Understanding TCP Vegas: Theory and Practice

Steven Low
University of Melbourne

Larry Peterson and Limin Wang
Princeton University

TR 616-00
February 8, 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 [10]. 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 [6] and [1] 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. These concerns are particularly significant in view of evidence that Reno's linear increase, multiplicative decrease algorithm stabilizes around a fair allocation to all connections [23, 12, 13]. 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 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 [22]—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 [5, 18] 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 [6] 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 [6], $w_s(t)/d_s$ 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 [19].

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 [16] 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 [4, 16]:

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

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).

There are two important points to note. First, given scalar prices p^s, each source s can easily solve (9) to obtain the individually optimal source rates $x(p) = (x_s(p^s), s \in S)$ without having to coordinate with any other sources; see (12) below. Second, 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. Furthermore, as we will see below, solution of the dual problem can be distributed to individual links and sources. Hence a better alternative to solving the primal problem (4–5) directly is to solve its dual (8) instead.

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. It can be interpreted in two ways. First, the price p is a congestion measure at the links: the larger the link price p_l, the more severe the congestion at link l. The path price p^s is thus a congestion measure of source s's path. Indeed in the special 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. Second, an *optimal* p^* is a shadow price (Lagrange multiplier) associated with the constrained maximization (4–5); i.e., 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. We emphasize however that p may be unrelated to the actual charge users pay. If sources are indeed charged according to these prices, then p^* aligns individual optimality with social optimality, thus providing the right incentive for sources to choose the optimal rates.

A scaled gradient projection algorithm to solve the dual problem takes the following form [16]. In each iteration t, each link l *individually* updates its own price $p_l(t)$ based on the *aggregate* rate at link l, and each source s *individually* adjusts its rate based on its path price $p^s(t)$.

Specifically 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 [16]; indeed the convergence proof in [2] 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 normalized queue length, $p_l(t) = b_l(t)/c_l$, 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:

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

That is, the backlog $b_l(t+1)$ in the next period is either zero or equals the current backlog $b_l(t)$ plus the total input $x^l(t)$ less the total output c_l in the current period. 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

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

[2]This is an approximation which holds in equilibrium when buffer stabilizes; see [15] for a more accurate model of the buffer process.

As for (7) this quantity is related to the backlog, and hence the prices, in the path:

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

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

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

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

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

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

that is, the prices are updated with a stepsize γ that is not necessarily one. This implies a multiplication of both sides of the first equality of (15) by γ, and hence the comparison in (14) becomes:

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

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.

Finally, we mention in passing that the Vegas algorithm can also be regarded as a Lagrangian method [4, Chapter 4] where the primal variable $x(t)$ and dual variable $p(t)$ are iterated together to solve the Karush–Kuhn–Tucker condition and the feasibility condition.

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,

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

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. Vegas measures congestion at a link by its queueing delay, and that of a path by the end–to–end queueing delay (without propagation delay). A Vegas source computes the queueing delay from the round trip time and the estimated propagation delay, and attempts to set its rate to be proportional to the ratio of propagation to queueing delay, the proportionality constant being between α_s and β_s. We now elaborate on this third interpretation.

The dynamics of the buffer process at link l implies the important 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. Moreover, the difference between the round trip time and the propagation delay is the path price $p^s(t)$, the congestion signal a source needs to adjust its rate. Let $q_l(t) := b_l(t)/c_l$ denote the queueing delay at link l and $q^s(t) = \sum_{l \in L(s)} q_l(t)$ be the end–to–end queueing delay in source s's path. Then (12) implies that, since $q^s(t) = p^s(t)$, a Vegas source sets its target rate to be proportional to the ratio of propagation to queueing delay:

$$x_s(t) \quad = \quad \alpha_s \, \frac{d_s}{q^s(t)} \tag{16}$$

As the number of sources increases, individual source rates necessarily decrease. The relation (16) then implies that queueing delay $q^s(t)$ 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.

It also follows from (16) 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:

$$x_s^* q^{*s} = \alpha_s d_s \tag{17}$$

This is just Little's Law in queueing theory when propagation delay is ignored.

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 [6], defines α_s and β_s in terms of bytes (or packets) per *second*. These two implementations have an obvious impact on fairness: the first penalizes sources with a large propagation delay, while the second favors such sources.

