Fairness Comparisons between TCP Reno and TCP Vegas for Future Deployment of TCP Vegas

Kenji Kurata† Go Hasegawa‡ Masayuki Murata†

†Department of Informatics and Mathematical Science
Graduate School of Engineering Science, Osaka University
1-3, Machikaneyama, Toyonaka, Osaka 560-8531, Japan
Phone: +81-6-6850-6616, Fax: +81-6-6850-6589
E-mail: {k-kurata, murata}@ics.es.osaka-u.ac.jp

‡Faculty of Economics, Osaka University
1-7, Machikaneyama, Toyonaka, Osaka 560-0043, Japan
Phone: +81-6-6850-5233
E-mail: hasegawa@econ.osaka-u.ac.jp

Abstract TCP Vegas version is expected to achieve higher throughput than TCP Tahoe and Reno versions, which are currently used in the Internet. However, we need to consider a migration path of TCP Vegas when it is deployed in the Internet. In this paper, we focus on the situation where multiple TCP Reno and Vegas connections coexist at the bottleneck router, by which the fairness property is investigated to seek the possibility of future deployment of TCP Vegas. We consider drop-tail and RED (Random Early Detection) algorithms as buffering discipline at the router, and evaluate the effect of RED algorithm on fairness enhancement. From the analysis and the simulation results, we have found the results that the fairness between TCP Reno and Vegas can not be kept at all with drop-tail router. Although RED algorithm improves the fairness to some degree, there are inevitable trade-off between fairness and throughput.

1 Introduction

TCP (Transmission Control Protocol) is widely used by many Internet services including HTTP (and World Wide Web) and FTP (File Transfer Protocol). Thus, even if the network infrastructure may change in the future Internet, TCP and its applications would be likely to be continuously used. However, TCP Tahoe and Reno versions (and their variants), which are widely used in the current Internet, are not perfect in terms of throughput and fairness among connections, as having been shown in the past literatures. Therefore, active researches on TCP have made great efforts to propose many improvement mechanisms of TCP (for example, see [1, 2, 3, 4] and the references therein).

Among them, TCP Vegas version [5, 6] is considered to be one of the promising mechanisms in its high performance. TCP Vegas enhances the congestion avoidance algorithm of TCP Reno. More specifically, TCP Vegas dynamically increases/decreases its sending window size according to observed RTTs (Round Trip Times) of sending packets, whereas TCP Tahoe/Reno only continues increasing its window size until packet loss is detected. For instance, the authors in [5] concludes through simulation and implementation experiments that TCP Vegas can obtain 40% higher throughput than TCP Reno.

However, we need to consider a migration path when the new protocol is deployed in the operating network, i.e., the Internet. That is, it is important to investigate the effect of traditional TCP versions (Tahoe and Reno) on TCP Vegas in the case where those different versions of TCP co-exist in the network. The authors in [7] have pointed out that when connections of TCP Reno version and Vegas version share the bottleneck router, the Ve-

gas connection may suffer from significant unfairness. However, the authors have assumed that only a single TCP Reno connection shares the link with another TCP Vegas connection, and no solution has been provided for fairness enhancement.

In this paper, therefore, we focus on the situation where multiple TCP Reno and Vegas connections coexist at the bottleneck router, and investigate the fairness property between two versions of TCP to seek the possibility of future deployment of TCP Vegas in the Internet. One important point is that the RED mechanism [8] is now being introduced while the original Vegas does not assume it. It may or may not be suitable to the RED mechanism. We therefore consider two mechanisms, drop-tail and RED routers, in our study. One of the contributions in this paper is to derive analysis results of the throughput of TCP Reno and Vegas in such situation. We further present the accuracy of our analysis by comparing the analysis results with the simulation results.

Through the analysis and simulation results, we evaluate the essential fairness property between TCP Reno and Vegas as follows. TCP Vegas receives significant unfair throughput compared with TCP Reno, when the router employs the drop-tail router. When the RED algorithm is applied, the fairness can be improved to some degree, but there exists an inevitable trade-off between fairness and throughput. That is, if the packet dropping probability of RED becomes large, fairness between TCP Reno and Vegas is improved, but the total throughput is degraded at the same time.

