Understanding TCP Vegas: A Duality Model [*]

Steven Low
Departments of CS and EE, Caltech, USA
slow@caltech.edu
Larry Peterson Limin Wang
Department of CS, Princeton University, USA
$\{$llp,lmwang$\}$@cs.princeton.edu

Technical Report TR-616-00
Updated November 17, 2000

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. In addition to presenting the model and exploring these three issues, the paper presents 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 [12]. In contrast to the TCP 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 (congestion window) in an attempt to keep a small number of packets buffered in the routers along the transmission 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.

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

[*]The first author acknowledges the support of the Australian Research Council through grants S499705, A49930405 and S4005343, and the second author acknowledges the support of NSF through Grant ANI-9906704.

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 packet(s) 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 explicit feedback—for example, in the form of the recently proposed ECN bit [25]—it is possible to avoid this problem. Explicit feedback 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. Models of Vegas are also analyzed in [6, 22] using a different framework.

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 primary 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 source s's own window $w_s(t)$ 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:

[1]The actual algorithm in [7] tries to keep this difference between α_s and β_s, with $\alpha_s < \beta_s$ to reduce oscillation. Our model assumes $\alpha_s = \beta_s$. It is simpler and captures the essence of Vegas.

Vegas Algorithm:

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

In the original paper [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

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

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

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

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

$$\max_{x \geq 0} \qquad \sum_s U_s(x_s) = \sum_s \alpha_s d_s \log x_s \tag{4}$$

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

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 *socially 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.

The following theorem clarifies the objective of Vegas. It was first proved in [23].

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 (4–5).*

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

Solving the primal problem (4–5) directly is impractical over a large network since it requires coordination among all sources due to coupling through shared links. However, a distributed solution can be obtained by appealing to duality theory, a standard technique in mathematical programming. In this subsection, we briefly present the dual problem of (4–5), interpret it in the context of congestion control, and derive a scaled gradient projection algorithm to solve it. A more detailed description can be found in [20] for general utility functions. In the next subsection, we interpret the Vegas algorithm (1) as a smoothed version of the scaled gradient projection algorithm.

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

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

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. A vector $p \geq 0$ that minimizes the dual problem (8) 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 (9) by $x_s(p)$ or by $x_s(p^s)$. A feasible rate vector $x(p) = (x_s(p), s \in S)$ is called *individually optimal* (with respect to p) when each individual rate $x_s(p)$ minimizes (9). By duality theory, there exists a dual optimal price $p^* \geq 0$ such that these individually optimal rates $x^* = (x_s(p^*), s \in S)$ are also socially optimal, that is, solve (4–5) as well.

In the rest of the paper we will refer to p_l as link price, p^s 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 $x_s(p(t))$ denote the unique source rate that maximizes (9–10) with p replaced by $p(t)$, and $x^l(p(t)) = \sum_{s \in S(l)} x_s(p(t))$ denote the aggregate source rate at link l. Then link l computes $p_l(t)$ according to:

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

where $\gamma > 0$ and $\theta_l > 0$ are constants. Here $x^l(p(t))$ represents the demand for bandwidth at link l and c_l represents the supply. The price is adjusted according to the law of demand and supply: if demand exceeds the supply, raise the price; otherwise reduce it.

Let $p^s(t) = \sum_{l \in L(s)} p_l(t)$ denote the path price at time t. Then source s sets its rate to the unique maximizer of (9–10) given by (setting the derivative of $U_s(x_s) - x_s p^s(t)$ to zero):

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

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.

The following result says that the scaled gradient projection algorithm defined by (11–12) converges to yield the unique optimal source rates. It is a minor modification of Theorem 1 of [20]; indeed the convergence proof in [3] for a (different) scaled gradient projection algorithm applies directly here.

Theorem 2 *Provided that the step-size 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.*

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 link l. Indeed, link price can be taken as the queueing delay , $p_l(t) = b_l(t)/c_l$, where $b_l(t)$ denotes the buffer occupancy at link l at time t. 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 (12).

Specifically, suppose the input rate at link l from source s is $x_s(t)$ at time t.[2] 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:

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

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).

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 for (7) 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)):

$$x_s(t) \;<\; \frac{\alpha_s d_s}{p^s(t)} \quad \text{or} \quad x_s(t) \;>\; \frac{\alpha_s d_s}{p^s(t)} \tag{16}$$

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 directly to the target rate.