In terms of our model, Theorem 1 implies that the equilibrium rates x^* are *weighted proportionally fair* [11, 12]: for any other feasible rate vector x, we have

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

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, for some α.

These two implementations lead to different fairness in equilibrium. When $\alpha_s d_s = \alpha$ (in unit of (say) 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. All sources with the same path price receive the same rate, for example, in a single–link network. In a network with multiple congested links, however, a source that traverses more links, *not merely* having higher propagation delay, will be discriminated against. This is because for each marginal increment in aggregate utility—the objective of the primal problem (4–5)—such a long connection consumes more resources than a short one that uses fewer links; see [16, Section V]. 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 being proportional to sources' propagation delays. (17) implies that if two sources r and s face the same path price (or equivalently, the same end–to–end queueing delay), then their equilibrium rates are proportional to their propagation delays:

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

In particular, if there is only a single congested link in the network, then a source that has twice the propagation delay will receive twice the bandwidth. 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).

It is argued in [13, Remark 2] that TCP Reno can be roughly modeled as maximizing problem (4–5) with utility functions (ignoring random loss) $U_s(x_s) = -1/d_s^2 x_s$ Hence in equilibrium source rates satisfy $d_s^2 x_s^{*2} = 1/p^{*s}$. If two sources r and s see the same path price (e.g., in a single–link network), then their rates are *inversely* proportional to their propagation delays:

$$d_r x_r^* = d_2 x_s^*$$

That is, a source with twice the propagation delay receives half as much bandwidth. This discrimination against connections with high propagation delay is well known in the literature, e.g., [7, 9, 14, 17, 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 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 congestion and recovering from it. This is confirmed by the experimental results of [6] and [1].

As observed in [6] 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. The minimum buffer needed in the *entire* network for equilibrium to exists is $\sum_{s \in S} \alpha_s d_s$.

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 Early Marking (REM), in the form of the recently proposed ECN bit [8, 22].

4.1 Coupling Backlog and Price

Vegas relies on the buffer process to compute its congestion measure $p_l(t)$. Indeed, the link price is proportional to the backlog, $p_l(t) = b_l(t)/c_l$. This is similar to the scheme in [15], where $p_l(t) = b_l(t)/\gamma$ for a small constant $\gamma > 0$ that is common for all links, and hence suffers from the same drawback [3]. Notice that 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 oscillation. For example, in a single–link network, if every source keeps $\alpha_s d_s = \alpha$ packets buffered at the link, 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 [18]. 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 under both the PF and the WPF scheme, 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 $-1 < \epsilon_s \leq D_s(t)/d_s(t) - 1$ for all t so that the estimate satisfies $0 < \hat{d}_s(t) \leq D_s(t)$. The next

[4] A remedy is suggested for the first problem in [18] 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 Early Marking (REM) eliminates persistent congestion, and thus facilitates the proposed modification.

result says that the estimation error effectively changes the utility function: source s appears to have a utility (cf. (3))

$$U_s(x_s) \;=\; (1+\epsilon_s)\alpha_s d_s \log x_s + \epsilon_s d_s x_s \tag{18}$$

and the objective of the Vegas sources appears to

$$\max_{x\geq 0} \quad \sum_s U_s(x_s) \;=\; \sum_s (1+\epsilon_s)\alpha_s d_s \log x_s + \epsilon_s d_s x_s \tag{19}$$

$$\text{subject to} \quad \sum_{s\in S(l)} x_s \;\leq\; c_l, \quad l \in L \tag{20}$$

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 (19–20).*

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

$$U_s'(x_s^*) \;=\; \frac{(1+\epsilon_s)\alpha_s d_s}{x_s^*} + \epsilon_s d_s \;=\; \sum_{l\in L(s)} p_l^* \tag{21}$$

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

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

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

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

This yields (21) 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 (19–20). Second, it allows us to compute the new equilibrium rates, and hence assess the fairness, when we know the relative error in propagation delay estimation. It provides a qualitative assessment of the effect of estimation error when such knowledge is not available.