The rest of this paper is organized as follows. Section 2 briefly introduces the congestion control mechanisms of TCP Reno and TCP Vegas. We next describe the network model used in our analysis and simulation experiments in Section 3. Section 4 shows the analysis results of fairness between the two versions of TCP, which are validated by the simulation results in Section 5. Finally, we conclude our presentation and present some future works in Section 6.

2 Congestion Control Mechanisms of TCP

In this paper, we consider two versions of TCP; TCP Reno and Vegas versions. For detailed explanation, refer to [9] for TCP Reno and [5, 6] for TCP Vegas.

TCP adopts a window-based flow control, which controls the number of on-the-fly packets in the network. The source terminal is allowed to send the number of packets given by its window size. The current window size of the source terminal is often denoted by *cwnd*. The window size is updated at the receipt of ACK (ACKnowledgement) packet. The key idea of the congestion avoidance mechanism of TCP is to dynamically control the window size according to severity of the congestion in the network. In what follows, we denote the current window size at time t by $cwnd(t)$.

2.1 TCP Reno

In TCP Reno, the window size is cyclically changed. The window size continues to be increased until packet loss occurs. TCP Reno has two phases in increasing its window size; Slow Start Phase and Congestion Avoidance Phase. When an ACK packet is received by TCP at the server side at time $t + t_A$ [sec], $cwnd(t + t_A)$ is updated from $cwnd(t)$ as follows (see, e.g., [9]);

$$cwnd(t+t_A) = \begin{cases} (\text{Slow Start Phase :}) & \\ cwnd(t)+1, & \text{if } cwnd(t) < ssth; \\ (\text{Congestion Avoidance Phase :}) & \\ cwnd(t)+\dfrac{1}{cwnd(t)}, & \text{if } cwnd(t) \geq ssth; \end{cases} \quad (1)$$

where $ssth$ [packets] is the threshold value at which TCP changes its phase from Slow Start Phase to Congestion Avoidance Phase. When packet loss is detected by retransmission timeout expiration [9], $cwnd(t)$ and $ssth$ are updated as;

$$cwnd(t) = 1 \quad (2)$$

$$ssth = \frac{cwnd(t)}{2} \quad (3)$$

On the other hand, When TCP detects packet loss by fast retransmit algorithm [9], it changes

$cwnd(t)$ and $ssth$ as follows;

$$ssth = \frac{cwnd(t)}{2} \quad (4)$$

$$cwnd(t) = ssth \quad (5)$$

TCP Reno then enters Fast Recovery Phase [9] if the packet loss is found by fast retransmit algorithm. In this phase, the window size is increased by one packet when a duplicate ACK packet is received, and $cwnd(t)$ is restored to $ssth$ when the non–duplicate ACK packet corresponding to the retransmitted packet is received.

2.2 TCP Vegas

In TCP Reno (and the older version Tahoe), the window size continues to be increased until packet loss occurs due to congestion. Then, when the window size is throttled because of packet loss, the throughput of the connection may degrade. However, it cannot be avoided because of an essential nature of the congestion control mechanism adopted in TCP Reno. That is, it can detect network congestion *only* by packet loss. However, throttling the window size is not adequate when the TCP connection itself causes the congestion because of its too large window size. If the window size is appropriately controlled such that the packet loss does not occur in the network, the throughput degradation due to the throttled window can be avoided. This is the reason that TCP Vegas was introduced.

TCP Vegas employs another mechanism, in which it controls its window size by observing RTTs (Round Trip Time) of packets that the connection has sent before. If observed RTTs become large, TCP Vegas recognizes that the network begins to be congested, and throttles the window size. If RTTs become small, on the other hand, TCP Vegas determines that the network is relieved from the congestion, and increases the window size again. Then, the window size in an ideal situation becomes converged to the appropriate value. More specifically, in Congestion Avoidance Phase, the window size is updated as;

$$cwnd(t + t_A) = \begin{cases} cwnd(t) + 1, & \text{if } diff < \frac{\alpha}{base\_rtt} \\ cwnd(t), & \text{if } \frac{\alpha}{base\_rtt} \leq diff \leq \frac{\beta}{base\_rtt} \\ cwnd(t) - 1, & \text{if } \frac{\beta}{base\_rtt} < diff \end{cases} \quad (6)$$

