Fairness and stability of congestion control mechanisms of TCP

Go Hasegawa [a], Masayuki Murata [b] and Hideo Miyahara [b]

[a] *Faculty of Economics, Osaka University, 1-7, Machikaneyama, Toyonaka, Osaka 560, Japan*
E-mail: hasegawa@econ.osaka-u.ac.jp

[b] *Department of Infomatics and Mathematical Science, Graduate School of Engineering Science, Osaka University, 1-3, Machikaneyama, Toyonaka, Osaka 560-8531, Japan*
E-mail: {murata;miyahara}@ics.es.osaka-u.ac.jp

In this paper, we focus on fairness and stability of the congestion control mechanisms adopted in several versions of TCP by investigating their time–transient behaviors through an analytic approach. In addition to TCP Tahoe and TCP Reno, we also consider TCP Vegas which has been recently proposed for higher throughput, and enhanced TCP Vegas, which is proposed in this paper for fairness enhancements. We consider the homogeneous case, where two connections have the equivalent propagation delays, and the heterogeneous case, where each connection has different propagation delay. We show that TCP Tahoe and TCP Reno can achieve fairness among connections in the homogeneous case, but cannot in the heterogeneous case. We also show that TCP Vegas can provide almost fair service among connection, but there is some unfairness caused by the essential nature of TCP Vegas. Finally, we explain the effectiveness of our enhanced TCP Vegas in terms of fairness and throughput.

1. Introduction

Inapplicabilities of the traditional transport-layer protocols such as TCP (Transmission Control Protocol) [9,11] to the future high-speed network have been repeatedly claimed in the literature. Accordingly, some new transport-layer protocols have been developed. Examples are XTP [12] and the one in [2] for ATM (Asynchronous Transfer Mode) networks. However, many of the current Internet services including HTTP (and World Wide Web) and FTP (File Transfer Protocol) use TCP. Thus, even if the network infrastructure may change in the future, many TCP applications would continuously be used. This observation leads to the active researches on TCP for high speed data transfer. See, e.g., [6,7]. However, most of past studies have concentrated on the effectiveness of TCP in spite of the fact that stability and fairness are other important issues, and those sometimes become more essential than effectiveness [10].

In this paper, we focus on stability and fairness of several versions of TCP through an analysis. To make clear the essential nature of the congestion control mechanisms of each version of TCP, we use a rather simple model, where two connections share the bottleneck bandwidth. We will present some findings through the analytic approach in this paper. In the homogeneous case, where two TCP connections have identical

propagation delays, we point out that current versions of TCP (Tahoe and Reno versions) can give a reasonably fair service at the expense of stability. Here, by "fair", we mean that by dynamically adjusting the window size of TCP, throughputs of connections sharing the bottleneck bandwidth is close. On the other hand, a more recently proposed TCP Vegas [3,4], which adopts a different congestion control mechanism from TCP Tahoe and Reno, can offer higher performance and the stable operation; i.e., the window size is converged to a fixed value if the number of connections is not varied and each connection continuously transmits segments (packets). However, it sometimes fails to obtain the fairness among connections, that is, the throughputs of connections are converged to different values. Based on our results, we will propose an enhanced version of TCP Vegas which can improve fairness among connections. Our solution is simple. We modify TCP Vegas in such a way that convergence of the window size of the connection is not allowed. By this mechanism, one promising property of original TCP Vegas that the TCP window size is stabilized is lost, but the fairness among connections can be achieved. It seems to be a good example that the fairness and stability (and effectiveness) cannot be achieved at the same time within networks.

We also investigate the heterogeneous case where two connections have different propagation delays. In this case, traditional TCP Tahoe and TCP Reno cannot obtain fairness among connections because of their inherent control mechanisms of the window size; the rate of increasing window size depends on the round trip time of the connection. TCP Vegas gives a better fairness property as investigators expected. However, it still has pitfalls as we will explore in the later. Then, our enhanced TCP Vegas can achieve fairness among connections even with different propagation delays.

This paper is organized as follows. We first describe the network model that we will use in our analysis and simulation in section 2. Next, we briefly introduce the congestion control mechanisms of TCP Tahoe, TCP Reno, TCP Vegas in section 3. In section 4, we show our analysis methods, by which we will reveal the characteristics of the above three versions, propose our enhanced TCP Vegas, and show its effectiveness. Finally, in section 5, we make some concluding remarks.

2. Network model

The network model that we will use in the analysis and simulation is depicted in figure 1. The model consists of two sources (SES1, SES2), two destinations (DES, DES2), two intermediate switches (or routers) (SW1, SW2), and links interconnecting between the end stations and switches. We consider two connections; connection 1 from SES1 to DES1, and connection 2 from SES2 to DES2. Both connections are established via SW1 and SW2, and the link between SW1 and SW2 is shared between two connections. The bandwidth of the shared link is μ [segment/s]. The buffer size of SW1 is B [segments]. The propagation delays between SESi and DESi are τ_i ($i = 1, 2$).

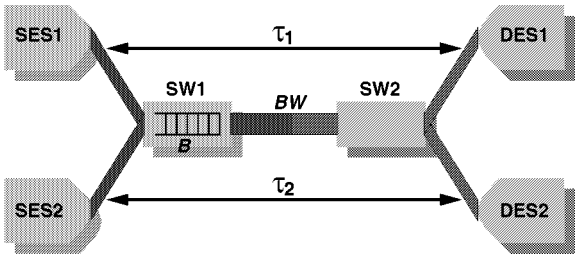


Figure 1. Network model.

