Analysis of TCP Vegas and TCP Reno

Omar Ait-Hellal and Eitan Altman
INRIA B.P. 93, 2004 Route des Lucioles
06902 Sophia Antipolis, FRANCE.
E-mail : {oaithel,altman}@sophia.inria.fr

1 Introduction

The most known congestion control mechanism for the TCP/IP protocols, is that proposed by Van Jacobson [10] dubbed Tahoe. Revised two years after by his author [11], by including *Fast Recovery* and *Fast Retransmit* [13], the new version called Reno achieves more bandwidth utilization and retransmits less data than its predecessor. However its manner to predict the available bandwidth by provoking losses, is the main source of large averages and variances of *rtt* (Round Trip Time) and also of aggressiveness (network saturation).
Brakmo and Peterson [5] proposed a new congestion avoidance algorithm dubbed Vegas based on some modifications on the source behavior of Reno. This new algorithm is able to interoperate with any other valid implementation of TCP and was reported in [5] to achieve between 37% and 71% better throughput than Reno, to retransmit between one fifth and one half as much data as does Reno and to use more efficiently the available bandwidth. In [6] The authors confirm these improvements, and report a gain between 4%-20% over Reno in throughput.
The existing studies of Vegas' performance have employed simulation results or live experiments. In this paper we use an analytic fluid approach in order to analyze the different features of both Vegas and Reno. We then use simulations to confirm our analytic results. When the available bandwidth is high, our results confirm the claims of [5, 6]; indeed Vegas can retransmit less than one-fifth as much data as Reno does, so that the higher the available bandwidth is, the more efficient Vegas is. However, under heavy congestion Vegas behaves like Reno and does not manage to make efficient use of its new mechanism for detection of congestion. The analytic results that we obtain are the evolution of the window size, round trip times and their averages, and the average throughput.

2 Description of Reno and Vegas

Reno and Vegas versions of TCP flow control are two dynamic window schemes that use two types of mechanisms to detect packet losses: duplicate Ack and timeouts (see details below). The congestion window (Cwnd) is defined as the number of allowed packets (or bytes) sent but not yet acknowledged (bytes in transit).
In our analysis we shall consider some discrete time epochs $\mathcal{T}_0, \mathcal{T}_1, \mathcal{T}_2 ...$, where an interval $(\mathcal{T}_n, \mathcal{T}_{n+1})$ is called a mini-slot (see e.g. [8]). $\mathcal{T}_n$ are defined recursively as follows. $\mathcal{T}_0 := 0$; at time $\mathcal{T}_0$ TCP sends the first packet. This is called the first generation of packets. $\mathcal{T}_1$ is defined to be the time instant at which its acknowledgment returns to the source. In all versions of TCP, each arriving Ack causes the transmission of zero, one or two packets (depending on the phase). We define the $n+1$st generation of packets to be those that were sent by the source due to the Acks of the packets of the nth generation. $\mathcal{T}_{n+1}$ is defined to be the time instant at which the acknowledgment for the last packet of the nth generation returns to the source. The packets belonging to the nth generation are often called in the literature the nth window. We shall denote the number of packets in the nth window by W_n and the duration of the nth mini-slot by $\delta_n = \mathcal{T}_{n+1} - \mathcal{T}_n$.
The control mechanism (Reno or Vegas) increases or decreases the window size depending on whether the packet is acknowledged or lost (its timeout has expired or the source received duplicate Acks) and it is set to min{receive window, congestion window}. The size of the receive window is defined by the destination and determines the number of packets it can receive(packets out of sequence are buffered at the destination), we assume that the receive window is always larger than the congestion window.
TCP Reno: Reno adjusts its window according to the phase it performs:
Slow start: It is an "exponential" increase of the window; the source sends two packets every time an Ack is received. A Slow start phase is initiated after a loss is detected through a time-out mechanism and it ends when the window reaches a certain level called *slow start threshold* W_{th} (which is set in our experiments at the beginning to 32 packets).
Congestion avoidance: This phase begins at the end of the slow start phase or after a loss is detected by duplicate Acks. TCP continues to increase its congestion window by $1/Cwnd$ every time it receives an Ack, till it reaches the receive window or till a loss occurs (it uses the loss of packets to detect an eventually congested network), whichever occurs first.
When the sender sends a packet, it starts a retransmission timer. If the timer expires (i.e if it exceeds some value, called the coarse grain timeout) and the packet is not yet acknowledged, the packet is considered to be lost and the slow start phase is performed again by setting the slow start threshold to one-half the current window size and the congestion window to one. Reno checks for timeout every 500 ms and determines the coarse grain timeout by keeping an estimator of the round trip time.