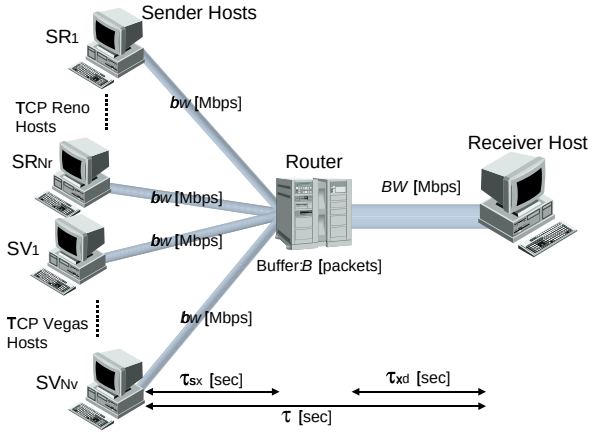


Figure 1: Network Model

$$diff = \frac{cwnd(t)}{base\_rtt} - \frac{cwnd(t)}{rtt} \quad (7)$$

where rtt [sec] is an observed round trip time, $base\_rtt$ [sec] is the smallest value of observed RTTs, and α and β are some constant values.

TCP Vegas has an another feature in its congestion control algorithm. That is *slow* Slow Start mechanism. The rate of increasing its window size in Slow Start Phase is a half of that in TCP Tahoe and TCP Reno. Namely, the window size is incremented at every other time an ACK packet is received. Note that Equation (6) used in TCP Vegas indicates that if observed RTTs of the packets are identical, the window size remains unchanged.

According to [5], TCP Vegas can achieve over 40% higher throughput than TCP Reno, which has been confirmed through simulation and implementation experiments. However, it has not been validated whether TCP Vegas could work well with TCP Reno or not. One of the things we want to do in this paper is to investigate that, that is, the fairness between the two versions of TCP when they co-exist in the network. We believe that it is very important to deploy TCP Vegas to the future Internet.

3 Network Model

Figure 1 shows the network model used in this paper. It consists of N_r sender hosts using TCP Reno (SR_1, ... SR_{N_r}), N_v sender hosts using TCP Vegas (SV_1, ... SV_{N_v}), a receiver host, a intermediate router and links that connect the router and the

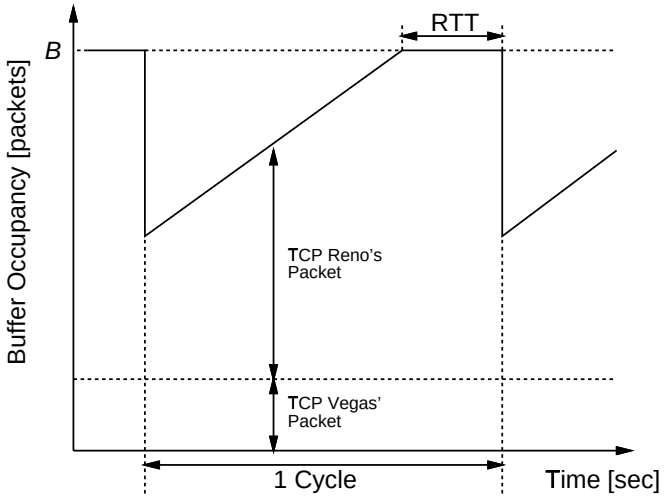


Figure 2: The Typical Change of Total Buffer Occupancy at the Drop-tail Router

sender/receiver hosts. The bandwidth of each link between sender hosts and the router is bw [Mbps]. The bandwidth of the bottleneck link between the router and the receiver host is BW [Mbps] = μ [packets/sec]. The size of the buffer in the router is B [packets], and the propagation delay between sender hosts and the router, and that between the router and the receiver host are τ_{sx} [sec] and τ_{xd}[sec], respectively. We denote the total propagation delay between the sender hosts and the receiver host by τ, which equals $\tau_{sx} + \tau_{xd}$. As the buffering discipline at the router, we use drop-tail and RED (Random Early Detection) [8] algorithms. We further assume that the sender hosts have infinite amount of sending data.

4 Analysis

In what follows, we use the network model depicted in Figure 1, and derive average throughput of each TCP connection through mathematical analysis. In the analysis, we assume that the throughput of each connection becomes proportional to buffer occupancy at the router. This assumption is appropriate for the drop-tail router, and for the RED router where the additional function is attached to the drop-tail router.

