Analysis of TCP Vegas and TCP Reno

Omar Ait Hellal, Eitan Altman

INRIA B.P. 93, 2004 Route des Lucioles 06902 Sophia Antipolis, FRANCE
E-mail: oaithel, altman@sophia.inria.fr

In this paper we use an analytic fluid approach in order to analyze the different features of both Vegas and Reno TCP versions. We then use simulations to confirm our analytic results. When the available bandwidth is high, 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 congestion detection. The analytic results that we obtain are the evolution of the window size, round trip times and their averages, and the average throughput.

Keywords: TCP Vegas, TCP Reno, fluid approximations.

1. Introduction

The most known congestion control mechanism for the TCP/IP protocols, is that proposed by Van Jacobson [7] dubbed Tahoe. Revised two years after by his author [8], 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 *round trip time* (*rtt*) 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 inter-operate 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 [2] 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 [5,2]; 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 it new mechanism for detection of congestion. The analytic results that we obtain are the evolution of window size, round trip times and their averages, and the average throughput.

The paper is structured as follows: in section 2 a brief description of Reno and Vegas is given. Then we present the model considered in the present work and the main results we obtained, in section 3. In section 4, we conclude the work with some remarks and future work. The detailed calculations and proofs are delayed to the appendix A.

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 (packets or 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. [11]). $\mathcal{T}_n$ 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. Both packets as well as bytes can be used as units for the window size. In the present work, packets are considered as units.

2.1. 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.

2.2. 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, β and γ (in our experiments, we set $\alpha = 2$, $\beta = 4$ and $\gamma = 1$ ($Vegas_{2,4}$)). If $Expected - Actual \leq \alpha/BaseRTT$ then we increase the congestion window by $1/Cwnd$ every time we receive an Ack (as does Reno). However, if $Expected - Actual > \beta/BaseRTT$ we decrease the congestion window by one packet size, finally if the difference $Expected - Actual$ falls between the thresholds (α and β), 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 i.e. one of every two δ_i's the congestion window is doubled, and in the other one it is kept unchanged, 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 $\gamma/BaseRTT$ 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 as follows: Vegas reads and records the system clock each time a packet is sent. When an Ack is received, Vegas reads the clock and does the RTT calculation by computing the mean round trip time, the mean deviation and the fine grain timeout value (*v_ timeout*), which is computed in the same way as does Reno but not adjusted to a multiple of 500 ms. Recall that coarse grain timeout used by Reno, remains effective under Vegas. Vegas decides to retransmit in the following situations:

- When a duplicate Ack is received, Vegas checks to see if the round trip time of the first (in sequence) unacknowledged packet is greater than the fine grain timeout value (v_timeout). If it is, then Vegas retransmits the packet without waiting for three duplicate Acks (required by Reno).

- When a non duplicate Ack is received, if it is the first or the second one after a retransmission, Vegas again checks to see if the time interval since the packet, not yet acknowledged, was sent exceeds the timeout value. If it is, then Vegas retransmits the packet.

This technique enables to detect losses faster and to recover from multiple drops without restarting the slow-start phase if the retransmission timer doesn't expire before. Hence it permits to deal with a problem from which Reno suffers too much, namely, multiple drops in the same window of data [1,12].

Figure 1 illustrates the ideal behavior of the window as well as the variation of the average round trip time in both Reno and Vegas. The slow start phase in the ideal behavior is performed only once during a session. We can see that when the window in Vegas stabilizes, the RTT in this version is smoother than in Reno, while the bandwidth is fully used. The window size in Reno varies between a maximum size and its half, where the maximum size corresponds to the window size for which a loss occurs. The vertical lines in Reno's window correspond to the *fast recovery* mechanism.

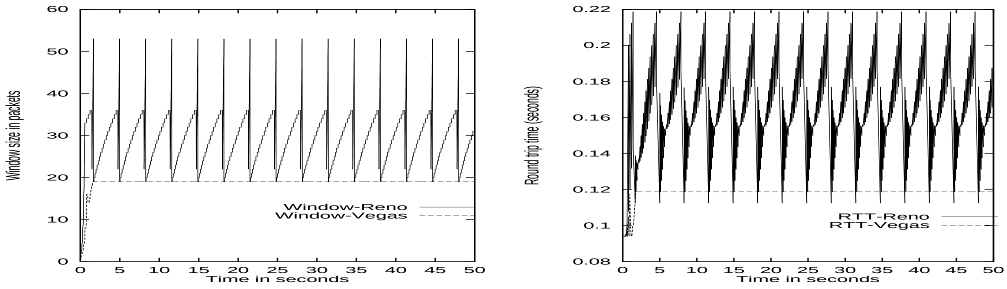

Figure 1. Ideal Vegas' and Reno's congestion window (left) and the average round trip time associated (right).

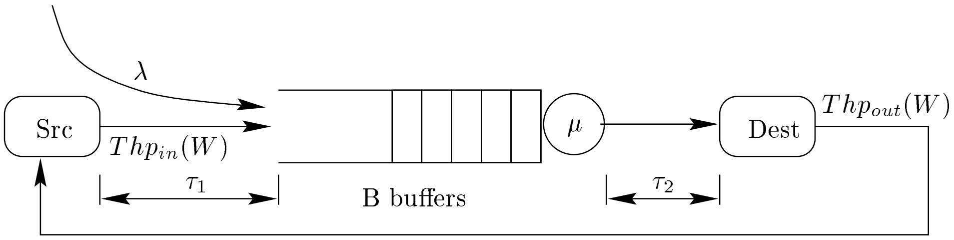

Figure 2. One TCP source with exogenous constant traffic

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, i.e. it represents a TCP connection, and the other one is uncontrolled: it represents a constant rate source (e.g. voice over Internet which does not use TCP). We assume that a fixed packet size (default of 576 bytes in our simulations) is used. The packet size is indeed fixed in most applications of TCP. The fact of analyzing a fixed packet size simplifies the analysis of the throughput as a function of time. 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. The fluid approach consists of approximating the flow of packets by a flow of fluid. The number of packets can then take non-integer values.

More precisely, we use the fact, established in [3,11] 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 holds 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

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

$W_{vmax} \triangleq$ window size at which Vegas stabilizes.

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

$Q(W_n) \triangleq$ total number of packets in the queue (router) when the first packet of the nth window arrives to it.

$B \triangleq$ buffer size.

$\mu \triangleq$ service rate of the router.

$\lambda \triangleq$ rate at which exogenous packets are transmitted. We assume that $\lambda < \mu$.

$Thp_{in}(W_n) \triangleq$ rate at which packets of the nth window W_n are transmitted.