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

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

With error, their rates are related by

$$\frac{(1+\epsilon_r)\alpha_r d_r}{x_r^*} + \epsilon_r d_r \;=\; \frac{(1+\epsilon_s)\alpha_s d_s}{x_s^*} + \epsilon_s d_s \tag{22}$$

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 source, 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 [21]. 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. Indeed, if two sources r and s see the same price, then they have the same queueing delay (because $p_l(t) = b_l(t)/c_l$). If the error in round trip time estimation is entirely due to the (average) queueing delay q, then $q = \epsilon_r d_r = \epsilon_s d_s$ for both sources. For PF implementation, (22) then implies that their rates are proportional to $1 + q/d_s$, i.e., instead of equally sharing the bandwidth, the source with a smaller propagation delay d_s will be favored. In a high speed network where q/d_s is small, this distortion is small.

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). Both are consequences of Vegas' reliance on the buffer process to compute link prices. If buffer capacity is not sufficient in the network, equilibrium cannot be reached, loss cannot be avoided, and Vegas reverts to Reno. This section demonstrates how binary feedback can be used to correct this situation.

Explicit feedback decouples price computation and the buffer process, so that buffer occupancy b_l^* can stay low while the price converges to its equilibrium value p_l^* (which can be much higher than b_l^*/c_l). Minimum round trip time would then 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 marking. This can be done using the Random Early Marking (REM) algorithm of [3].

REM is a congestion control mechanism derived from a global optimization framework. It consists of a link algorithm and a source algorithm. The link algorithm computes the link price and feeds it back to sources through packet marking. The source algorithm estimates its path price from the observed marks and

adjusts its rate. We now summarize REM in the context of Vegas; see [3] for its derivation and evaluation of its stability, fairness and robustness through extensive simulations.

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:

$$p_l(t+1) \quad = \quad [p_l(t) + \gamma(\mu_l b_l(t) + x^l(t) - c_l)]^+ \tag{23}$$

where $\gamma > 0$ is a small constant and $0 < \mu_l < 1$. The parameters γ controls the rate of convergence and μ_l trades off link utilization and average backlog. Hence $p_l(t)$ is increased when the backlog $b_l(t)$ or the aggregate input rate $x^l(t)$ at link l is large compared with its capacity $c_l(t)$, and is reduced otherwise. Note that the algorithm does not require per–flow information. 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:

$$m_l(t) \quad = \quad 1 - \phi^{-p_l(t)} \tag{24}$$

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

$$m^s(t) \quad = \quad 1 - \prod_{l \in L(s)} (1 - m_l(t)) \;=\; 1 - \phi^{-p^s(t)} \tag{25}$$

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 (25):

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

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

$$x_s(t) \quad = \quad \frac{\alpha_s d_s}{\hat{p}^s(t)} \quad = \quad \frac{\alpha_s d_s}{-\log_\phi(1 - \hat{m}^s(t))} \tag{26}$$

Hence the source algorithm (26) says: if the path is congested (the fraction of marked packets is large), transmit at a small rate, and vice versa.

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

Vegas with REM:

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

How to set parameters ϕ, γ, μ_l is discussed in [3] which also shows that REM is very robust to parameter setting.

As argued in [3], the price adjustment (23) leads to small backlog ($b_l^* \simeq 0$) and high utilization ($x^{l*} \simeq c_l$) in equilibrium at bottleneck links l, regardless of the equilibrium price p_l^*. Hence high utilization is not achieved but 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.

6 Evaluation

This section presents three 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.

We use the *ns-2* network simulator [20] 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.

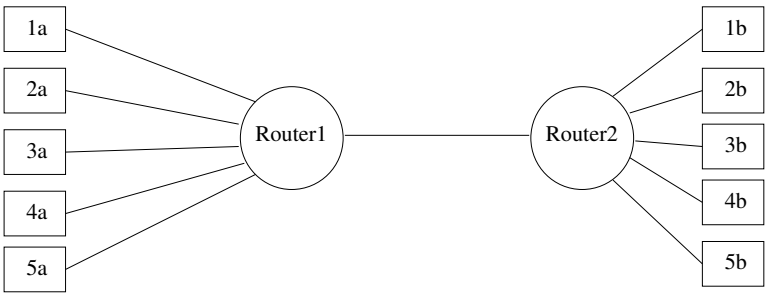


Figure 1: Network topology

6.1 Equilibrium and Fairness

We first run five connections across the network (i.e., between Host1a and Host1b, 2a and 2b etc.) in an effort to understand how these connections compete for bandwidth on the shared link. 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.