4.1 Case of Drop-tail Router

Figure 2, we model the typical change of the total number of packets in the router buffer if we use drop-tail algorithm. Since TCP Reno connections continue to increase their window sizes until packet loss occurs at the buffer, the change of the window size also has cycles in the this case, where the TCP Reno connections co-exist with the TCP Vegas connection in the bottleneck touter. Furthermore, if we assume that all packet losses can be detected by Fast Retransmit algorithm [9], It takes 1 RTT (Round Trip Time) [sec] for sender side TCP to detect the packet loss until the packet loss really occurs at the route buffer. This is depicted the flat part of the change of the window size in Figure 2.

TCP Vegas connections, on the other hand, control their window size according to the observed RTTs of sending packets. In more detail, they try to keep the number of buffered packets in the router buffer from α to β [packets] [4]. If RTTs continue to become larger, TCP Vegas connections continue to decrease their window size. In this case, since TCP Reno connections continue increasing their window size regardless the increase of RTT, the RTTs for TCP Vegas connections also become larger. This results that the window sizes of the TCP Vegas connections decrease to reach within the range from α to β [packets], in accordance with Eq. (6). Therefore, W_v [packets], the total window size of the TCP Vegas connections, is obtained as follows;

$$N_v \cdot \alpha \quad < W_v < \quad N_v \cdot \beta \tag{8}$$

As the reasonable assumption, we determine $\overline{W_v}$ [packets], the average value of W_v, from Eq. (8) as follows;

$$\overline{W_v} \quad = \quad N_v \cdot \frac{\alpha + \beta}{2} \tag{9}$$

On the other hand, TCP Reno connections continue increasing their window size until the router buffer becomes full. Accordingly, W_r [packets], the total window size of the TCP Reno connections when packet loss occurs at the router buffer, can be calculated;

$$W_r \quad = \quad 2\tau\mu + B - W_v \tag{10}$$

The number of packet losses in one buffer overflow becomes N_r [packets], since from Eq. (1),

the window sizes of TCP Reno connections are increased by 1 [packet/RTT], according to the Congestion Avoidance Phase explained in Section 2. When we assume that packet loss probability for each connection is proportional to its window size, we can obtain L_r [packets] and Lv [packets], which are the number of packet losses of TCP Reno and Vegas connections in the one buffer overflow, as follows;

$$L_r = N_r \cdot \frac{W_r}{W_r + W_v} \quad (11)$$

$$L_v = N_r \cdot \frac{W_v}{W_r + W_v} \quad (12)$$

The TCP Reno connections which detect the packet loss halve their window sizes according to the fast retransmit algorithm. Therefore, W_r' [packets], the total window size of the TCP Reno connections just after the buffer overflow, can be calculated from Eq. (1) and (11);

$$W_r' = \frac{1}{2} \cdot \frac{W_r}{N_r} \cdot L_r + \frac{W_r}{N_r} \cdot (N_r - L_r) \quad (13)$$

$$= \frac{W_r + 2W_v}{2(W_r + W_v)} \cdot W_r \quad (14)$$

From Figure 2 and Eq. (1), we can derive $\overline{W_r}$ [packets], which is the average value of the total window size of the TCP Reno connections;

$$\overline{W_r} = \frac{\frac{1}{2}(W_r + W_r')\frac{W_r - W_r'}{N_r} + W_r}{\frac{W_r - W_r'}{N_r} + 1} \quad (15)$$

Accordingly, we can obtain $\overline{B_r}$ [packets] and $\overline{B_v}$ [packets], the average number of packets at the router buffer of each version of TCP, as follows;

$$\overline{B_r} = \overline{W_r} \cdot \frac{B}{2\tau\mu + B} \quad (16)$$

$$\overline{B_v} = \overline{W_v} \cdot \frac{B}{2\tau\mu + B} \quad (17)$$

Finally, we can calculate ρ_r [packets/sec], ρ_v [packets/sec], the average throughput of the connections of the two versions of TCP, since we have assumed that they become proportional to the buffer occupancy at the router. The result is as follows;

$$\rho_r = \mu \cdot \frac{B_r}{B_r + B_v} \quad (18)$$

$$\rho_v = \mu \cdot \frac{B_v}{B_r + B_v} \quad (19)$$

4.2 Case of RED Router

RED algorithm drops incoming packets at the preset probability when the number of packets in the buffer exceeds a certain threshold value [8]. For simplicity of the following analysis, it is assumed that all packet loss are occurred at the probability p by the RED algorithm, and no buffer overflow takes place.