$Thp_{out}(W_n) \triangleq$ rate at which packets of the nth window W_n are served.

$C \triangleq$ 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).

$\tau_1 \triangleq$ time between the transmission of a packet until it reaches the queue.

$\tau_2 \triangleq$ time between the departure of a packet from the queue till it reaches the destination.

$\tau = 2(\tau_1 + \tau_2) \triangleq$ 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.

$T \triangleq \tau + \mu^{-1}$ "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).

3.1. Round trip time and the duration of a mini-slot

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 the Appendix A.

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

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

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

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

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

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

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

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

$$\overline{RTT}_n = \begin{cases} T + \dfrac{W_n}{4(\mu - \frac{\lambda}{2})} - \dfrac{1}{2C} \text{ if} & T(\mu - \lambda/2) \geq W_n/2 \\ \dfrac{3W_n}{4\mu - 2\lambda} - \dfrac{1}{2C} & \text{otherwise} \end{cases} \tag{5}$$

(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:

$$\delta_n = \delta_n^{(1)} = \begin{cases} T + \dfrac{W_n\mu + \lambda}{2\mu^2 - \lambda^2} \text{ if} & T\dfrac{\mu^2 - \lambda^2/2}{\mu + \lambda} \geq W_n/2 \geq 1 \\ \dfrac{W_n(2\mu + \lambda)}{2\mu^2 - \lambda^2} & \text{otherwise} \end{cases} \tag{6}$$

$$\overline{RTT}_n = \overline{RTT}_n^{(1)} = \begin{cases} T + \dfrac{W_n\mu + \lambda}{4\mu^2 - 2\lambda^2} - \dfrac{1}{2C} & \text{if} \quad T\dfrac{\mu^2 - \lambda^2/2}{\mu + \lambda} \geq W_n/2 \\ \dfrac{W_n(3\mu + 2\lambda) - \lambda}{4\mu^2 - 2\lambda^2} - \dfrac{1}{2C} & \text{otherwise} \end{cases} \tag{7}$$

and when the window remains constant, we have

$$\delta_n = \delta_n^{(2)} = \begin{cases} T + \dfrac{(W_n - 1)\lambda}{2\mu^2 - \lambda^2} & \text{if} \quad T\dfrac{2\mu^2 - \lambda^2}{2\mu + \lambda} \geq W_n \\ \dfrac{W_n(\mu + \lambda)}{\mu^2 - \lambda^2/2} & \text{otherwise} \end{cases} \tag{8}$$

$$\overline{RTT}_n = \overline{RTT}_n^{(2)} = \begin{cases} T + \dfrac{(W_n - 1)\lambda}{4\mu^2 - 2\lambda^2} & \text{if} \quad T\dfrac{2\mu^2 - \lambda^2}{2\mu + \lambda} \geq W_n \\ \dfrac{W_n(2\mu + \frac{3}{2}\lambda) + \frac{\lambda}{2}}{2\mu^2 - \lambda^2} & \text{otherwise} \end{cases} \tag{9}$$

From equations (6) and (8) we have

$$\frac{2W_n}{\mu - \lambda/2} \leq \delta_n^{(1)} + \delta_n^{(2)} = \frac{2W_n}{\mu - \lambda/2}\left(\frac{1}{1 - \frac{1}{4}\frac{\mu\lambda + \lambda^2/2}{(\mu - \lambda/2)(\mu + 3\lambda/4)}}\right) \leq \frac{3.5W_n}{\mu - \lambda/2} \tag{10}$$

This relation shows how much excess time does Vegas take to send the same amount of data than Reno, in the slow-start phase (it takes between 0- 75% excess time than Reno). This excess time helps Vegas to keep less data in the network and hence to avoid successive losses which may happen in case of Reno (the average rate of Vegas is less than Reno's one). This happens mainly when the traffic generated by Vegas is continuous (the queue never empties). Indeed, in this case Vegas helps the queue to drain by keeping its window size constant during a round trip time. Thus, packets sent, when the size of the window is doubled, are more likely to find a queue size less important than in case of Reno.

3.2. Slow start threshold in Vegas and the maximum window size

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 of the following corollary is given in the Appendix B.

Corollary 3.2. Consider Vegas. (i) The congestion window stabilizes when it reaches W_{vmax}, such that

$$T(\mu-\lambda)+\alpha < W_{vmax} \leq T(\mu-\lambda)+\alpha+2 \tag{11}$$

(ii) The window threshold is given by

$$W_{th} = 2^{[\log_2 Min_{th}]+1} \tag{12}$$

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

$$Min_{th} = \begin{cases} \dfrac{(\gamma\mu-\lambda)C+2\mu^2-\lambda^2}{2C\mu} \times \sqrt{1+\dfrac{4\gamma C\mu((2CT-1)(2\mu^2-\lambda^2)+C\lambda)}{((\gamma\mu-\lambda)C+2\mu^2-\lambda^2)^2}} \\[2ex] \quad +\dfrac{(\gamma\mu-\lambda)C+2\mu^2-\lambda^2}{2C\mu} \quad \text{if } T\dfrac{\mu^2-\lambda^2/2}{\mu+\lambda} \geq W_{th}/2, \text{ else} \\[2ex] \dfrac{(3\gamma\mu+(2\gamma+1)\lambda)C+(2CT+1)(2\mu^2-\lambda^2)}{2C(3\mu+2\lambda)} \times \\[2ex] \quad \left(1+\sqrt{1-\dfrac{4\gamma(C\lambda+2\mu^2-\lambda^2)(3\mu+2\lambda)}{((3\gamma\mu+(2\gamma+1)\lambda)C+(2CT+1)(2\mu^2-\lambda^2))^2}}\right) \end{cases} \tag{13}$$

3.3. Average round trip time and throughput

We compute the throughput in Vegas and Reno by making the following assumptions respectively:

(i) The available bandwidth is high ($\mu-\lambda > \alpha\mu/B$ see Theorem 3.5) 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 3.5. One can analyze also situations where (ii) does not hold, by making other assumptions (see [12]). Theorem 3.1 (i) implies the following:

Corollary 3.3. 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

$$\overline{rtt} = W_{vmax}/(\mu - \lambda). \tag{14}$$

Consider Reno under the assumption (ii) above. Then we distinguish two cases:
1)$W_{max}/2 > T(\mu - \lambda)$: (this corresponds to the case where the queue does not empty) The average throughput ($\overline{Thp}$), the cycle duration (T_1), the total number of packets successfully sent (N_1) and the average round trip time (R_1) are respectively

$$\overline{Thp} \cong \mu - \lambda, \quad T_1 = \frac{3}{8}\frac{W_{max}(W_{max}+2)}{(\mu - \lambda)}, \quad N_1 = \frac{3}{8}W_{max}(W_{max}+2),$$

$$R_1 = \frac{T_1}{9N_1}(7W_{max} - 8) = \frac{7W_{max} - 8}{9(\mu - \lambda)}.$$