We next describe the fast retransmit and fast recovery mechanisms.
A loss can also be detected by receiving three duplicate Acks; in this case we retransmit *only* the packet that is supposed to be lost but not the subsequent ones, without waiting for a retransmission timer to expire (this is the *Fast Retransmit* mechanism), and the congestion avoidance phase is again performed:
1) The Slow Start threshold (W_{th}) is set to one-half the current window size.
2) The congestion window ($Cwnd$) is set to W_{th} plus three times the packet size.
3) Each time the sender receives a duplicate Ack, it increments $Cwnd$ by one packet and sends a new packet.
4) When the first non-duplicated Ack arrives, $Cwnd$ is set to W_{th}.
1-4 is the *fast recovery* mechanism.
TCP Vegas: Vegas also uses the fast retransmit and fast recovery mechanisms. It differs from Reno in its manner to detect losses, to detect the available bandwidth and it also has a different slow start behavior. The congestion window is updated according to the phase being performed:
Congestion avoidance: In its congestion avoidance phase, Vegas estimates the amount of data buffered at switches and according to that, it decides whether it increases or decreases its congestion window by one packet. This estimation is done one time every mini-slot δ_n: Vegas sets $BaseRTT$ to the minimum of all measured round trip times (including those of previous mini-slots) and computes the $Expected$ rate as $Expected = Cwnd/BaseRTT$ where the $Cwnd$ is the number of bytes in transit. During this period, Vegas records the sending time of a distinguished packet and how many bytes are transmitted between this time and time when its acknowledgment is received and it computes the sending rate as $Actual = Cwnd/RTT_d$, where RTT_d is the round trip time of the distinguished packet. The sender defines three thresholds α_{th}, β_{th} and γ_{th} (in our experiments, we set $\alpha_{th} = 2$, $\beta_{th} = 4$ and $\gamma_{th} = 1$ ($Vegas_{2,4}$)). If $Expected - Actual \leq \alpha_{th}/BaseRTT$ then we increase the congestion window by $1/Cwnd$ every time we receive an Ack (as does Reno). However, if $Expected - Actual > \beta_{th}/BaseRTT$ we decrease the congestion window by one packet size, finally if the difference $Expected - Actual$ falls between the thresholds (α_{th} and β_{th}), we keep the congestion window unchanged.
In our analysis we use a slightly different version corresponding to the implemented version (Vegas release 0.8 [4]), where RTT_d is considered as the average round trip time over all packets acknowledged between sending this packet till receiving its Ack.
Slow start: The principle is similar to that of Reno, However, Vegas doubles its congestion window every other δ_i, as opposed to Reno's every δ_i. The slow start phase ends, once Vegas detects queue buildup. This is done, when the congestion window reaches a value W_{th} for which the difference $Expected - Actual$ exceeds a certain threshold $\gamma_{th}/BaseRTT$ and then Vegas transitions to its congestion avoidance phase. This helps to avoid losses in the slow start phase, where they always may be in the order of half the current window size in case of Reno.
Losses detection: Vegas extends Reno's retransmission mechanism by introducing a new fine grain timeout, which allows to detect a loss faster. In many cases this helps to recover from (multiple) losses without waiting the retransmission timer to expire (see [2, 5, 6] for more description).