The sufficient condition in Theorem 2 requires that the stepsize $\gamma > 0$ be sufficiently small to guarantee convergence. The original Vegas algorithm however assumes that $\gamma = 1$; see (13). We now describe a way

[2] This is an approximation which holds in equilibrium when buffer stabilizes; see [18] for a more accurate model of the buffer process.

to reintroduce 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

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

that is, by using *weighted* queueing delays as prices, they are updated with a stepsize γ that is not necessarily one. Then (15) is modified to

$$\gamma(w_s(t) - x_s(t)d_s) \quad = \quad x_s(t) \sum_{l \in L(s)} \gamma \frac{b_l(t)}{c_l} \quad = \quad x_s(t)\, p^s(t) \tag{17}$$

Since the modification should not alter the utility functions nor the equilibrium rates, $x_s(t)$ should still be adjusted according to (16) so that, in equilibrium, $p^{*s} = \alpha_s d_s / x_s^*$. This together with (17) modifies the 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 a α_s that is $1/\gamma$ times larger, i.e., use a unit of 10KBps (say) instead of KBps for α_s.[3] Note that γ (or unit of α_s) should be the same at all sources.

Smaller γ ensures convergence of source rates, albeit slower, but it leads to a larger backlog since $b_l(t) = c_l p_l(t)/\gamma$. This dilemma 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 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 optimization model suggests a third interpretation. The dynamics of the buffer process at link l implies the relation (comparing (11) and (13)):

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

It says that the link price $p_l(t)$ is just 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

[3] Using a smaller link capacity, say, Mbps instead of 10Mbps, has the same effect.

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.

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:

$$x_s^* p^{*s} = \alpha_s d_s \tag{18}$$

This is just Little's Law in queueing theory when propagation delay is ignored. As the number of sources increases, individual source rates necessarily decrease. The relation (18) then implies that queueing delay $p^s(t)$ 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

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

The first implementation has $\alpha_s = \alpha / d_s$ inversely proportional to the source's propagation delay, and the second has identical $\alpha_s = \alpha$ for all sources.

These two implementations lead to different fairness in equilibrium. When $\alpha_s d_s = \alpha$ (in unit of packets) are the same for all sources, 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).

When $\alpha_s = \alpha$ are identical, sources have different utility functions, and the equilibrium rates are weighted proportional fair, with weights proportional to sources' propagation delays. (18) 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:

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

In a network with multiple congested links, weighting the utility by propagation delay has a balancing effect to the discrimination against long connections, 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 constrasts with TCP Reno which attempts to equalize *window* [14, 16, 19]:

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

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, 17, 21, 6].

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 since the aggregate source rate $\sum_{s \in S(l)} x_s^*$ is no more than the link capacity c_l in the network (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 [6], 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 are reported that 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 6.4 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, 25].

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 [22]. 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.[4] 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 $-1 < \epsilon_s \le D_s(t)/d_s(t) - 1$ for all t so that the estimate satisfies $0 < \hat{d}_s(t) \le D_s(t)$. The next result says that the estimation error effectively changes the utility function: source s appears to have a utility (cf. (3))

$$U_s(x_s) \;=\; (1+\epsilon_s)\alpha_s d_s \log x_s + \epsilon_s d_s x_s \tag{19}$$

and the objective of the Vegas sources appears to

$$\max_{x \ge 0} \quad \sum_s U_s(x_s) \;=\; \sum_s (1+\epsilon_s)\alpha_s d_s \log x_s + \epsilon_s d_s x_s \tag{20}$$

$$\text{subject to} \quad \sum_{s \in S(l)} x_s \;\le\; c_l, \quad l \in L \tag{21}$$

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 (20–21).*

Proof. The argument follows the proof of Theorem 1, except that (6) is replaced by

$$U_s'(x_s^*) \;=\; \frac{(1+\epsilon_s)\alpha_s d_s}{x_s^*} + \epsilon_s d_s \;=\; \sum_{l \in L(s)} p_l^* \tag{22}$$

To show that the equilibrium backlog at the links provide 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 (2) to get

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

[4] A remedy is suggested for the first problem in [22] 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.