2) $W_{max}/2 \leq T(\mu - \lambda)$: The duration of the cycle is given by $T_1 = T_{21} + T_{22}$ where

$$T_{21} = \left(T + \frac{1}{\mu - \lambda}\right)\left(\frac{2T(\mu - \lambda) + 2 - W_{max}}{2}\right),$$

$$T_{22} = \frac{W_{max}^2 - T^2(\mu - \lambda)^2 + W_{max} - T(\mu - \lambda)}{2(\mu - \lambda)}$$

the amount of data sent (safely sent, i.e. sent and acknowledged) in a cycle is given by $N_1 = N_{21} + N_{22} = \frac{8T_1}{9N_1}W_{max}$, where

$$N_{21} = \frac{(2T(\mu - \lambda) + 2 - W_{max})(2T(\mu - \lambda) + W_{max})}{8}, \quad N_{22} = (\mu - \lambda)T_{22}$$

and the average round trip time (per packet) equals to $R_1 = \dfrac{N_{21}T + N_{22}R_{22}}{N1}$ where

$$R_{22} = \frac{2}{3(\mu - \lambda)}(W_{max} + T(\mu - \lambda)) - \frac{(W_{max} - T(\mu - \lambda))W_{max}(T(\mu - \lambda) + 2)}{3N_{22}(\mu - \lambda)}$$

The average throughput is given by $\overline{Thp} = N_1/T_1$. ∎

The proof of this corollary is given in the Appendix C.

3.4. Simulations and numerical results

Table 1 summarizes different results from several simulations and the results obtained analytically, for the RTT, W_{th} and the maximum window size W_{vmax} under $Vegas_{2,4}$. We have chosen all the parameters such that Vegas' window stabilizes when it reaches W_{vmax}.

parameters			W_{th}			W_{vmax}			$\overline{RTT}$			
μ	λ	τ	Min_{th}	*ana*	*sim*	*ana*	*sim*	*%err*	*ana*	*sim*	*%err*	B_{min}
150	10	0.32	15.07	16	16	49.30	49	0.61	0.35	0.34	0.02	2.14
160	10	0.32	15.51	16	16	52.49	52	0.94	0.34	0.34	0.00	2.13
160	50	0.12	9.909	16	16	17.33	17	1.94	0.15	0.15	0.00	2.90
160	50	0.20	12.30	16	16	26.09	27	3.37	0.24	0.24	0.00	2.90
200	100	0.12	10.50	16	16	15.90	15	6.00	0.15	0.15	0.00	4.00
200	100	0.32	16.17	32	32	35.90	35	2.57	0.35	0.35	0.00	4.00
350	250	0.26	17.67	32	32	29.66	29	2.27	0.29	0.29	0.00	7.00
120	00	0.40	15.08	16	16	52.60	52	1.15	0.43	0.43	0.00	2.00

Table 1
Simulation and analytic results

Table 1 shows that the analytic results are very close to the simulations one and the relative error $|analytic - simulation|/simulation$ doesn't exceed 6% in all cases for the window size and 0.02% for the RTT. In fact, the maximum window size obtained in simulations is an integer (we increase the window by one segment for every received Ack or every RTT), so, in other words we have

$$[T(\mu - \lambda) + \alpha] < W_{vmax} \leq [T(\mu - \lambda) + \beta]$$

where $[x]$ is the integer part of x. Then the relative error can be close to $(100/([T(\mu - \lambda)] + \alpha + 1))\%$. In the last column we compute B_{min} the buffer size for which Vegas' window does not stabilize for the same parameters (Vegas has the same behavior as Reno). The $\overline{RTT}$ we have taken, the simulation value, is the average, since it varies for the same window and the analytic value is computed for W_{vmax} obtained by simulation. Min_{th} is the minimum size of the window in the slow-start phase, equation (13).

In table 2, we compute the average RTT per packet, and Average throughput which is the number of packets sent divided by the time from the beginning. From this results we see how much the analytic values are close to the simulations one for the throughput (0.41-1.28%) and we expect that the relative error for the average round trip time be close to that of the maximum window size for Vegas (see corollary 3.3, equation (14).

parameters			$\overline{Thp}$			$\overline{rtt}$			
μ	λ	τ	*analy*	*sim*	*%err*	*analy*	*sim*	*%err*	*losses*
150	10	0.3241	131.25	131.86	0.46	0.3504	0.3486	0.51	0
160	10	0.3237	139.90	140.51	0.43	0.3482	0.3454	0.81	0
160	50	0.1237	109.20	110.22	0.92	0.1572	0.1544	1.81	0
160	50	0.2037	108.04	108.86	0.75	0.2367	0.2443	3.11	0
200	100	0.1238	99.04	100.24	1.19	0.1593	0.1501	6.12	0
200	100	0.3238	97.29	98.09	0.81	0.3588	0.3493	2.71	0
350	250	0.2638	97.64	98.91	1.28	0.2985	0.2903	2.82	0
120	00	0.4050	109.85	110.31	0.41	0.4355	0.4312	0.99	0

Table 2
Average Throughput and $\overline{rtt}$: Simulation and Analytic results under Vegas

Several results concerning Reno, analytical and simulation can be seen in Table 3. The relative error for the throughput is approximatively the same for the round trip time and doesn't exceed 2.5%. In fact it depends on the number of losses occurred during the simulation, since the fast recovery mechanism as well as the time spent in detecting are not taken into account in the analytical results.

3.5. Vegas' behavior

Definition 3.4. 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.

parameters				$\overline{Thp}$			$\overline{rtt}$			
μ	λ	τ	*Buffer*	*analy*	*sim*	*% err*	*analy*	*sim*	*%err*	*losses*
150	10	0.3240	30	131.95	132.78	0.61	0.3960	0.3916	1.12	6
160	10	0.4037	25	134.90	135.54	0.47	0.4609	0.4539	1.54	5
160	50	0.4037	25	100.62	101.41	0.77	0.4625	0.4589	0.78	6
160	50	0.2437	35	108.39	110.12	1.57	0.3637	0.3665	0.76	9
178.3	50	0.4039	35	119.13	120.18	0.87	0.4833	0.4774	1.23	4
200	100	0.4038	35	92.85	94.87	2.12	0.4723	0.4672	1.09	6
120	00	0.4050	40	115.29	116.15	0.74	0.5728	0.5591	2.45	3
120	00	0.4050	20	109.91	110.46	0.49	0.4707	0.4643	1.35	5

Table 3
Average Throughput and $\overline{rtt}$: Simulation and Analytic results under Reno

Theorem 3.5. For $B \in \left[0, \frac{\alpha\mu}{\mu-\lambda}\right]$, $Vegas_{\alpha,\beta}$ "has the same behavior as Reno" and can't estimate the available bandwidth.