3 The Model and the main results

We consider a situation of two sources sending to the same sink and sharing the bottleneck. One source is controlled and the other is modeled by a constant rate source. We assume that a fixed packet size (default of 576 bytes in our simulations) is used. We shall use below a model that describes the system's evolution as a function of the generation n. Within each slot δ_n, a fluid approach will be used.
More precisely, we use the fact, established in [3, 8] for the slow-start phase, that packets are sent in *bursts*, or in *trains*, where each train represents a generation; between trains there are periods of silence, so that the time interval between the transmission of the first packet of two consecutive generations is the (fixed) round trip time T (to be defined below). A similar situation will be shown to hold also during the congestion avoidance phase. In both phases, this situation occurs as long as the period of transmission of a generation is smaller than T, and it is characterized by the fact that the first packet of a generation finds the queue empty. Once it reaches T we shall have a continuous transmission (no silent periods), and the queue does not empty anymore. We shall approximate in our analysis below the transmission of packets during a train by a continuous flow of "fluid", whose rate is $Thp_{in}(W_n)$. $Thp_{in}(W_n)$ will be zero during silent periods. This approximation will imply, in particular, that the number of packets in the queue will be real-valued (instead of integer valued) and can be viewed as a fluid.
We shall use the same notations as [3]. Define

$W_{max} \stackrel{\Delta}{=}$ maximal size attained by Reno's window before a loss occurs.

$W_{vmax} \stackrel{\Delta}{=}$ window size at which Vegas stabilizes.

$W^v_{max} \stackrel{\Delta}{=}$ maximal size attained by Vegas' window (maximum of W_{vmax}).

$B \stackrel{\Delta}{=}$ buffer size.

$\mu \stackrel{\Delta}{=}$ service rate of the router.

$\lambda \stackrel{\Delta}{=}$ rate at which exogenous packets are transmitted. We assume that $\lambda < \mu$.

$C \stackrel{\Delta}{=}$ the maximum bandwidth at which the controlled source can transmit (e.g., due to the maximum bandwidth of the link connecting the source to the first node on its path). We shall assume that C is large (so that it does not slow down the transmission of packets).

$\tau \stackrel{\Delta}{=}$ round trip delay when the queue is empty (i.e, the time between the transmission of a packet until its Ack is received), not including the service time.

$T \stackrel{\Delta}{=} \tau + \mu^{-1}$ "sojourn time" of a packet in an empty system, i.e., the round trip delay plus the service time μ^{-1} (the smallest round trip delay).

$\overline{RTT}_n \triangleq$ the average round trip delay taken over the nth window.
$\overline{Thp} \triangleq$ the average throughput.
$\delta_n \triangleq$ time that takes the nth window (W_n) to be acknowledged. It is the time between receiving the Ack of the last packet sent in the previous window (W_{n-1}) and the arrival time of the Ack of the last packet sent in the actual one (W_n).
$\epsilon_{W_n} \triangleq \overline{RTT}_n - T =$ the average excess time over a window W.
All rates and bandwidths are expressed in packets per second.
Our fluid approximation implies the following property.

- **(Q0)** The queue does not empty during the time period that a burst of controlled packets arrives to it.

This is shown inductively. When the burst is of size 2 for the first time, the two transmitted packets are spaced by a time period of $1/C$. Thus the transmission rate of controlled packet into the system is $C \geq \mu - \lambda$. The rate at which packets are served is lower bounded by $\mu - \lambda$ ($\mu - \lambda$ would be obtained if the exogenous traffic had full priority over the controlled traffic). Therefore, acknowledgments return to the source at a rate which is not smaller than $\mu - \lambda$. Thus, in the next generation, packets are now transmitted from the source at a rate that is not smaller than $\mu - \lambda$. Assume now that the W_n packets of the nth generation are transmitted at a rate not less than $\mu - \lambda$. Then, using the same argument, their acknowledgments return at a rate not less than $\mu - \lambda$. Hence, the W_{n+1} packets of the following generations will be transmitted at a rate not less than $\mu - \lambda$ (even in the case that $W_{n+1} < W_n$, as may happen in Vegas).
First, we compute δ_n the time necessary to receive all acknowledgments of the nth window W_n, and $\overline{RTT}_n$ the average round trip time over this window, in a congestion avoidance and slow-start phase. The proof of the following Theorem is given in [1].