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

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

This yields (22) upon identifying $p_l^* = \frac{b_l^*}{c_l}$ and rearranging terms. As in the proof of Theorem 1, x^* must be feasible and the complementary slackness condition must be satisfied. Hence the proof is complete. ■

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 that optimizes (20–21). 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:

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

With error, their rates are related by

$$\frac{(1+\epsilon_r)\alpha_r d_r}{x_r^*} + \epsilon_r d_r \quad = \quad \frac{(1+\epsilon_s)\alpha_s d_s}{x_s^*} + \epsilon_s d_s \tag{23}$$

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, 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.

4.3 Remarks

Note that we did not see persistent congestion in our original simulations of Vegas. This is most likely due to three factors. One is that Vegas reverts to Reno-like behavior when there is insufficient buffer capacity in the network. The second is that 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 [24]. The third is that the situation of connections starting up serially is pathological. In practice, connections continually come and go, meaning that all sources are likely to measure a baseRTT that represents the propagation delay plus the average queuing delay.

5 Vegas with REM

As explained in the last section, excessive backlog may arise because 1) each source maintains some extra packets buffered in its path and hence backlog increases as the number of sources increases, and 2) overestimation of a source's propagation delay distorts the utility, leading to larger equilibrium prices and backlogs (as well as unfairness to older sources). Fundamentally, both are consequences of Vegas' reliance on *queueing* delay as a congestion measure, which makes backlog indispensible in conveying congestion to the sources. This section demonstrates how REM (Random Exponential Marking) [4] can be used to correct this situation.

REM is an active queue management scheme like RED [10] that feeds back congestion information to sources by probabilistically dropping or marking packets. Unlike RED, REM attempts to match rate and clear buffer, leading to high utilization with negligible delay or buffer overflow. With buffer cleared, 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 path price must be estimated by the source from packet dropping or marking. We now summarize REM; see [4] for derivation, 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:

$$p_l(t+1) \quad = \quad [p_l(t) + \gamma(\mu_l b_l(t) + x^l(t) - c_l)]^+ \tag{24}$$

where $\gamma > 0$ is a small constant and $0 < \mu_l < 1$. 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. Note that the algorithm does not require per–flow information. Moreover, the price adjustment (24) 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^*. Hence high utilization is not achieved by maintaining a large backlog, but by feeding back accurate congestion information for sources to set their rates. This is 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:

$$m_l(t) \quad = \quad 1 - \phi^{-p_l(t)} \tag{25}$$

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 acknowledgement.

The exponential form is critical for multilink 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

$$m^s(t) \quad = \quad 1 - \prod_{l \in L(s)} (1 - m_l(t)) \; = \; 1 - \phi^{-p^s(t)} \tag{26}$$

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 (26):

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

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

$$x_s(t) \;=\; \frac{\alpha_s d_s}{\hat{p}^s(t)} \;=\; \frac{\alpha_s d_s}{-\log_\phi(1-\hat{m}^s(t))} \tag{27}$$

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 (27)), 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 Evaluation

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 mostly 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.