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. 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

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

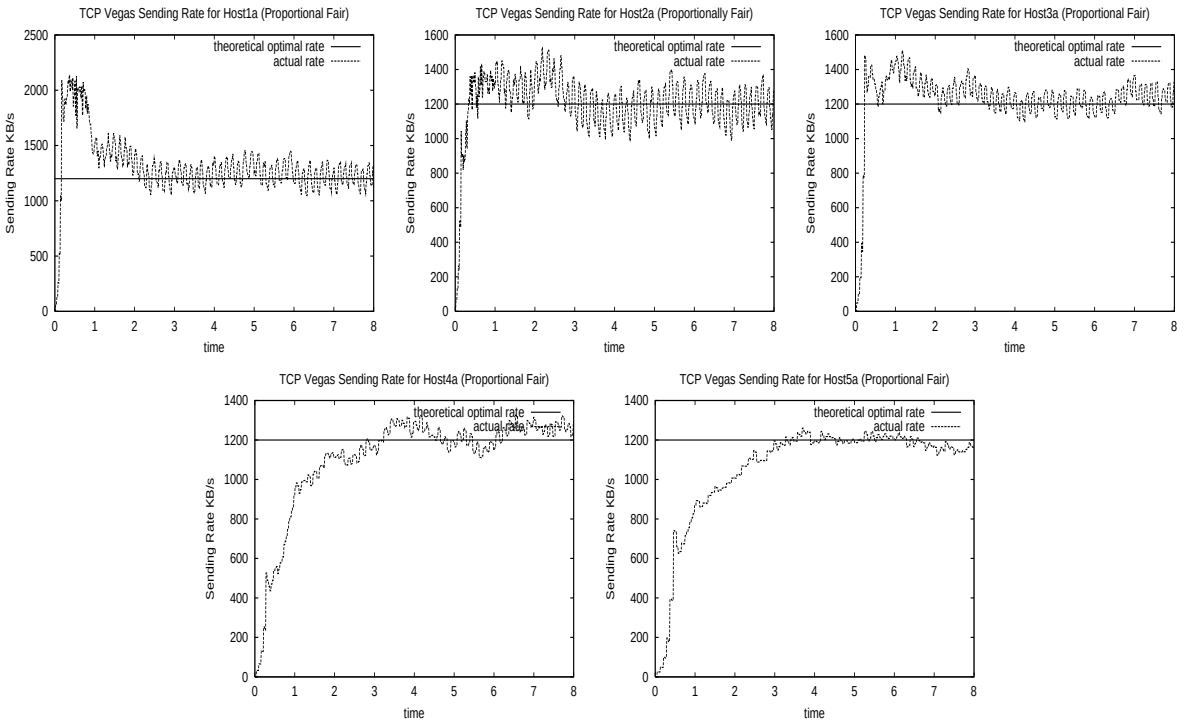


Figure 2: Stability (PF): sending rate of five connections

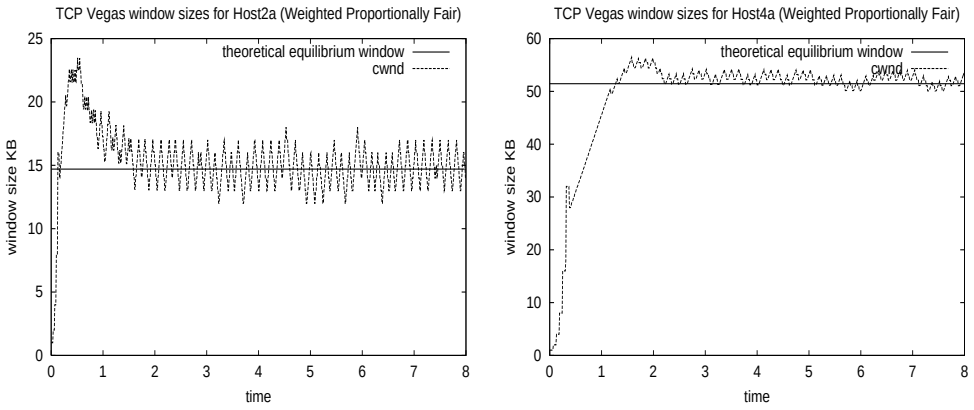
how well the model predicts the simulation.

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 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.

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



(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 stands for Model and S stands for Simulation, all simulation numbers are averaged at equilibrium point.