Theorem 3.1. *(i) Consider the congestion phase. In Vegas when the window remains constant (stabilizes),*

$$\overline{RTT}_n = T + \epsilon_{W_n} = \frac{W_n}{\mu - \lambda} = \delta_n. \tag{1}$$

For both Vegas and Reno, when the window increases, we have

$$\delta_n \cong \begin{cases} T + \dfrac{1}{\mu - \lambda} & \text{if} \quad 1 \leq W_n - 1 \leq T(\mu - \lambda) \\ \dfrac{W_n}{\mu - \lambda} & \text{otherwise,} \end{cases} \tag{2}$$

$$\overline{RTT}_n \cong \begin{cases} T & \text{if} \quad W_n - 1 \leq T(\mu - \lambda) \\ \dfrac{W_n(W_n - 1)}{W_n(\mu - \lambda) + \lambda} & \text{otherwise} \end{cases}$$

W^v_{max} and W_{max} are given by

$$W^v_{max} = \left(\frac{B}{\mu} + T\right)(\mu - \lambda) \tag{3}$$

$$W_{max} = \begin{cases} \dfrac{\mu + B\lambda}{\mu - B(\mu - \lambda)} & \text{if } \lambda > \dfrac{(B-1)\mu}{B} \\ & \text{and } W_{max} - 1 \leq T(\mu - \lambda) \\ \left(\frac{B}{\mu} + T\right)(\mu - \lambda) & \text{otherwise} \end{cases} \tag{4}$$

(ii) In the slow-start, we have for Reno

$$\delta_n = \begin{cases} T + \dfrac{W_n}{2\mu - \lambda} & \text{if } \ T(\mu - \lambda/2) \geq W_n/2 \geq 1 \\ \dfrac{W_n}{\mu - \lambda/2} & \text{otherwise} \end{cases}$$

$$\overline{RTT}_n = \begin{cases} T + \dfrac{W_n}{4(\mu - \frac{\lambda}{2})} - \dfrac{1}{2C} & \text{if } \ T(\mu - \lambda/2) \geq W_n/2 \\ \dfrac{3W_n}{4\mu - 2\lambda} - \dfrac{1}{2C} & \text{otherwise} \end{cases}$$

(iii) For Vegas, in the slow-start, since it doubles its congestion window every other δ, we distinguish two cases:
The case when Vegas doubles its window:

$$\delta_n^{(1)} = \begin{cases} T + \dfrac{W_n\mu + \lambda}{2\mu^2 - \lambda^2} & \text{if} \quad T\dfrac{\mu^2 - \lambda^2/2}{\mu + \lambda} \geq W_n/2 \geq 1 \\ \dfrac{W_n(2\mu + \lambda)}{2\mu^2 - \lambda^2} & \text{otherwise} \end{cases}$$

$$\overline{RTT}_n^{(1)} = \begin{cases} T + \dfrac{W_n\mu + \lambda}{4\mu^2 - 2\lambda^2} - \dfrac{1}{2C} & \text{if } T\dfrac{\mu^2 - \lambda^2/2}{\mu + \lambda} \geq W_n/2 \\ \dfrac{W_n(3\mu + 2\lambda) - \lambda}{4\mu^2 - 2\lambda^2} - \dfrac{1}{2C} & \text{otherwise} \end{cases}$$

and when the window remains constant:

$$\delta_n^{(2)} = \begin{cases} T + \dfrac{(W_n - 1)\lambda}{2\mu^2 - \lambda^2} & \text{if} \quad T\dfrac{2\mu^2 - \lambda^2}{2\mu + \lambda} \geq W_n \\ \dfrac{W_n(\mu + \lambda)}{\mu^2 - \lambda^2/2} & \text{otherwise} \end{cases}$$

$$\overline{RTT}_n^{(1)} = \begin{cases} T + \dfrac{(W_n - 1)\lambda}{4\mu^2 - 2\lambda^2} & \text{if } \ T\dfrac{2\mu^2 - \lambda^2}{2\mu + \lambda} \geq W_n \\ \dfrac{W_n(2\mu + \frac{3}{2}\lambda) + \frac{\lambda}{2}}{2\mu^2 - \lambda^2} & \text{otherwise} \end{cases}$$

Next, we compute the maximum window size of Vegas and the window size at the end of the slow-start phase, in order to get the throughput. The proof is given in [1].