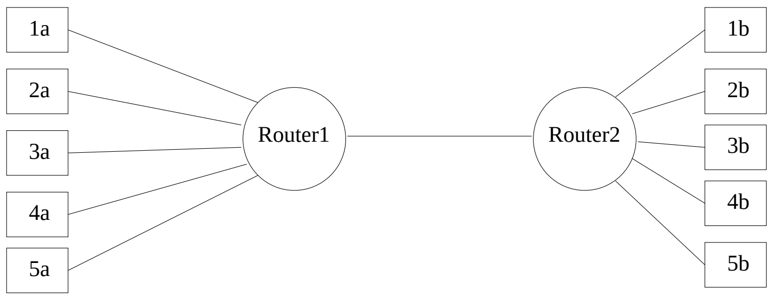


Figure 1: Network Topology

6.1 Equilibrium and Fairness

6.1.1 Single Link Case

We first 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,

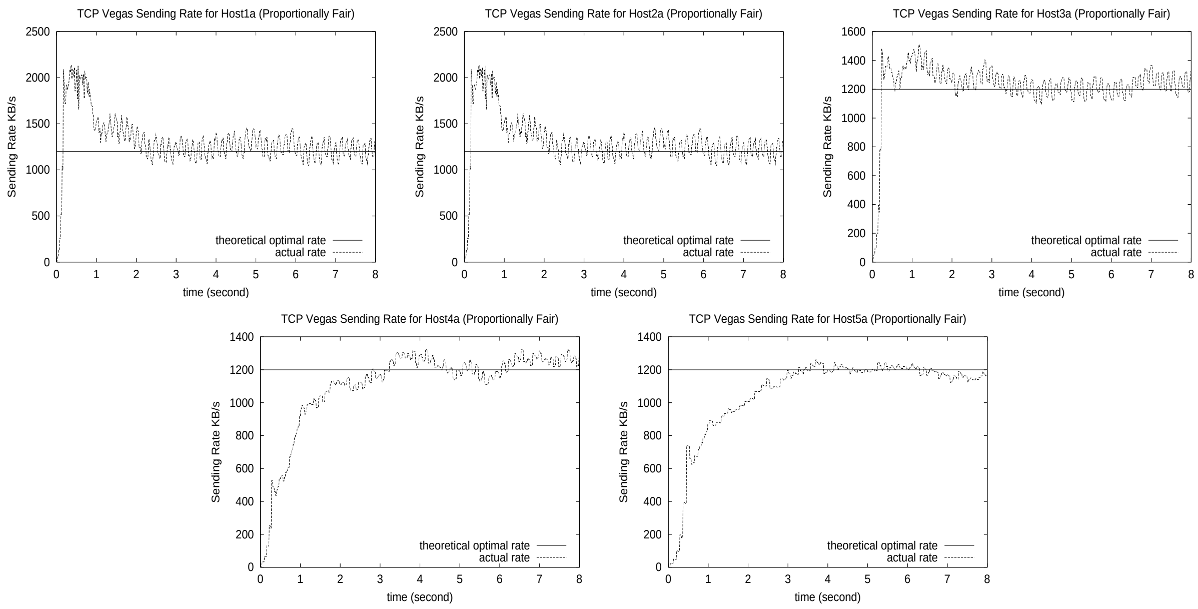


Figure 2: Stability (PF): sending rate of five connections

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.

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,[5] which further demonstrate how well the model predicts the simulation.

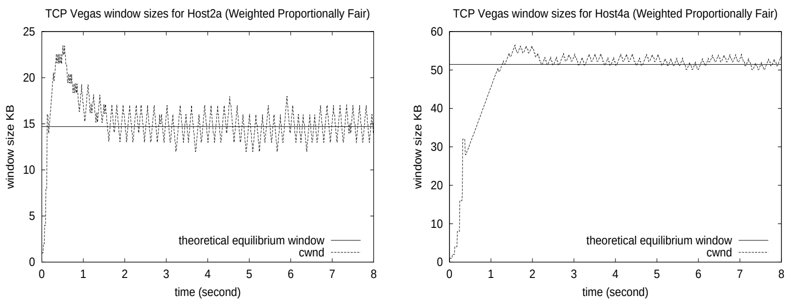
Host	*1*a		*2*a		*3*a		*4*a		*5*a	
	M	S	M	S	M	S	M	S	M	S
baseRTT (ms)	15.34	15.34	15.34	15.34	20.34	20.34	30.34	30.34	40.34	40.34
RTT w/ queueing (ms)	17	17.1	17	17.1	22	21.9	32	31.9	42	41.9
Sending rate (KB/s)	1200	1205	1200	1183	1200	1228	1200	1247	1200	1161
Congestion window (pkts)	20.4	20.5	20.4	20.2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

Table 1: Stability (PF): comparison of theoretical and simulation results. M stands for Model and S stands for Simulation. All simulation numbers are averaged at the equilibrium point.

For weighted proportional fairness, we set α_s to 2 packets *per 10ms*, which means each source will

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

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.



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

Figure 3: Stability (WPF): congestion window size for two (of the five) connections

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	Model					Simulation				
at Router1 (pkts)	24.34					24.24				

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

Both the sending rates (Figure 2) and the congestion windows (Figure 3) *oscillate* around the equilibrium. This is an artifact of setting $\alpha_s = \beta_s$ in our simulations, which we have assumed in the model for simplicity. Vegas adjusts the congestion window by one packet in each round trip time. The adjustment is large relative to $\alpha_s = \beta_s = 2$ packets, rendering the window prone to oscillation. We have repeated the simulation using an α_s that is 10 times as large (corresponding to a stepsize γ 10 times as small). This smoothed out the curves.