Even with the RED algorithm, the TCP Reno connections continue increasing their window sizes until packet loss occurs. Therefore, as in the drop-tail case, the TCP Vegas connections can not open their window sizes and keep them ranging from α to β. Therefore, the following equations are satisfied for W_v and $\overline{W_v}$;

$$N_v \cdot \alpha < W_v < N_v \cdot \beta \quad (20)$$

$$\overline{W_v} = N_v \cdot \frac{\alpha + \beta}{2} \quad (21)$$

TCP Reno connections, on the other hand, change their window size cyclically triggered by packet losses, as in the drop-tail router case. Since all arriving packets are dropped at the router with probability p by our assumption, the connection can send $1/p$ packets in one cycle (between the events of packet losses) on the average. We define the number of packets transmitted during one cycle as N_p, that is, $N_p = 1/p$. Different from the previous Subsection of drop-tail router case, we focus on a certain TCP Reno connection because we assume that all TCP Reno connections behave identically under the RED's stochastic packet dropping algorithm.

Although the RED algorithm can eliminate the bursty packet losses leading to TCP's retransmission timeout expiration, timeout expiration cannot be avoided perfectly [10]. Even if timeout expiration rarely happens, the effect of timeout expiration on throughput is large. Therefore, we consider the throughput degradation caused by retransmission timeout expiration. We denote the probability of occurring timeout expiration in the

window by p_{to}. As we denote the average value of the window size of a certain TCP Reno connection when packet loss is detected by $\overline{w_r}$, we can determine p_{to} according to the following simple equation;

$$p_{to} = \sum_{i=2}^{\overline{w_r}} \binom{\overline{w_r}}{i} \cdot p^i \cdot (1-p)^{\overline{w_r}-i} \quad (22)$$

In what follows, we distinguish two methods of detecting packet loss; retransmission timeout expiration (TO case) and fast retransmit (FR case), because the two cases have the different algorithms of changing the window size.

If retransmission timeout expiration occurs (TO case), the window size is reset to 1 [packet] and it is updated according to the Slow Start Phase (Eq. (1)), until it reaches $\overline{w_r}/2$ [packets]. From Eq. (1), we can derive $T_{to,1}$ [sec], which is the time duration of the Slow Start Phase, and $A_{to,1}$ [packets], which is the number of packets transmitted in the Slow Start Phase. That is,

$$T_{to,1} = rtt \cdot \log_2 \frac{\overline{w_r}}{2} \quad (23)$$

$$A_{to,1} = \frac{\overline{w_r}}{2} - 1 \quad (24)$$

where rtt [sec] is the mean value of RTT of sending packets. Furthermore, we can easily obtain $T_{to,2}$ [sec] and $A_{to,2}$ [packets], which are the time duration and the number of transmitted packets in the following Congestion Avoidance Phase from Eq. (1).

$$T_{to,2} = rtt \cdot \left(\overline{w_r} - \frac{\overline{w_r}}{2}\right) \quad (25)$$

$$A_{to,2} = \frac{1}{2}\left(\overline{w_r} + \frac{\overline{w_r}}{2}\right)\left(\overline{w_r} - \frac{\overline{w_r}}{2}\right) \quad (26)$$

These equations are determined from that the window size is increased by 1 [packet] per RTT [sec] in the Congestion Avoidance Phase (Eq. (1)).

On the other hand, if the TCP Reno connection detects the packet loss by Fast Retransmit algorithm (FR case), The window size is halved to $\overline{w_r}/2$, and the Congestion Avoidance Phase starts again. That is, the time durations and the number of transmitted packets in the Slow Start Phase ($T_{fr,1}$ and $A_{fr,1}$) and the Congestion Avoidance Phase ($T_{fr,2}$ and $A_{fr,2}$) can be derived as follows;

$$T_{fr,1} = 0 \quad (27)$$

$$A_{fr,1} = 0 \quad (28)$$

$$T_{fr,2} = rtt \cdot \left(\overline{w_r} - \frac{\overline{w_r}}{2}\right) \quad (29)$$

$$A_{fr,2} = \frac{1}{2}\left(\overline{w_r} + \frac{\overline{w_r}}{2}\right)\left(\overline{w_r} - \frac{\overline{w_r}}{2}\right) \quad (30)$$