Proof: The size of the congestion window depends strongly on the variation of $\overline{RTT}$, the mean round trip time taken over the window. Hence, when the mean can't exceed sufficiently T, because of a small available bandwidth or a 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 $T(Expected - Actual) \leq \alpha$. 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 or stabilizes) if and only if

$$\frac{W^v_{max}}{T} - \frac{W^v_{max}}{\overline{RTT}} \leq \frac{\alpha}{T} \quad \text{or equivalently} \quad \epsilon_{W^v_{max}} \leq \frac{\alpha T}{W^v_{max} - \alpha} \tag{15}$$

By substituting (3) into (15), and since $\epsilon_{W^v_{max}} = B/\mu$, then (15) is equivalent to

$$\frac{B}{\mu} \leq \frac{\alpha T}{\left(\frac{B}{\mu} + T\right)(\mu - \lambda) - \alpha} \Longleftrightarrow B^2 + \left(T\mu - \frac{\alpha\mu}{\mu - \lambda}\right) B - \frac{\alpha T \mu^2}{\mu - \lambda} \leq 0 \tag{16}$$

Solving this, we obtain the sufficient and necessary condition for (15): $B \in \left[0, \frac{\alpha\mu}{\mu-\lambda}\right]$, with $W_{max} > \alpha$. ∎

Remark 3.1. During heavy congestion, we have $\lim_{\lambda\to\mu}(\alpha\mu)/(\mu-\lambda) = +\infty$ Thus, it follows from theorem 3.5 that for any buffer size, Vegas cannot operate well if λ is sufficiently close to μ.

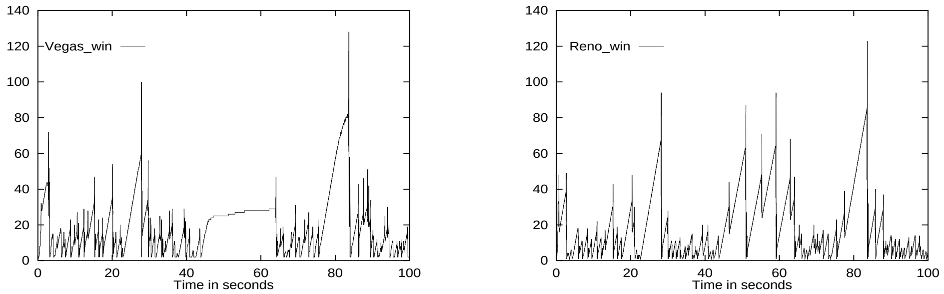

Figure 3. Vegas and Reno's congestion window for $\tau = 81.2$ ms and a buffer size of 20 $pkts$ in presence of video traffic.

Figure 3 illustrates an example where Vegas behaves like Reno for a buffer size of 20 pkts. We used REAL simulator [10] 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 3.5 (~ 6 packets). We can see also that the higher the available bandwidth is, the more efficient Vegas is (Figure 3 (b), between time 40-60 sec).

Generalization: Consider multiple connections running Vegas share a bottleneck node together with a constant exogenous traffic (same model discussed in this paper with N TCP Vegas connections instead of a single connection). Assume that once the queue length exceeds its maximum size, all the connections will experience (at least one) a loss (synchronous losses). The total rate of Vegas connections when all the windows stabilize remains equal to $\mu - \lambda$. Following the same reasoning as above, Vegas *has the same behavior as Reno* if the maximum window size $W^{v,i}_{vmax}$ of connection i satisfies equation (15), i.e. (T_i is the round trip time for connection i)

$$\begin{aligned}
&\frac{W^{v,i}_{max}}{T_i} - \frac{W^{v,i}_{max}}{RTT_i} \leq \frac{\alpha}{T_i} \Longleftrightarrow \frac{W^{v,i}_{max}}{RTT_i}\frac{B}{\mu} \leq \alpha \\
\Longrightarrow\ &\frac{B}{\mu}\sum_{i=1}^{N} \frac{W^{v,i}_{max}}{RTT_i} \leq N\alpha \Longleftrightarrow \frac{B}{\mu}(\mu - \lambda) \leq N\alpha \\
\Longleftrightarrow\ &B \leq \frac{N\alpha\mu}{\mu - \lambda}.
\end{aligned} \tag{17}$$

Thus, the previous theorem can be generalized, in case of multiple connections and synchronous losses as follows:

In case of N Vegas connections with synchronous losses sharing a bottleneck node together with a constant exogenous traffic, Vegas$_{\alpha,\beta}$ has the same behavior as Reno *and cannot estimate the available bandwidth if $B \leq \dfrac{N\alpha\mu}{\mu - \lambda}$.*

This is the reason why Vegas' performances decrease as the number of active connections increases [9].

4. Conclusion

In this paper we have studied the behavior of TCP Vegas and TCP Reno. We have considered a simple model, which allowed us to investigate different features of each algorithm, and to understand the behavior of Vegas in presence of multiple connections, and different network conditions.

TCP Vegas aims to detect the available bandwidth without provoking losses, as Reno does. We have seen that Vegas has good performances when the queue sizes at intermediate routers are large; it enables a full utilization of the link while it keeps the sending rate smoother. However, when the available bandwidth is not sufficiently large, Vegas has the same behavior as Reno and does not manage to make efficient use of its new mechanism for congestion detection.

We computed different performance criterions, such as the average round trip time, the window size in each phase, and the average throughput. Several simulations presented in the paper confirm our hypothesis, and the errors in our analytical results are always negligible. Other performance criterions can be evaluated using this model, such as fairness and stability. This constitutes our future directions.

References

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

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

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

[4] L.S. Brakmo. TCP Vegas Release 0.8, Nov 94. available from ftp://ftp.cs.arizona.edu/xkernel/new-protocols/Vegas.Tar.Z.

[5] L.S. Brakmo, L.L. Peterson, TCP Vegas : End to End Congestion Avoidance on a Global Internet, IEEE Journal on selected Areas in communications, vol. 13(1995), 1465-1480.

[6] S. Floyd, TCP and successive fast retransmits, Lawrence Berkeley Laboratory, Technical Report, February 24, 1995. available from anonymous ftp from ftp.ee.lbl.gov/papers/fastrans.ps.

[7] Van Jacobson, Congestion avoidance and control, *ACM SIGCOMM 88*, pp 273-288.