6.1.2 Multi-link Case

We also simulated network topology with multiple links as shown in Figure 4. This topology is almost the same as that used in [7], expect that to simplify computation, we set the bandwidth of the "backbone" to be 48Mbps and we now have six sources (HostXa) and six sinks (HostXb). 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.

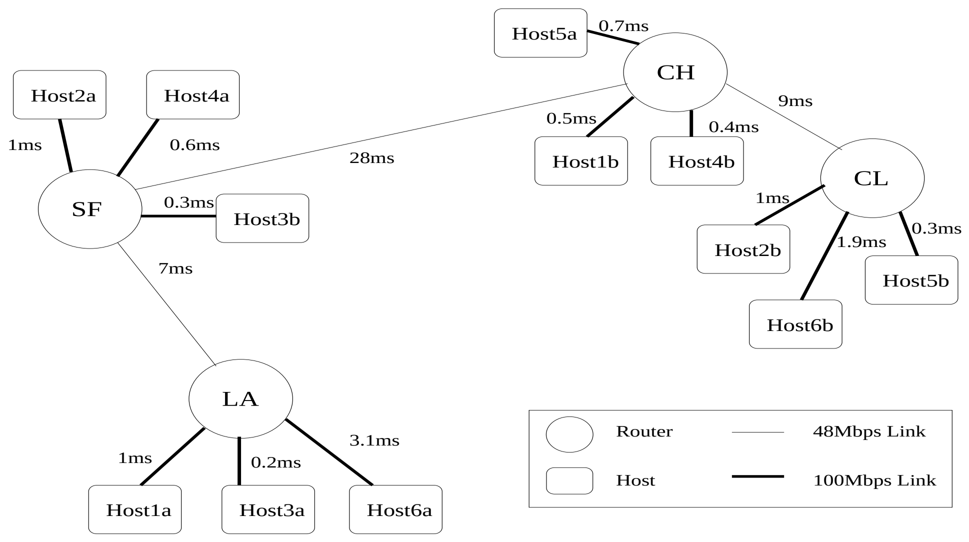


Figure 4: Complex Network Topology

Host	*1*a		*2*a		*3*a		*4*a		*5*a		*6*a	
	M	S	M	S	M	S	M	S	M	S	M	S
baseRTT (ms)	75.51	75.51	80.51	80.51	15.34	15.34	60.34	60.34	20.34	20.34	100.69	100.69
RTT w/ queueing (ms)	76.96	76.98	81.96	81.98	15.89	15.89	61.23	61.22	20.89	20.9	102.69	102.73
Sending rate (KB/s)	1382	1368	1382	1374	3618	3630	2236	2245	3618	3632	1000	980
Congestion window (pkts)	106.35	106.45	113.27	113.7	57.5	58.3	136.93	139	75.6	75.7	102.69	100.6

Buffer occupancy (pkts)	LA		SF		CH	
	M	S	M	S	M	S
	3.32	3.5	5.37	5.51	3.32	3.31

Table 3: Multi-Link (PF): comparison of theoretical and simulation results. M-Model and S-Simulation.

We repeated simulations for Proportionally Fair and Weighed Proportionally Fair cases under this new setup for 20 seconds. At the same time, we use our theory to predict the RTT (including queueing delay), cwnd etc. of the sources and the queue size at the routers. Tables 3 and 4 summarize the results. Again, the simulation measurements match our predictions very well.

6.2 Persistent Congestion

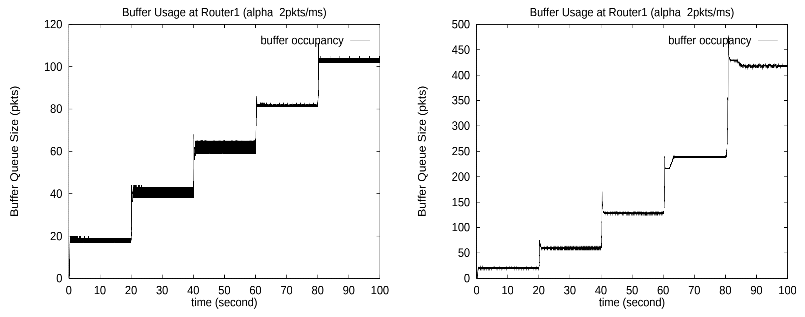
We next validate that Vegas leads to persistent congestion under pathological conditions. We set the round trip latency to 10ms for *all* connections, the host–router links are all 1600 Mbps, and the bottleneck link has a bandwidth of 48 Mbps. We set α_s to 2 packets–per–ms and we assume the routers have infinite buffer capacity. We pick such extreme numbers so as to make the result trend more obvious.

