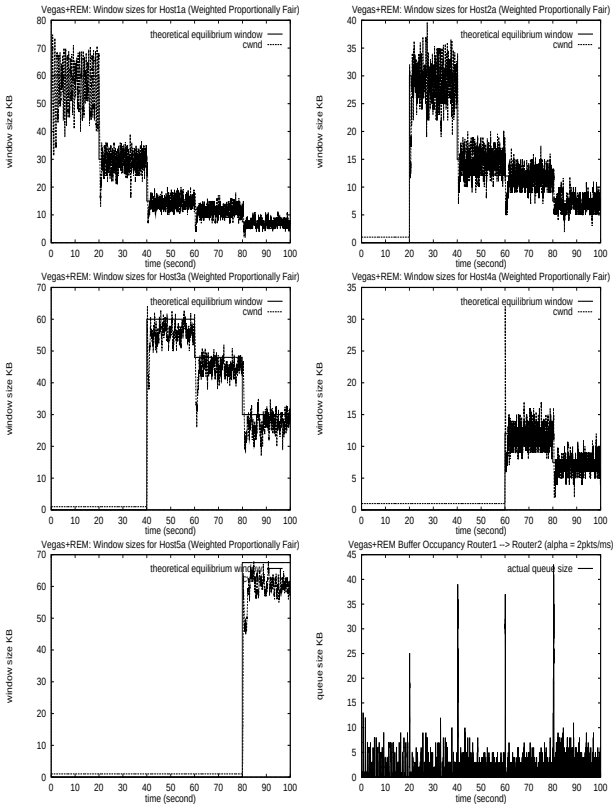


Figure 5: Experiment 3: Vegas+REM. Link utilization: 90%(0-20s), 96%(20-40s), 93%(40-60s), 91%(60-80s), 90%(80-100s)

[10] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Trans. on Networking*, 1(4):397–413, August 1993. Available at `ftp://ftp.ee.lbl.gov/papers/early.ps.gz`.

[11] U. Hengartner, J. Bolliger, and T. Gross. TCP Vegas revisited. In *Proceedings of IEEE Infocom*, March 2000.

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

[13] Frank P. Kelly. Charging and rate control for elastic traffic. *European Transactions on Telecommunications*, 8:33–37, 1997. `http://www.statslab.cam.ac.uk/~frank/elastic.html`.

[14] Frank P. Kelly. Mathematical modelling of the Internet. In *Proc. 4th International Congress on Industrial and Applied Mathematics*, July 1999. Available at `http://www.statslab.cam.ac.uk/~frank/mmi.html`.

[15] Frank P. Kelly, Aman Maulloo, and David Tan. Rate control for communication networks: Shadow prices, proportional fairness and stability. *Journal of Operations Research Society*, 49(3):237–252, March 1998.

[16] Srisankar Kunniyur and R. Srikant. End–to–end congestion control schemes: utility functions, random losses and ECN marks. In *Proceedings of IEEE Infocom*, March 2000. Available at `http://www.ieee-infocom.org/2000/papers/401.ps`.

[17] Richard La and Venket Anantharam. Charge-sensitive TCP and rate control in the Internet. In *Proceedings of IEEE Infocom*, March 2000. Available at `http://www.ieee-infocom.org/2000/papers/401.ps`.

[18] T. V. Lakshman and Upamanyu Madhow. The performance of TCP/IP for networks with high bandwidth–delay products and random loss. *IEEE/ACM Transactions on Networking*, 5(3):336–350, June 1997. Available at `http://www.ece.ucsb.edu/Faculty/Madhow/Publications/ton97.ps`.

[19] Steven H. Low. A duality model of TCP flow controls. In *Proceedings of ITC Specialist Seminar on IP Traffic Measurement, Modeling and Management*, September 18-20 2000. `http://netlab.caltech.edu`.

[20] Steven H. Low and David E. Lapsley. Optimization flow control, I: basic algorithm and convergence. *IEEE/ACM Transactions on Networking*, 7(6):861–874, December 1999. `http://netlab.caltech.edu`.

[21] Steven H. Low, Larry Peterson, and Limin Wang. Understanding Vegas: a duality model (full version). Submitted for publication, `http://netlab.caltech.edu/pub.html`, February 2000.

[22] Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi, and Teunis Ott. The macroscopic behavior of the TCP congestion avoidance algorithm. *ACM Computer Communication Review*, 27(3), July 1997. Available at `http://www.psc.edu/networking/papers/model_ccr97.ps`.

[23] J. Mo, R. La, V. Anantharam, and J. Walrand. Analysis and comparison of TCP Reno and Vegas. In *Proceedings of IEEE Infocom*, March 1999.

[24] Jeonghoon Mo and Jean Walrand. Fair end-to-end window-based congestion control. Technical Report Tech. Rep. M98/46, UCB/ERL, UC Berkeley, 1998.

[25] V. Paxson. End-to-end routing behavior in the Internet. In *Proceedings of SIGCOMM'96, ACM*, August 1996.

[26] K. K. Ramakrishnan and S. Floyd. A Proposal to add Explicit Congestion Notification (ECN) to IP. RFC 2481, January 1999.