Consequently, the following equations are satisfied for the number of transmitted packets in one cycle, and $\overline{w_r}$ from Eqs.(23)-(30);

$$\frac{1}{p} = p_{to}(A_{to,1} + A_{to,2}) + (1-p_{to})(A_{fr,1} + A_{fr,2}) \quad (31)$$

$$\overline{w_r} = rtt \cdot p_{to}\left(\frac{A_{to,1} + A_{to,2}}{T_{to,1} + T_{to,2} + rto}\right) + rtt \cdot (1-p_{to})\left(\frac{A_{fr,1} + A_{fr,2}}{T_{fr,1} + T_{fr,2}}\right) \quad (32)$$

where rto [sec] is the retransmission timeout value of the connection. Since we can obtain p_{to} and $\overline{w_r}$ by solving Eqs. (31) and (32), the total window size of all TCP Reno connections, $\overline{W_r}$, can be easily obtained as follows;

$$\overline{W_r} = N_r \cdot \overline{w_r} \quad (33)$$

Finally, ρ_r and ρ_v in the RED case can be determined as similarly to the drop-tail router case, from Eqs.(16)-(18), (20), and (33).

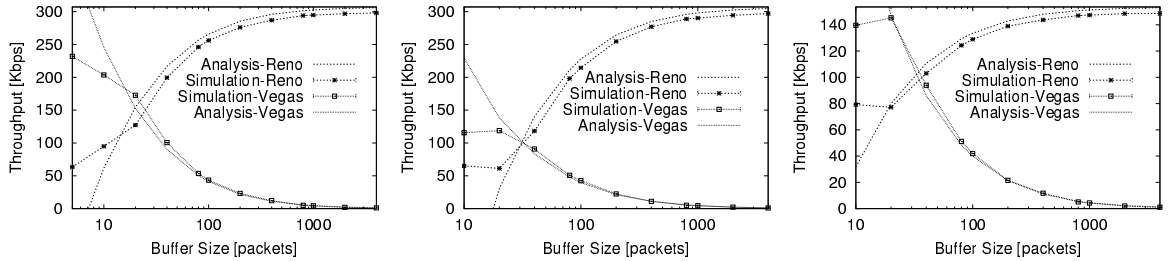
5 Numerical Examples and Discussions

We next show some numerical examples of the analysis results, which is validated by comparing them with simulation results. Furthermore, we present some discussions on the fairness between the two versions of TCP, using the numerical results.

In what follows, we use the network model depicted in Figure 1, and set τ_{sx} = 0.0015 [sec], τ_{xd} = 0.005 [sec], bw = 10 [Mbps] and BW = 1.5 [Mbps]. For the RED router, we set the threshold values: th_{min} = 5 [packets] and th_{max} = 0.6·B [packets].

5.1 Case of Drop-Tail Router

Figure 3 shows the average throughput of the TCP Reno connections and the TCP Vegas connections, as a function of the router buffer size



(a) N_r = 5, N_v = 5. (b) N_r = 5, N_v = 10. (c) N_r = 10, N_v = 5.

Figure 3: Case of Drop-Tail Router

B [packets] in drop-tail router case. We consider three cases of the number of connections of TCP Reno and Vegas (N_r and N_v); Figure 3(a) for N_r = 5, N_v = 5, Figure 3(b) for N_r =5, N_v = 10, and Figure 3(c) for N_r = 10 N_v = 5. In these figures, we show both of the analysis results and the simulation results for validating our analysis in Section 4. We can say from these figures that our analysis gives appropriate estimations of throughput, regardless of the number of connections of the two versions of TCP. However, especially when the router buffer size is very small ($<$ 20 [packets]), the analysis results under-estimate the throughput of the TCP Reno connections, and over-estimate that of TCP Vegas connections. This is because the assumption in the analysis, that the window sizes of TCP Vegas connections are fixed at $\overline{W_v}$ = $(\alpha+\beta)/2$, can not be satisfied when the buffer size is too small.