Corollary 3.1. *Consider Vegas. (i) The congestion window stabilizes when it reaches W_{vmax}, such that*

$$T(\mu - \lambda) + \alpha_{th} < W_{vmax} \leq T(\mu - \lambda) + \alpha_{th} + 2 \tag{5}$$

(ii) The window threshold is given by

$$W_{th} = 2^{[\log_2 Min_{th}]+1} \tag{6}$$

where $[x]$ is the integer part of x, and

$$Min_{th} = \begin{cases} \frac{(\gamma_{th}\mu-\lambda)C+2\mu^2-\lambda^2}{2C\mu}\times \\ \left(1+\sqrt{1+\frac{4\gamma_{th}C\mu((2CT-1)(2\mu^2-\lambda^2)+C\lambda)}{((\gamma_{th}\mu-\lambda)C+2\mu^2-\lambda^2)^2}}\right) \\ \quad if \quad T\frac{\mu^2-\lambda^2/2}{\mu+\lambda} \geq W_{th}/2 \\ \frac{(3\gamma_{th}\mu+(2\gamma_{th}+1)\lambda)C+(2CT+1)(2\mu^2-\lambda^2)}{2C(3\mu+2\lambda)}\times \\ \left(1+\sqrt{1-z}\right) \qquad otherwise \end{cases}$$

widh $z = \frac{4\gamma_{th}(C\lambda+2\mu^2-\lambda^2)(3\mu+2\lambda)}{((3\gamma_{th}\mu+(2\gamma_{th}+1)\lambda)C+(2CT+1)(2\mu^2-\lambda^2))^2}$

The case of $W_{th}/2 > T\frac{\mu^2-\lambda^2/2}{\mu+\lambda}$, happens when T is very small, and we need large buffers to avoid losses in this situation. In accordance with Corollary 3.1 (i), we shall use in the sequel the approximation $W_{vmax} = T(\mu-\lambda)+\alpha_{th}+1$.

Throughput We compute the throughput in Vegas and Reno by making the following assumptions, respectively:
(i) The available bandwidth is high ($\mu - \lambda > \alpha_{th}\mu/B$ see Theorem 3.2) so Vegas' window stabilizes when reaching W_{vmax}.
(ii) During the congestion avoidance phase, there is at most one loss in a window under Reno congestion control algorithm and when it occurs, it is detected by receiving three duplicate Acks.
We note that assumption (ii) implies that there are only finitely many slow start phases in a session, after which there is a periodic cyclic behavior at the congestion avoidance phase. This is, in fact the desirable operation of Reno. Assumption (i) describes the desirable operation of Vegas. The case when assumption (i) does not hold is delayed to Theorem 3.2. One can analyze also situations where (ii) does not hold, by making other assumptions (see [9]). Theorem 3.1 (i) implies the following:

Corollary 3.2. *Consider Vegas and assumption (i) above. The window size stabilizes at $w = W_{vmax}$, the throughput is $\overline{Thp} = \mu - \lambda$, and the average round trip time is $\overline{rtt} = W_{vmax}/(\mu-\lambda)$.*
Consider Reno under the assumption (ii) above. Then we distinguish two cases:
1)$(W_{max}+1)/2 > T(\mu-\lambda)$: (this corresponds to the case where the queue does not empty) The average throughput, the cycle duration, the total number of packets successfully sent and the average round trip time are, respectively,

$$\overline{Thp} \cong \mu-\lambda, \quad T_3 = \frac{3}{8}\frac{(W_{max}+1)(W_{max}+3)}{(\mu-\lambda)},$$

$$N_3 = \frac{3}{8}(W_{max}+1)(W_{max}+3),$$

$$R_3 = \frac{T_3}{3N_3}(3W_{max}+1) - \frac{(W_{max}+3)^2(W_{max}+1)}{12N_3(\mu-\lambda)}.$$

2) $(W_{max}+1)/2 \leq T(\mu-\lambda)$: The duration of the cycle is given by $T_3 = T_{31}+T_{32}$ where

$$T_{31} = \left(T+\frac{1}{\mu-\lambda}\right)\left(T(\mu-\lambda)-\frac{W_{max}-1}{2}\right),$$

$$T_{32} = \frac{(W_{max}+1)^2 - T^2(\mu-\lambda)^2 + W_{max} - T(\mu-\lambda)+1}{2(\mu-\lambda)}$$

the amount of data sent (safely sent, i.e. sent and acknowledged) in a cycle is given by $N_3 = N_{31}+N_{32}$, where