[8] Van Jacobson, modified TCP congestion avoidance algorithm, mailling list end2end-interest, received on Monday 30 April 1990.

[9] V. Jacobson, L. Peterson, L. Brakmo, S. Floyd, Problems with Arizona's Vegas, mailing list end2end-tf (discussion on TCP Vegas), 1994.

[10] S. Keshav, REAL: A network simulator, Department of Computer Science, UC Berkeley, Technical Report 88/472, 1988.

[11] 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*, North-Holland, 1994, pp. 135-150.

[12] 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 for publication.

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

Appendix

A. Appendix: Proof of Theorem 3.1

Under Reno each mini-slot time (δ_i) the ith window is incremented by one packet in a congestion avoidance phase or doubled in the slow-start phase. We have

$$W_n = \begin{cases} 2W_{n-1} & \text{if } W_n \leq W_{th} \\ W_{n-1}+1 & \text{otherwise} \end{cases} \tag{18}$$

In Vegas, during the congestion avoidance phase, the window may also remain constant, and we shall thus consider in addition the case of $W_n = W_{n-1}$. In both congestion (when the window increases) and slow-start phases, packets of window W_n are sent while those of W_{n-1} are received, only the last packet of W_n is sent $1/C$ after receiving the last Ack of W_{n-1} (if $W_n > W_{n-1}$ and $C > 2\ Thp_{out}(W_n)$, $\forall n$). Thus, the transmission time of a burst W_n (i.e. the time necessary to send packets belonging to W_n) is the receiving time of a burst W_{n-1} (i.e. the time necessary to receive (serve) packets belonging to W_{n-1}) plus $1/C$. Then, we can write

$$W_n\ Thp_{in}(W_n)^{-1} = W_{n-1} Thp_{out}(W_{n-1})^{-1} + \frac{1}{C} \tag{19}$$

Since $C >> 1$, from (19), we have

$$Thp_{in}(W_n) \cong \frac{W_n}{W_{n-1}} Thp_{out}(W_{n-1}). \tag{20}$$

We shall use the following fluid approximation. We replace the instantaneous input rate during a burst by its average over that burst, and do similarly with the output rate. We have during the burst transmission

$$Thp_{out}(W_n) = \mu \frac{Thp_{in}(W_n)}{Thp_{in}(W_n) + \lambda} \tag{21}$$

which reflects the fact that, as long as controlled packets are transmitted (the burst periods), the queue is nonempty (property Q0), and the output rate for each source is proportional to its input rate. This holds, also, when there are no silent periods (the queue doesn't empty).

(i) Consider some slot n during which the window remains constant ($W_n = W_{n-1}$, in Vegas). Thus, we have $Thp_{in}(W_n) = Thp_{out}(W_{n-1})$, combining this with (21), we get $Thp_{in}(W_n) = Thp_{out}(W_n) = \mu - \lambda$. The first packet of W_n finds an empty queue if and only if, the time it takes to reach the router, from its sending time, exceeds the service time of the remaining packets of W_{n-1} at the router. This is equivalent to

$$\tau_1 \geq \left(W_{n-1} - (\tau/2 + \tau_2) Thp_{out}(W_{n-1}) - \frac{Thp_{out}(W_{n-1})}{\mu} \right) Thp_{out}(W_{n-1})^{-1}$$
$$\iff W_n \leq T(\mu - \lambda)$$

The number of controlled packets in the queue, when it is always nonempty ($W_n > T(\mu - \lambda)$), is constant over the time, while the window remains constant. This is due to the fact that the total input rate equals to the total output rate μ, and the flow of packets is continuous over the time (there are no silent periods). Hence, all packets of W_n find a queue size of $Q(W_n)$ such that

$$Q(W_n)\frac{Thp_{in}(W_{n-1})}{Thp_{in}(W_{n-1})+\lambda} = \left(W_{n-1} - \tau Thp_{out}(W_{n-1}) - \frac{Thp_{out}(W_{n-1})}{\mu}\right) \quad (22)$$

This means that the number of controlled packets in transit, just before transmitting the first packet of W_n, equals W_{n-1}. The first packet of W_n is sent, once the first Ack of packets belonging to W_{n-1} is received. At this moment, the number of packets belonging to W_{n-1} that were served equals to $(\tau_2 + \frac{\tau}{2})\,Thp_{out}(W_{n-1}) - Thp_{out}(W_{n-1})/\mu$. The first packet of W_n spends τ_1 to reach the router and during this time, $\tau_1\ Thp_{out}(W_{n-1})$ packets of W_{n-1} will be served. Then, the number of controlled packets that remain in the queue when the first packet of W_n arrives, which is proportional to the input rate $Thp_{in}(W_{n-1})$, equals to W_{n-1} minus the number of packets belonging to W_{n-1} that were, already, served.

Since $Thp_{out}(W_n) = Thp_{out}(W_{n-1}) = \mu - \lambda$, from (22) we get

$$Q(W_n) = \left(\frac{W_n}{\mu-\lambda} - T\right)\mu.$$

Finally, we have

$$\delta_n = \overline{RTT}_n = T + \frac{Q(W_n)}{\mu} = \frac{W_n}{\mu-\lambda} = T + \epsilon_{W_n}. \quad (23)$$

For $W_n \le T(\mu-\lambda)$, $\delta_n = \overline{RTT}_n = T$.

Vegas' window cannot exceed W^v_{max}, such that $Q(W^v_{max}) = B$, from which we get (3).

It is easily checked that for W^v_{max}, $\epsilon_{W^v_{max}} = B/\mu$.

In a congestion phase under both Reno and Vegas, when the window increases, $Thp_{in}(W_n)$ and $Thp_{out}(W_n)$ are obtained from (20) and (21) by replacing W_{n-1} by $W_n - 1$. This yields, approximatively

$$Thp_{in}(W_n) \cong \mu - \lambda + \frac{\mu}{W_n - 1} \quad \text{and} \quad Thp_{out}(W_n) \cong \mu - \lambda + \frac{\lambda}{W_n}$$

Since we send one packet for every received Ack, and only for the last Ack two packets are sent, we can assume, as discussed above, that the rates are constant over a burst (a generation).