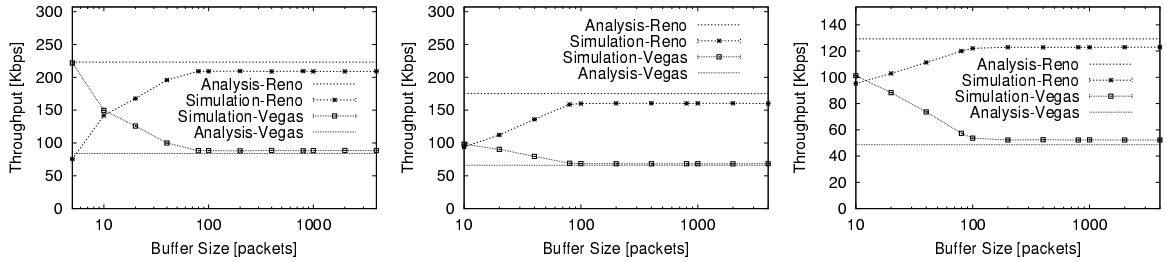
The another important observation obtained from these figures is that the TCP Vegas connections suffer from significantly low throughput, compared with the TCP Reno connections. This is because of the difference of buffer occupancy at the router. The TCP Reno connections can increase their window sizes until the buffer becomes full and packet loss occurs. On the other hand, the TCP Vegas connections can not inflate their window size larger than β, as have pointed out in the Section 4. That is, the larger the router buffer size becomes, the worse the fairness between the TCP Reno connections and the TCP Vegas connections becomes.

In short, this serious unfairness is caused by the difference of the congestion control algorithm of the two versions of TCP, and the drop-tail algorithm at the router buffer. We conclude that the drop-tail router can not provide fairness between TCP Reno and TCP Vegas connections when they share the bottleneck router in the network.

5.2 Case of RED Router

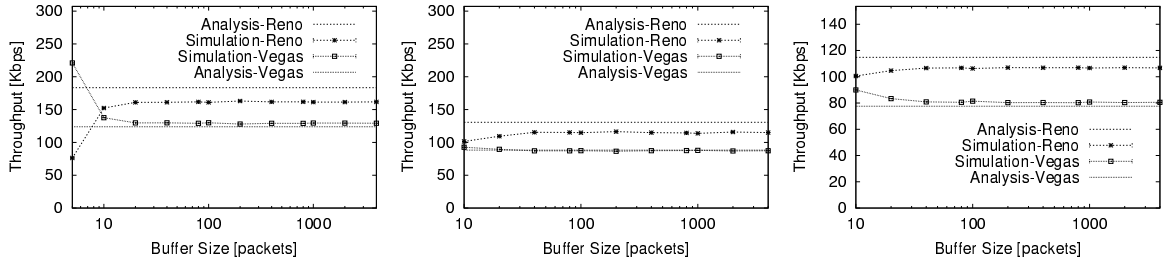
Figure 4 shows the result of the case where RED algorithm is adopted at the router buffer. We first set p, the packet dropping probability, to 1/30. In the figure, we can see that the analysis results are not effected by the router buffer size. This is because we have assumed in our analysis that the packet dropping probability is constant, and that all packet drops are caused by the stochastic dropping of RED algorithm, not by the buffer overflow. On the other hand, we can also find from this figure that the simulation results are affected by the buffer size, especially when the buffer size is small. This is because the packet loss occurs by buffer overflow, as well as by the stochastic packet dropping of the RED algorithm. Taking this observations into consideration, we can again say that our analysis results can approximate the throughputs of the connections of the two versions of TCP with the appropriate accuracy, especially when the buffer size is large.

We further present some discussions from these figures that the fairness between the two versions of TCP greatly improves, compared with the case of drop-tail router. This can be explained as follows. With the RED algorithm, the TCP Reno connections can not inflate their window sizes until the router buffer becomes fully-utilized, since packet loss occurs before the buffer becomes full, which is caused by RED algorithm. That results in



(a) N_r = 5,N_v = 5. (b) N_r = 5, N_v = 10. (c) N_r = 10, N_v = 5.

Figure 4: Case of RED Router: $p = \frac{1}{30}$



(a) N_r = 5, N_v = 5. (b) N_r = 5, N_v = 10. (c) N_r = 10, N_v = 5.

Figure 5: Case of RED Router: $p = \frac{1}{10}$

the deterioration of buffer occupancy of the TCP Reno connections, that decreases the difference of the throughput of the TCP Reno and Vegas connections.