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 5(a), the buffer occupancy increases linearly in the number of sources.

Host	*1*a		*2*a		*3*a		*4*a		*5*a		*6*a	
	M	S	M	S	M	S	M	S	M	S	M	S
baseRTT (ms)	75.51	75.51	80.51	80.51	15.34	15.34	60.34	60.34	20.34	20.34	100.69	100.69
RTT w/ queueing (ms)	85.74	85.77	91.12	91.13	16.4	16.44	69.5	69.51	21.78	21.78	112.35	112.36
Sending rate (KB/s)	1463	1443	1509	1512	2819	2821	1310	1306	2775	2760	1714	1703
Congestion window (pkts)	125.47	125.3	137.5	139	46.24	46.65	91.04	91.6	60.44	60.3	192.6	193

Buffer occupancy (pkts)	LA		SF		CH	
	M	S	M	S	M	S
	6.39	6.38	54.96	55.3	8.65	8.64

Table 4: Multi-Link (WPF): comparison of theoretical and simulation results. M-Model and S-Simulation.



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

Figure 5: Persistent congestion: buffer occupancy at router

Next, we take propagation delay estimation error into account by letting the Vegas sources discover the propagation delay for themselves. As shown in Figure 5(b), buffer occupancy grows much faster than linearly in the number of sources. We have also applied Theorem 3 to calculate the equilibrium rates, queue size, and estimated baseRTT. The predicted and measured numbers are shown in Tables 5 and 6. They match very well, further verifying our model.

As the Table 5 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

Finally, we implement REM at Router1 , which updates link price every 1ms according to (24). 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 phrase; its slow–start behavior stays unchanged.[6]

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 for WPF. In order to verify our new mechanism

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

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 5: Equilibrium rates and queue lengths with propagation delay error. M-Model, S-Simulation

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 (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 6: Error in propagation delay estimation under persistent congestion

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.

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 6 plots the congestion window size of the three connections and buffer occupancy 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 5(b), as shown in Table 7, while maintaining high link utilization (90% to 96%).

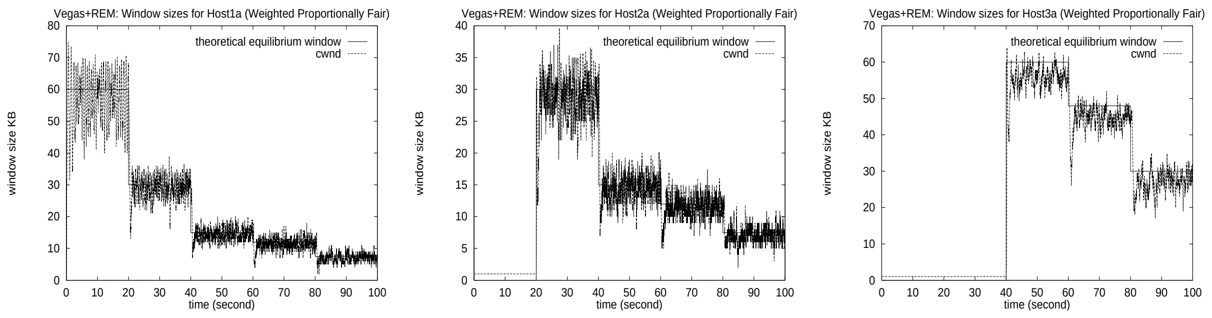
Host	*1*a		*2*a		*3*a		*4*a		*5*a	
	M	S	M	S	M	S	M	S	M	S
baseRTT (ms)	10.18	10.18	10.18	10.18	20.18	20.18	10.18	10.18	30.18	30.19