The first packet of W_n finds an empty queue if and only if

$$\tau_1 \ge \left(W_{n-1} - (\tau/2 + \tau_2)Thp_{out}(W_{n-1}) - \frac{Thp_{out}(W_{n-1})}{\mu}\right) Thp_{out}(W_{n-1})^{-1}$$

$$\Longleftrightarrow W_n - 1 \leq T(\mu - \lambda) + \frac{T\lambda}{W_{n-1}} \cong T(\mu - \lambda)$$

For $W_n - 1 \leq T(\mu - \lambda)$ (first packet of W_n finds an empty queue or the queue becomes empty after each window), the *wait time* of packet number i, $i = 0 \cdots W_n - 1$ is given by

$$i\left(Thp_{out}(W_n)^{-1} - Thp_{in}(W_n)^{-1}\right) = \frac{i}{(W_n - 1)(\mu - \lambda) + \mu}$$

The packet number $W_n - 1$ is sent $1/C$ after the $W_n - 2$nd packet then

$$\delta_n = T + \frac{W_n - 1}{(W_n - 1)(\mu - \lambda) + \mu} + \frac{1}{C} \cong T + \frac{1}{\mu - \lambda}$$

$$\text{and} \quad \overline{RTT}_n = T + \frac{1}{W_n}\left(\sum_{i=0}^{W_n - 1} \frac{i}{(W_n - 1)(\mu - \lambda) + \mu}\right) \cong T$$

W_{max} is the window size for which the round trip time of the last packet ($W_n - 1$st) equals $T + B/\mu$, from which the first equation of (4) is obtained.

When the queue is always nonempty ($W_n > T(\mu - \lambda)$), we have

$$\delta_n = W_n Thp_{out}(W_n)^{-1} = \frac{W_n}{\mu - \lambda}\left(\frac{1}{1 + \frac{\lambda}{(\mu - \lambda)W_n}}\right) \cong \frac{W_n}{\mu - \lambda}.$$

For the average round trip time over a window in the congestion phase(when the window increases), we remark that packets of the nth window W_n are sent as long as Acks of packets belonging in the window $W_n - 1$ are received. Let $t_0 = 0$ be the time we receive the Ack of the first packet of $W_n - 1$, then the send time t_s and the receive time t_r of packet number i ($i = 0 \cdots W_n - 2$) of W_n are respectively given by

$$t_s = i\frac{W_n - 1}{(W_n - 1)(\mu - \lambda) + \lambda}$$

$$\text{and} \quad t_r = (W_n - 2)\frac{W_n - 1}{(W_n - 1)(\mu - \lambda) + \lambda} + (i + 1)\frac{W_n}{W_n(\mu - \lambda) + \lambda}.$$

The round trip time of packet i is then

$$t_r - t_s = (W_n - 1)\frac{W_n - 1}{(W_n - 1)(\mu - \lambda) + \lambda} + \frac{(i + 1)\lambda}{((W_n - 1)(\mu - \lambda) + \lambda)(W_n(\mu - \lambda) + \lambda)}$$

and the last packet $W_n - 1$st is sent $1/C$ after $W_n - 2$nd packet and its Ack will be received $Thp_{out}(W_n)^{-1}$ after that of $W_n - 2$, its round trip time is, then, given by $t_r - t_s = W_n \dfrac{W_n}{W_n(\mu-\lambda)+\lambda} - \dfrac{1}{C}$. Finally we get

$$\overline{RTT}_n = \frac{1}{W_n} \sum_{i=0}^{i=W_n-1} RTT_i \cong \frac{W_n(W_n-1)}{W_n(\mu-\lambda)+\lambda} \tag{24}$$

W_{max} is the window size for which the round trip time of the last packet (W_n-1st) equals $T + B/\mu$, then we get, approximatively, (Eq 2 of) (4).

(ii) In the slow-start phase $Thp_{in}(W_n) = 2Thp_{out}(W_{n-1}) = 2Thp_{out}(W_n/2)$. Let

$$Thp_{in}(W_n) = 2Thp_{out}(W_n/2) = 2\mu - \lambda + f(W_n) \tag{25}$$

be the solution of (20) and (21). From (21) and (25) we get

$$f(W_n) = \frac{\lambda f(W_{n-1})}{2\mu + f(W_{n-1})} \tag{26}$$

The solution for this equation depends on the initial condition (which depends on C). Since after receiving the first Ack we send two packets separated by $1/C$, then we have $Thp_{in}(W_1) = Thp_{in}(2) = C$ combining this with (25) we get $f(W_1) = f(2) = C - 2\mu + \lambda$. Hence, from (26), it follows, by induction, that for $C \geq 2\mu - \lambda$, $\forall n,\ f(W_n) \geq 0$ and

$$f(W_n) = \frac{\lambda f(W_{n-1})}{2\mu} \frac{1}{1 + \frac{f(W_{n-1})}{2\mu}} \leq \frac{\lambda f(W_{n-1})}{2\mu} \leq \cdots \leq \left(\frac{\lambda}{2\mu}\right)^{n-1} f(W_1).$$

Since $\lambda < 2\mu$, we have $f(W_n) \longrightarrow 0$ when $W_n \longrightarrow +\infty$. Thus for $C \geq 2\mu - \lambda$, the solution converges to $Thp_{in}(W_n) = 2Thp_{out}(W_n/2) = 2\mu - \lambda$.

The first packet of the nth window W_n finds an empty queue if and only if

$$\tau_1 \geq \left(\frac{W_n}{2} - (\tau/2 + \tau_2)Thp_{out}(W_n/2) - \frac{Thp_{out}(W_n/2)}{\mu}\right) Thp_{out}(W_n/2)^{-1}$$
$$\Longleftrightarrow \frac{W_n}{2} \leq T(\mu - \lambda/2).$$

- When the first packet of W_n finds an empty queue $\left(\frac{W_n}{2} \leq T(\mu - \lambda/2)\right)$

The RTT of packet number $2i-2$, $i = 1 \cdots W_n/2$ (packets numbered from 0 to $W_n - 1$), exceeds T by

$$\frac{2i-2}{\mu - \lambda/2} - \frac{i-1}{\mu - \lambda/2} = \frac{i-1}{\mu - \lambda/2}$$

which is the *wait time*, time when it is served minus its arrival time. The *wait time* of packet number $2i-1$ equals

$$\frac{2i-1}{\mu - \lambda/2} - \frac{i-1}{\mu - \lambda/2} - \frac{1}{C} = \frac{i}{\mu - \lambda/2} - \frac{1}{C} \tag{27}$$

and hence, the average round trip time over a window W_n, in the slow-start phase, under Reno is given by

$$\overline{RTT}_n = T + \frac{1}{W_n} \sum_{i=1}^{W_n/2} \left(\frac{2i-1}{\mu - \lambda/2} - \frac{1}{C} \right)$$

from which we get (5). δ_n equals to the round trip time of the last packet in the window W_n plus $1/C$ (because it is sent $1/C$ after receiving the Ack of the last packet sent in the window W_{n-1}. The round trip time is T plus the wait time computed in (27), so

$$\delta_n = \left(T + \frac{W_n}{2\mu - \lambda} - \frac{1}{C} \right) + \frac{1}{C} = T + \frac{W_n}{2\mu - \lambda}$$

- When the queue is nonempty ($T(\mu - \lambda/2) < W_n/2$)

We have $\delta_n = W_n Thp_{out}(W_n)^{-1} = \dfrac{W_n}{\mu - \lambda/2}$.