From the above discussion, we may expect that if the packet dropping probability is further increased, the fairness can be further improved because the window sizes of the TCP Reno connections becomes still smaller. This observation can be confirmed by Figure 5, where we increase the packet dropping probability to 1/10. We can see the further fairness enhancement by comparing this figure with Figure 4.

As we can naturally guess, however, if the packet dropping probability of RED algorithm is set too high for further fairness improvement, we can not avoid the degradation of the total throughput. Figure 6 shows the simulation results of the throughput of the TCP Reno connections , that of the TCP Vegas connections, and the total throughput at the router, as a function of p, the packet dropping probability of RED algorithm. We set

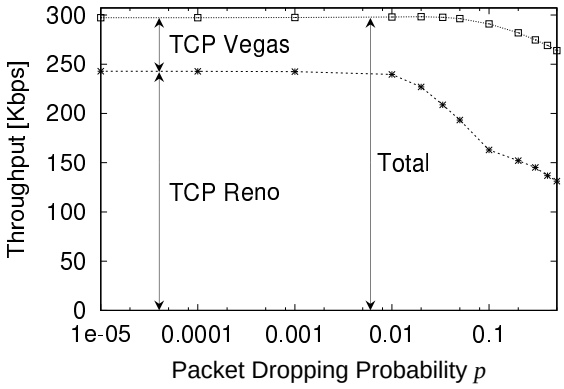


Figure 6: Throughput vs. Packet Dropping Probability of RED algorithm

N_r = 5, N_v = 5, and B = 100 [packets] to obtain this figure. We can see from the figure that when the packet dropping probability becomes too large (> 0.01), the fairness between the two versions of TCP improves, while the total throughput degrades. In other words, there is an inevitable

trade-off between fairness and throughput in the RED algorithm, therefore it is difficult to set p to appropriate value.

6 Conclusion

In this paper, we have investigated the fairness between TCP Reno and Vegas, when the TCP connections of the two versions share the bottleneck link. We have derived the following results through the mathematical analysis and the simulation experiments; TCP Vegas suffers from serious performance degradation with drop-tail routers, because of the difference of buffer occupancy of the router buffer. RED routers can improve the fairness to some degree, but we have also revealed that there exists an inevitable trade-off between fairness and throughput.

As the future works, we plan to propose the two kinds of modification for further fairness enhancement. The first one is to improve the congestion control algorithm of TCP Vegas to compete equally with TCP Reno. The second is to modify the RED algorithm at the router buffer so that the router can detect mis-behaving flows, which correspond to TCP Reno connections in this research, and eliminate the unfairness by intentionally dropping more packets from the mis-behaving flows than well-behaved flows.

References

[1] Z. Wang and J. Crowcroft, "Eliminating periodic packet losses in 4.3–Tahoe BSD TCP congestion control," *ACM Computer Communication Review*, vol. 22, pp. 9–16, April 1992.

[2] M. Perloff and K. Reiss, "Improvements to TCP performance," *Communications of ACM*, vol. 38, pp. 90–100, February 1995.

[3] M. Mathis and J. Mahdavi, "Forward acknowledgment: Refining TCP congestion control," *ACM SIGCOMM Computer Communication Review*, vol. 26, pp. 281–291, October 1996.

[4] G. Hasegawa, M. Murata, and H. Miyahara, "Fairness and stability of the congestion control mechanism of TCP," in *Proceedings of IEEE INFOCOM'99*, pp. 1329–1336, March 1999.

[5] L. S. Brakmo, S. W.O'Malley, and L. L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance," in *Proceedings of ACM SIGCOMM'94*, pp. 24–35, October 1994.

[6] L. S. Brakmo and L. L. Peterson, "TCP Vegas: End to end congestion avoidance on a global Internet," *IEEE Jounal on Selected Areas in Communications*, vol. 13, pp. 1465–1480, October 1995.

[7] J. Mo, R. J. La, V. Anantharam, and J. Walrand, "Analysis and comparison of TCP reno and vegas," in *Proceedings of IEEE INFOCOM'99*, March 1999.

[8] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, pp. 397–413, August 1993.

[9] W. R. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*. Reading, Massachusetts: Addison-Wesley, 1994.

[10] K. Fall and S. Floyd, "Simulation-based comparisons of Tahoe, Reno, and SACK TCP," *ACM SIGCOMM Computer Communication Review*, vol. 26, pp. 5–21, July 1996.