$$N_{31} = \frac{T^2(\mu-\lambda)^2 - \left(\frac{W_{max}+1}{2}\right)^2 + \frac{W_{max}+1}{2} + T(\mu-\lambda)}{2},$$

$$N_{32} = (\mu-\lambda)T_{32}$$

and the average round trip time (per packet) equals to

$$R_3 = \frac{N_{31}T + N_{32}R_{32}}{N3} \qquad where$$

$$\begin{aligned} R_{32} &= \frac{2T_{32}}{3N_{32}}(W_{max}+T(\mu-\lambda)+1) \\ &\quad -\frac{(W_{max}+1-T(\mu-\lambda))(W_{max}+1)(T(\mu-\lambda)+2)}{3N_{32}(\mu-\lambda)} \end{aligned}$$

The average throughput is given by $\overline{Thp} = N_3/T_3$. ∎

The proof of the corollary and the performance measures that relate to the transient behavior of Vegas and Reno are given in [1].

Definition 3.1. *A congestion control algorithm "has the same behavior as Reno", if it has a cyclic behavior in which its congestion window continues to increase until a loss occurs.*

Theorem 3.2. *For $B \in \left[0, \frac{\alpha_{th}\mu}{\mu-\lambda}\right]$, $Vegas_{\alpha_{th},\beta_{th}}$ "has the same behavior as Reno" and can't estimate the available bandwidth.*

Proof: We have $W_{vmax} > T(\mu-\lambda)+\alpha_{th}$, so if $T(\mu-\lambda)+\alpha_{th} \geq W^v_{max}$ then a loss occurs before Vegas' window reaches the bottom threshold and hence the theorem is obtained from replacing W^v_{max} by its value from (3). Another equivalent proof can be given as follows:
The size of the congestion window depends strongly on the variation of $\overline{RTT}$, the mean round trip time taken over the window, so when the mean can't exceed sufficiently T, cause of a small available bandwidth or small buffer size, the congestion window continues to increase until a loss occurs. In a congestion avoidance phase a loss occurs when the window reaches $W^v_{max}+1$, and Vegas increments its window in the

next δ_i if $T(Expected - Actual) \leq \alpha_{th}$. Since the average round trip time over a window increases as the window size increases, then under Vegas a loss occurs before Vegas' window reaches the bottom threshold (α_{th} or stabilizes) if and only if

$$\frac{W^v_{max}}{T} - \frac{W^v_{max}}{\overline{RTT}} \leq \frac{\alpha_{th}}{T}$$

$$\text{or equivalently } \epsilon_{W^v_{max}} \leq \frac{\alpha_{th} T}{W^v_{max} - \alpha_{th}} \qquad (7)$$

By substituting (3) into (7), and since $\epsilon_{W^v_{max}} = B/\mu$, then (7) is equivalent to

$$\frac{B}{\mu} \leq \frac{\alpha_{th} T}{\left(\frac{B}{\mu} + T\right)(\mu - \lambda) - \alpha_{th}}$$

$$\text{and thus, to } B^2 + \left(T\mu - \frac{\alpha_{th}\mu}{\mu - \lambda}\right) B - \frac{\alpha_{th} T \mu^2}{\mu - \lambda} \leq 0$$

Solving this, we obtain the sufficient and necessary condition for (7): $B \in \left[0, \frac{\alpha_{th}\mu}{\mu - \lambda}\right]$ with $W_{max} > \alpha_{th}$ ∎.

Remark 3.1. *Note that in durations of heavy congestion, we have* $\lim_{\lambda \to \mu}(\alpha_{th}\mu)/(\mu - \lambda) = +\infty$. *Thus, it follows from theorem 3.2 that for any buffer size, Vegas cannot operate well if* λ *is sufficiently close to* μ. *This situation may occur when many connections running Vegas share the link. In this case Vegas allows every connection to have* α_{th} *extra buffers in the network (see (5)) and thus the total extra buffers in the network is, approximatively, some constant* $K \cong \alpha_{th}$ *times the number of connections and then we retrieve the conditions of theorem 3.2. This is the reason why Vegas' performance decreases as the number of active connections increases.*