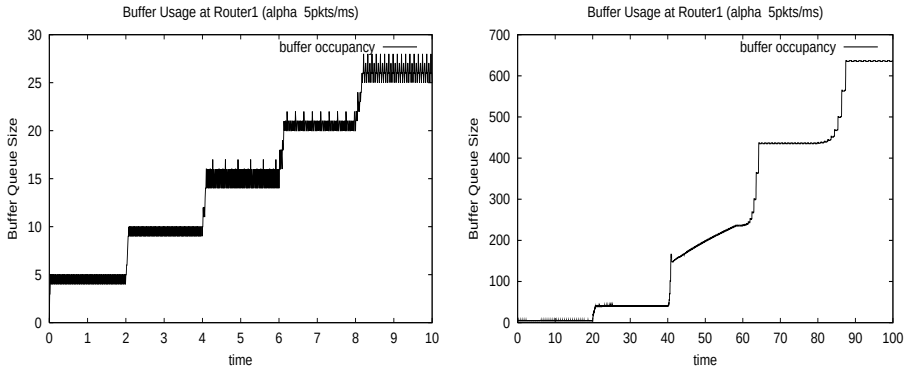
simplicity. Vegas adjusts the congestion window by one packet in each round trip time. The adjustment is large relative to $\alpha_s = \beta_s = 2$ packets, rendering the window prone to oscillation. We have repeated the simulation using an α_s that is 10 times as large (corresponding to a step-size γ 10 times as small). This reduces the impact of adjusting the window by one packet, and the curves smooth out.

6.2 Persistent Congestion

We next validate that Vegas leads to persistent congestion under pathological conditions. We set the round trip latency to 1ms for *all* connections, the host–router links are all 1600 Mbps, and the bottleneck link has a bandwidth of 8 Mbps. We set α_s to 5 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 1 ms for each source, thus eliminating the error in propagation delay estimation. We then run five connections, each starting 2 seconds after the previous connection. That is, Host 1a starts sending at time 0, 2a starts at 2s, and so on. As shown in Figure 4(a), the

buffer occupancy increases linearly in the number of sources.



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

Figure 4: 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. Since each source perceives a larger round trip delay due to queueing at the router, it takes longer for them to reach equilibrium. Therefore, sessions are now staggered 20 seconds apart. As shown in Figure 4(b), buffer occupancy grows much faster than linearly in the number of sources. We have also applied Theorem 3 to calculate the queue size, RTT with queueing and equilibrium rates. The measured numbers match very well with the prediction of Theorem 3 for the first half of the simulation up to a queue size of 272, further verifying our Vegas model. At very large buffer sizes, the Karush–Kuhn–Tucker equation describing the equilibrium situation becomes very ill–conditioned, and the system can be easily jolted into a different point, as it has apparently been.

The 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, at $t = 40$ when two sources are active, the ratio of the measured (equilibrium) source rates is $x_1 : x_2 = 1 : 7.7$; at $t = 60$ when three sources are active, the ratio is $x_1 : x_2 : x_3 = 1 : 5.7 : 36.3$ (the ratio calculated using Theorem 3 is $x_1 : x_2 = 1 : 7.7$ at $t = 40$ and $x_1 : x_2 : x_3 = 1 : 6.2 : 48.3$ at $t = 60$).

6.3 Vegas + REM

Finally, we implement REM at Router1, which updates link price every 1ms according to (23). 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.[6]

We set the bottleneck link bandwidth to be 48Mbps and run three sources (Host1-3a) with a round trip latency of 10ms, 20ms and 30ms, respectively. Host-router links are 100Mbps and α_s is 2 pkts–per–ms for WPF. Parameters related to REM are set as follows: ϕ = 1.15, μ_l = 0.5, γ = 0.005.

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

We start 3 connections with an inter–start interval of 5ms 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 three connections and buffer occupancy at Router1. As expected, each of the three connections converges to its appropriate share of link bandwidth over time. When the link is not congested, source rate oscillates more severely, as seen from Host1a during time 0 - 5s. This is a consequence of the log utility function; see [3]. As more sources become active (5 - 16s), oscillation becomes smaller and convergence faster. Without REM, each connection would have maintained $\alpha_s d_s$ packets buffered in the router, amounting to 120 packets in equilibrium. With REM, buffer occupancy is much smaller in equilibrium, even though the link utilitzation is high (varies from 92% to 96%). Setting μ_l large keeps buffer occupancy small while decreases link utilization. This tradeoff could be decided by each router separately based on its own resources, such buffer space, and other policies. Small buffer occupancy reduces the estimation error and eliminates the superlinear growth in queue length demonstrated in Figure 4(b) of Section 6.2. This is confirmed by the measurement shown in Table 3.