Packets of the window W_n are sent once the sender begins to receive Acks of those sent when the window was $W_{n-1} = W_n/2$. Let $t_0 = 0$ be the arrival time of the Ack of the first packet sent in the previous window ($W_n/2$) and t_s *resp.* t_r the send *resp.* arrival time of a packet *resp.* its Ack. The packet number $2i-2$, ($i = 1 \cdots W_n/2$, packets numbered from 0 to $W_n - 1$), is sent when we receive the Ack of packet i of $W_n/2$, so the sent time is $t_s = \dfrac{i-1}{\mu - \lambda/2}$ and, since $Thp_{out}(W_n)^{-1} = \mu - \lambda/2$, its Ack will be received $(2i-2)Thp_{out}(W_n)^{-1}$ after the Ack of the first packet of W_n, which will be received at time $(W_n/2)Thp_{out}(W_n/2)^{-1}$, hence we have $t_r = \dfrac{W_n/2 + 2i - 2}{\mu - \lambda/2}$ and its RTT is given by

$$t_r - t_s = \frac{W_n/2 + i - 1}{\mu - \lambda/2}.$$

Packet number $2i-1$ is sent $1/C$ after packet $2i-2$ (we send two packets for one received Ack), then we have $t_s = \dfrac{i-1}{\mu-\lambda/2} + \dfrac{1}{C}$, and its Ack will be received $Thp_{out}(W_n){-}1$ after that of $2i-2$, so $t_r = \frac{W_n/2+2i-1}{\mu-\lambda/2}$, hence its RTT is given by

$$t_r - t_s = \frac{W_n/2+i}{\mu-\lambda/2} - \frac{1}{C}$$

Finally, we get

$$\overline{RTT}_n = \frac{1}{W_n}\sum_{i=1}^{W_n/2}\left(\frac{2W_n-2}{2\mu-\lambda} + \frac{2i}{\mu-\lambda/2} - \frac{1}{C}\right) = \frac{3W_n}{4\mu-2\lambda} - \frac{1}{2C}$$

(iii) Under Vegas, in the slow start phase, we distinguish two cases:
1) When the window doubles its size (once every other δ), in this case we send two packets for every received Ack.
2) When the window remains constant, we send one packet for every received Ack. The two cases are performed in an alternative way over the time. Let $Thp_{in}(W_n)$ *resp.* $Thp_{out}(W_n)$ be the rate at which packets of the nth window W_n are transmitted *resp.* served in the first case, and $\widetilde{Thp}_{in}(W_n)$, *resp.* $\widetilde{Thp}_{out}(W_n)$ the rate at which packets of the nth window W_n are transmitted *resp.* served in the second case. In the first case, we send two packets for every received Ack of a packet sent in the second case, and then we have, from (20)

$$Thp_{in}(W_n) = 2\widetilde{Thp}_{out}(W_n/2) \tag{28}$$

In the second case, we send one packet for every received Ack of a packet sent in the first case, then we have

$$\widetilde{Thp}_{in}(W_n) = Thp_{out}(W_n) \tag{29}$$

Also, we have from (21)

$$Thp_{out}(W_n) = \mu\frac{Thp_{in}(W_n)}{Thp_{in}(W_n)+\lambda} \quad \text{and} \quad \widetilde{Thp}_{out}(W_n) = \mu\frac{\widetilde{Thp}_{in}(W_n)}{\widetilde{Thp}_{in}(W_n)+\lambda} \tag{30}$$

Combining (28), (29) and (30) we get

$$Thp_{in}(W_n) = 2\mu\frac{Thp_{out}(W_n/2)}{Thp_{out}(W_n/2)+\lambda} = 2\mu\frac{\mu Thp_{in}(W_n/2)}{(\mu+\lambda)Thp_{in}(W_n/2)+\lambda^2} \tag{31}$$

Let $Thp_{in}(W_n) = \dfrac{2\mu^2 - \lambda^2}{\mu + \lambda} + f(W_n)$ be the solution of (31). Since $f(W_1) = f(2) = C$, then for $C \geq \dfrac{2\mu^2 - \lambda^2}{\mu + \lambda}$ we can recursively prove that ($W_n = 2W_{n-2}$ and $W_n = W_{n-1}$)

$$f(W_n) = \left(\frac{\lambda}{2\mu}\right)^2 f(W_n/2) \frac{1}{1 + \frac{\mu+\lambda}{2\mu^2} f(W_n/2)} \leq \left(\frac{\lambda}{2\mu}\right)^{n-2} f(W_2)$$

Since $\lambda < 2\mu$, then $f(W_n) \longrightarrow 0$ when $W_n \longrightarrow +\infty$. Thus for $C \geq \dfrac{2\mu^2 - \lambda^2}{\mu + \lambda}$, the solution converges to $Thp_{in}(W_n) = \frac{2\mu^2-\lambda^2}{\mu+\lambda}$. From (28), (29) and (30) we get

$$Thp_{in}(W_n) = \frac{2\mu^2 - \lambda^2}{\mu + \lambda} \Longrightarrow Thp_{out}(W_n) = \frac{2\mu^2 - \lambda^2}{2\mu + \lambda} \tag{32}$$

$$\Longrightarrow \widetilde{Thp}_{in}(W_n) = \frac{2\mu^2 - \lambda^2}{2\mu + \lambda} \Longrightarrow \widetilde{Thp}_{out}(W_n) = \frac{\mu^2 - \lambda^2/2}{\mu + \lambda} \tag{33}$$

When the window doubles, the first packet of the window W_n finds an empty queue ($W_{n-1} = W_n/2$) if and only if

$$\tau_1 \geq \left(\frac{W_n}{2} - (\tau/2 + \tau_2)\widetilde{Thp}_{out}(W_n/2) - \frac{\widetilde{Thp}_{out}(W_n/2)}{\mu}\right) \widetilde{Thp}_{out}(W_n/2)^{-1}$$
$$\Longleftrightarrow \frac{W_n}{2} \leq T \times \widetilde{Thp}_{out}(W_n/2) = \frac{\mu^2 - \lambda^2/2}{\mu + \lambda} T$$

and when it remains constant, the first packet of W_n finds an empty queue ($W_{n-1} = W_n$) if and only if

$$\tau_1 \geq \left(W_n - (\tau/2 + \tau_2)Thp_{out}(W_n) - \frac{Thp_{out}(W_n)}{\mu}\right) Thp_{out}(W_n)^{-1}$$
$$\Longleftrightarrow W_n \leq T \times Thp_{out}(W_n) = \frac{2\mu^2 - \lambda^2}{2\mu + \lambda} T$$