Figure 1 illustrates an example where Vegas behaves like Reno for a buffer size of 20 pkts. We used REAL simulator [12] after introducing Vegas function from [4] to simulate the model described earlier with a *Star war* video traffic traces as an exogenous traffic. The exogenous source is assumed to send one frame every 30 *ms* and it divides its frames into packets with a fixed size of 576 bytes. During the 100 seconds simulation, the average rate of the exogenous source is 934.136 *pkts/sec*. The capacity of the bottleneck is set to 1500 *pkts/sec*. We can see that the behavior of the window is relatively the same for Vegas and Reno, even if the buffer size is larger than that computed from Theorem 3.2 (~ 6 packets). We can see also that the higher the available bandwidth is, the more efficient Vegas is. Recall that in this example Vegas causes more losses for the exogenous traffic than Reno, because of some problems outlined in [2]. ∎

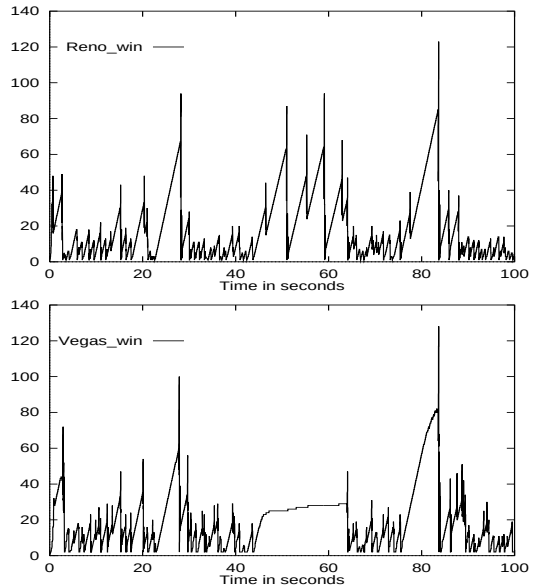

Figure 1: Vegas and Reno's congestion window for $\tau = 81.2$ *ms* and buffer size = 20pkts in presence of video traffic.

References

[1] O. Ait Hellal, E. Altman "Analysis of TCP Vegas and TCP Reno", manuscript.

[2] O. Ait Hellal, E. Altman "Problems in TCP Vegas and TCP Reno", *DNAC Cong. (De Nouvelles Architectures pour les Communications)* , UVSQ, Paris, Dec. 3-5, 1996.

[3] E. Altman, F. Boccara, J. Bolot, P. Nain, P. Brown, D. Collange and C. Fenzy "Analysis of TCP/IP Flow Control Mechanism in Hight- Speed Wide-Area Networks", *IEEE CDC Conference*, Dec 95, New Orleans.

[4] Lawrence Brakmo. "TCP Vegas Release 0.8", Nov 94.

[5] L.S Brakmo, L.L. Peterson, "TCP Vegas : End to End Congestion Avoidance on a Global Internet," *IEEE JSAC,* vol. 13, pp. 1465-1480, 1995.

[6] J.S. Ahn, P. B. Danzig, Z. Liu and L. Yan, "Experience with TCP Vegas: Emulation and experiment," *Proc. SIGCOMM'95 Symp.,* Aug. 1995.

[7] S. Floyd, "TCP and successive fast retransmits". *Lawrence Berkeley Laboratory, Technical Report*, February 24, 1995.

[8] T. V. Lakshman, U. Madhow "Performance analysis of window-based flow control using TCP/IP: the effect of high bandwidth-delay products and random loss", *IFIP Transactions C-26, High Performance Networking*, pp. 135-150, North-Holland, 1994.

[9] T. V. Lakshman, U. Madhow, B. Suter "Window-based error recovery and flow control with a slow acknowledgment channel: a study of TCP/IP performance" submitted.

[10] Van Jacobson. "Congestion avoidance and control". *ACM SIGCOMM 88*, pages 273-288, 1988.

[11] Van Jacobson. "modified TCP congestion avoidance algorithm", *mailling list, end2end-interest*, 30 Apr 1990.

[12] S. Keshav, "REAL: A network simulator", *Comp. Sci. Dept., UC Berkeley, Technical Report* 88/472, 1988.

[13] W. Richard Stevens. "TCP/IP illustrated", *volume 1. Addison Wesley*, 1994.