Table 7: Comparison of baseRTT in Vegas+REM. M – Model, S – Simulation

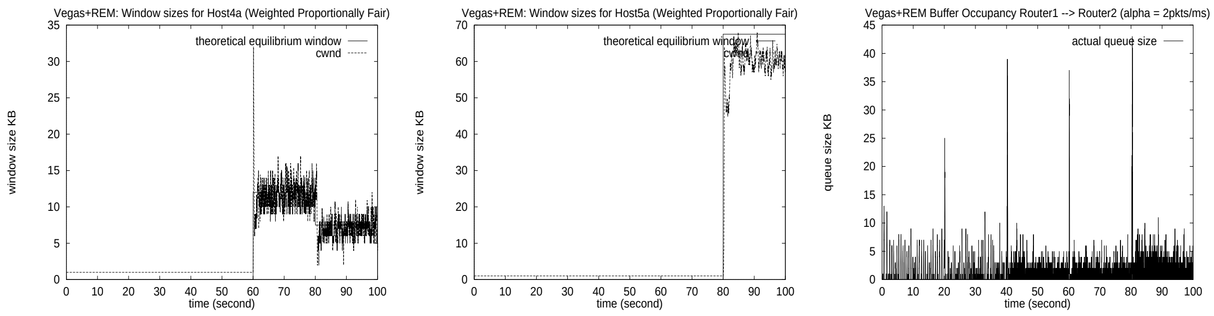
6.4 Effect of buffer capacity

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 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



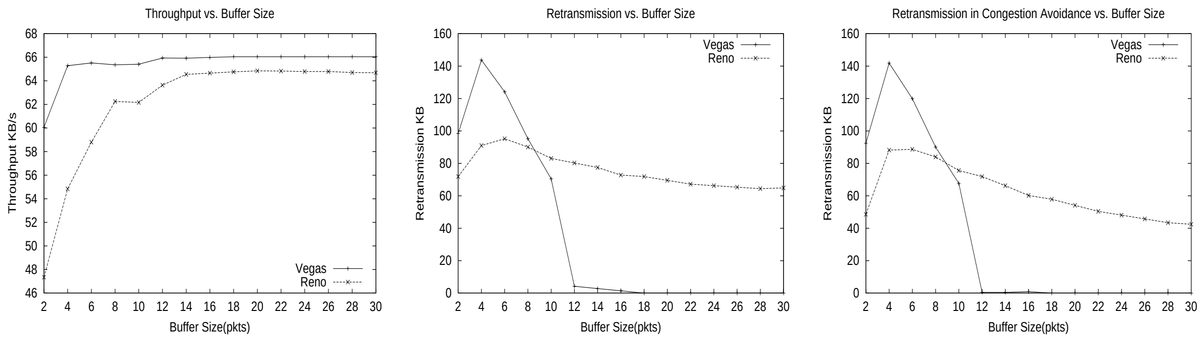
(a) Window Size



(b) Buffer Occupancy

Figure 6: Stability of Vegas+REM. Link utilization: 90%(0-20s), 96%(20-40s), 93%(40-60s), 91%(60-80s), 90%(80-100s)

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. We choose to use such long transfers to 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 7 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



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

Figure 7: 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)

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 there's still large performance advantage over Reno, although not as pronouncing, which can mainly be ascribed to Vegas' congestion avoidance mechanism. When buffer space is small, Vegas' *cwnd* looks like Reno's and that's why Vegas' more aggressive recovery mechanism takes most of the performance credit. We plot Vegas and Reno's *cwnd* for different buffer sizes (to save space, not listed here), and find out when buffer capacity is below the threshold, during congestion avoidance, Vegas behaves much like Reno; but when buffer size exceeds the threshold, Vegas' *cwnd* oscillates around the optimal value.

7 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.

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 and Steven Low. Optimization flow control with Newton–like algorithm. In *Proceedings of IEEE Globecom'99*, December 1999.

[4] Sanjeewa Athuraliya and Steven Low. Optimization flow control, II: Random Exponential Marking. Submitted for publication, `http://www.ee.mu.oz.au/staff/slow/research/`, May 2000.

[5] D. Bertsekas. *Nonlinear Programming*. Athena Scientific, 1995.

[6] 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`.

[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://netweb.usc.edu/yaxu/Vegas/Reference/brakmo.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.

[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] 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`.

[18] Steven H. Low. Optimization flow control with on-line measurement. In *Proceedings of the ITC*, volume 16, June 1999.

[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.

[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://www.ee.mu.oz.au/staff/slow/research/`.

[21] 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`.

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

[23] Jeonghoon Mo and Jean Walrand. Fair end–to–end window–based congestion control. In *Proceedings of SPIE '98 International Symposium on Voice,Video and Data Communications, October 1998*, October 1998.

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

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