Once we have the different rates in each case, we can compute the average round trip time over a given window W_n ($\overline{RTT}_n$), and the time that takes this window to be acknowledged (δ_n), by following the same steps as in the case of Reno. ∎

B. Appendix: Proof of Corollary 3.2

(i) Vegas' congestion window stabilizes when it reaches W_{vmax} such that

$$\frac{\alpha}{T} < \frac{W_{vmax}}{T} - \frac{W_{vmax}}{T + \epsilon_{W_{vmax}}} \leq \frac{\beta}{T} \tag{34}$$

$$\Longleftrightarrow \frac{\alpha T}{W_{vmax} - \alpha} < \epsilon_{W_{vmax}} \leq \frac{\beta T}{W_{vmax} - \beta} \tag{35}$$

where $T + \epsilon_{W_{vmax}}$ is the average round trip time when the nth window W_n equals to W_{vmax}. W_{vmax} is then a solution of the following inequality; by substituting (23) into (35), we get

$$\frac{\alpha T}{W_{vmax} - \alpha} < \left(\frac{W_{vmax}}{\mu - \lambda} - T\right) \leq \frac{\beta T}{W_{vmax} - \beta} \tag{36}$$

Finally, (11) is obtained by solving this inequality.

(ii) In the slow start, Vegas' congestion window doubles its size every other δ if $(Expected - Actual)T \leq \gamma$ (γ is always taken equal to 1), hence W_{th} satisfies

$$\overline{RTT}_n - T > \frac{T\gamma}{W_{th} - \gamma}. \tag{37}$$

By substituting (7) into (37), we get two cases:

1) If $T\dfrac{\mu^2 - \lambda^2/2}{\mu + \lambda} \geq W_{th}/2$, W_{th} is solution of

$$W_{th}^2 - \left(\gamma + \frac{4\mu^2 - 2\lambda^2}{2C\mu} - \frac{\lambda}{\mu}\right) W_{th} - \gamma \frac{(2TC - 1)(4\mu^2 - 2\lambda^2) + 2C\lambda}{2C\mu} > 0$$

2) Otherwise, W_{th} satisfies

$$W_{th}^2 - \left(\frac{(3\gamma\mu + (2\gamma + 1)\lambda)C + (2CT + 1)(2\mu^2 - \lambda^2)}{C(3\mu + 2\lambda)}\right) W_{th} + \gamma\left(\frac{C\lambda + 2\mu^2 - \lambda^2}{C(3\mu + 2\lambda)}\right) > 0$$

Solving these inequalities with equalities instead of inequalities yield (13). Recall that W doubles every other δ, so that W_{th} equals to the smallest power of 2 which satisfies $W > Min_{th}$. This yields (12). ∎

C. Appendix: Proof of Corollary 3.3

Under Vegas as we assumed previously that the window stabilizes, and since the duration of the slow-start-phase is negligible, then we have $\overline{Thp} = \mu - \lambda$. Thus the average round trip time is $\overline{rtt} = \dfrac{W_{vmax}}{\mu - \lambda}$.

It is always easy to compute the amount data successfully transmitted and the average round trip time, once we define exactly the transient behavior. To simplify the calculations, we made the assumption (ii) for Reno where the case of multiple drops is not taken into account [12]. We supposed that no more than one loss in a window (burst) can occur, so that the slow-start phase is performed only one time during a session. In what follows we will focus our calculations on the "steady state" behavior. Indeed, since the slow-start is performed only one time and its duration is relatively short, this phase can be neglected and only the periodic behavior is "interesting" (congestion phase).

In a congestion phase, we have to compute the amount of data sent (N_1), the time necessary for these data to be acknowledged (T_1) and the average round trip time (per packet) (R_1), from the beginning of a given cycle ($W = W_{max}/2$) till a loss occurs ($W = W_{max}$). We distinguish two cases:

- $W_{max}/2 > T(\mu - \lambda)$ (The queue doesn't empty), we have:

$$\begin{aligned} T_1 &= \sum_{i=W_{max}/2}^{W_{max}} \delta_i = \sum_{i=W_{max}/2}^{W_{max}} \frac{i}{\mu - \lambda} \quad \text{and} \quad N_1 = \sum_{i=W_{max}/2}^{W_{max}} i \\ R_1 &= \frac{1}{N_1} \sum_{i=W_{max}/2}^{W_{max}} i \times \overline{RTT}_i = \frac{1}{N_1} \sum_{i=W_{max}/2}^{W_{max}} i \frac{i(i-1)}{i(\mu - \lambda) + \lambda} \\ &\cong \frac{1}{N_1} \sum_{i=W_{max}/2}^{W_{max}} \frac{i^2 - i}{\mu - \lambda}. \end{aligned}$$

- $W_{max}/2 < T(\mu - \lambda)$ (The queue empties after each cycle), we have

$$N_1 = N_{21} + N_{22} \qquad T_1 = T_{21} + T_{22} \qquad R_1 = \frac{N_{21}R_{21} + N_{22}R_{22}}{N_1}.$$

Where the indices 21 correspond to the period where the queue is empty and 22 refer to the period where the queue is not empty. During the period where the queue is empty, we have

$$N_{21} = \sum_{i=W_{max}/2}^{T(\mu-\lambda)} i \qquad \text{and} \qquad T_{21} = \sum_{i=W_{max}/2}^{T(\mu-\lambda)} \left(T + \frac{1}{\mu - \lambda}\right)$$

the average round trip time, since the queue is empty, equals to $R_{21} = T$. During the second period (the queue is not empty), we have

$$N_{22} = \sum_{i=T(\mu-\lambda)+1}^{W_{max}} i \quad \text{and} \quad T_{22} = \sum_{i=T(\mu-\lambda)+1}^{W_{max}} \frac{i}{\mu - \lambda}$$

$$R_{22} = \frac{1}{N_{22}} \sum_{i=T(\mu-\lambda)+1}^{W_{max}} i \times \overline{RTT}_i = \frac{1}{N_{22}} \sum_{i=T(\mu-\lambda)+1}^{W_{max}} i \frac{i(i-1)}{i(\mu - \lambda) + \lambda}$$

$$\cong \frac{1}{N_{22}} \sum_{i=T(\mu-\lambda)+1}^{W_{max}} \frac{i^2 - i}{\mu - \lambda} \quad \blacksquare$$