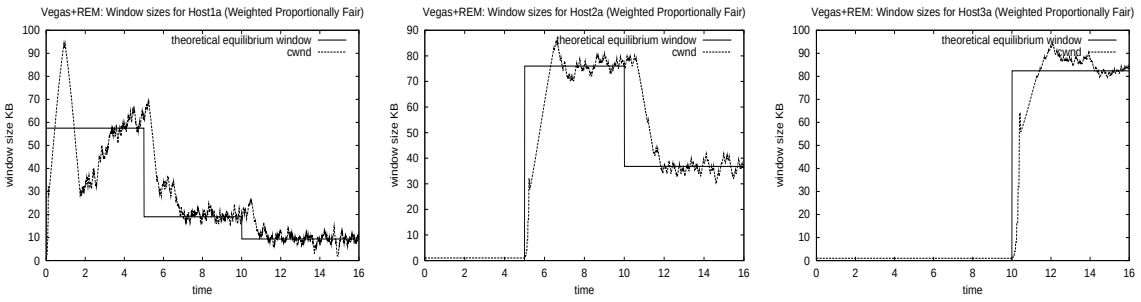
Host	*1*a		*2*a		*3*a	
	Model	Simulation	Model	Simulation	Model	Simulation
baseRTT (ms)	10.34	10.34	20.34	20.34	30.34	30.34

Table 3: Comparison of baseRTT in Vegas+REM

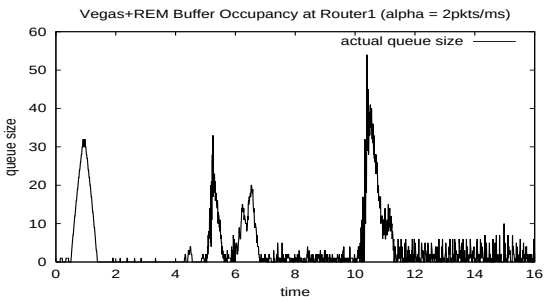
7 Conclusions

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 through binary feedback that decouples the computation and the feedback of congestion measure. We have presented simulation results that validate our conclusions. Extensive simulations to compare Vegas+REM and Reno+RED will be reported in a future paper.

Acknowledgement. We gratefully acknowledge helpful discussions with Sanjeewa Athuraliya. This work supported by the Australian Research Council through grants S499705, A49930405 and S4005343, and by the National Science Foundation through grant ANI-9906704.



(a) Window Size



(b) Buffer Occupancy

Figure 5: Stability of Vegas+REM. Link utilization: 96%(0-5s), 95%(5-10s), 92%(10-16s)

References

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

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

[3] Omitted for anonymity. January 2000.

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

[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] 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.

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

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

[9] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Trans. on Networking*, 1(4):397–413, August 1993.

[10] 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.

[11] F. P. Kelly. Charging and rate control for elastic traffic. *European Transactions on Telecommunications*, 8:33–37, 1997. http://www.statslab.cam.ac.uk/$\tilde{}$frank/elastic.html.

[12] 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.

[13] 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.

[14] 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.

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

[16] Steven H. Low and David E. Lapsley. Optimization flow control, I: basic algorithm and convergence. *IEEE/ACM Transactions on Networking*, 7(6), December 1999.

[17] 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.

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

[19] Jeonghoon Mo and Jean Walrand. Fair end–to–end window–based congestion control. Preprint, 1999.

[20] *Ns network simulator*. Available via http://www-nrg.ee.lbl.gov/ns/.

[21] V. Paxson. End–to–End Routing Behavior in the Internet. *Proceedings of SIGCOMM'96, ACM*, August 1996.

[22] K. K. Ramakrishnan and S. Floyd. A Proposal to add Explicit Congestion Notification (ECN) to IP. Internet draft draft-kksjf-ecn-01.txt, July 1998.

[23] K. K. Ramakrishnan and Ran Jain. A binary feedback scheme for congestion avoidance in computer networks. *ACM Transactions on Computer Systems*, 8(2):158–181, May 1990.