In analysis and simulation, we consider the situation that connection 1 starts to transfer data segments at first, and connection 2 joins the network afterward. Each SES transmits data segments according to the TCP protocol. It is assumed that each SES is a greedy source, that is, each SES has infinite data to transmit. TCP segment size is fixed at m [bytes]. Then, we will focus on the dynamics of congestion window size as a function of time, which is defined as $cwnd_i(t)$. Stability and fairness between connections are investigated by comparing $cwnd_1(t)$ and $cwnd_2(t)$.

3. Congestion control mechanisms of TCP

In this paper, we consider three versions of TCP: Tahoe, Reno, and Vegas versions. TCP Tahoe and Reno are widely used in the current Internet. TCP Vegas is a recently proposed one in [1,3,4]. In TCP, the window size for the connection, called a congestion window size (*cwnd*), is changed according to the network congestion indication. In what follows, we focus on the time-dependent behavior of *cwnd*, since the change of the window size has a significant impact on TCP behavior as will be shown later. For this purpose, we first summarize algorithms to update $cwnd(t)$ in three TCP versions in turn. The analysis methods of the transient behavior of $cwnd_i(t)$ (for connections 1 and 2) in the network model depicted in figure 1 will then be developed in the next section.

For reference purposes, we depict figure 2 which shows typical behaviors of the window sizes as a function of time, $cwnd(t)$, observed in four TCP versions. The figure is obtained by computer simulation using the network model depicted in figure 1 except that only one connection (connection 1) is activated. For parameters, we used the link bandwidth $\mu = 20$ Mbps, the propagation delay $\tau_1 = 2$ ms, the buffer size $B = 10$ segments, and $\alpha = 2$, $\beta = 4$, $\delta = 3$ for control parameters of TCP Vegas and enhanced TCP Vegas (see section 3.3).

3.1. TCP Tahoe version

In TCP Tahoe, the window size *cwnd* is cyclically changed as indicated in figure 2(a). *cwnd* continues increasing until segment loss occurs. When it does occur, TCP determines that the network is congested, and throttles *cwnd* down to the size of

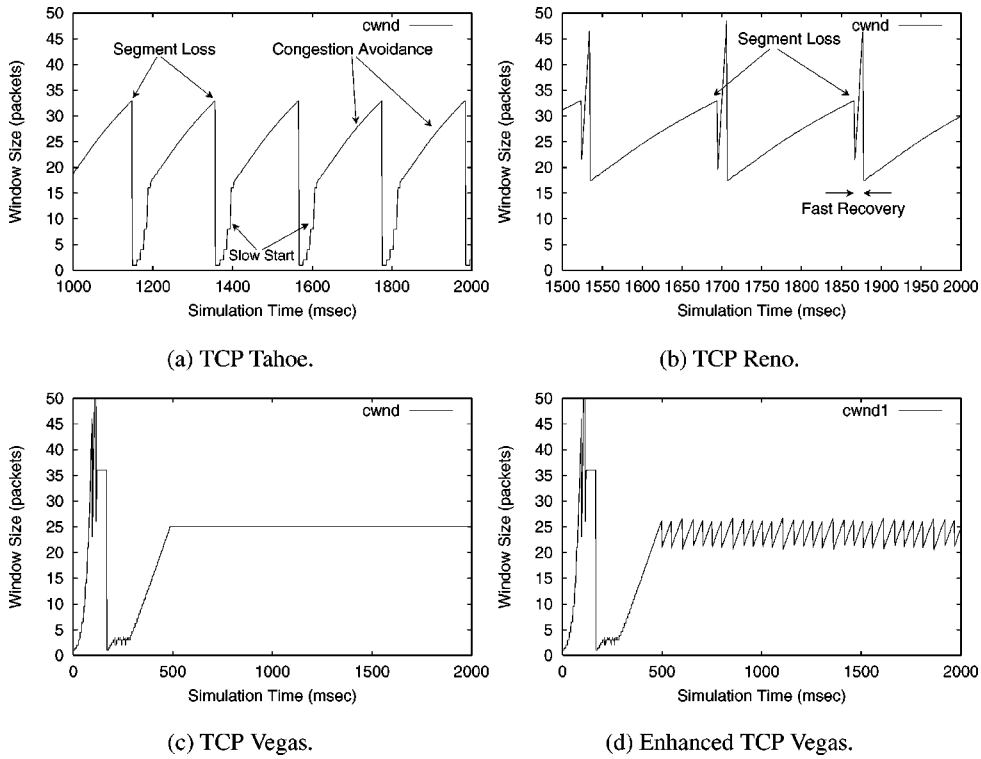


Figure 2. The change of window size of four versions of TCP.

one segment. TCP Tahoe has two phases in increasing *cwnd*; Slow Start Phase and Congestion Avoidance Phase. When an ACK segment is received by TCP at the server side at time $t + t_A$, *cwnd*$(t + t_A)$ is updated from *cwnd*(t) as follows (see, e.g., [9]):

$$\textit{cwnd}(t+t_A) = \begin{cases} \text{Slow Start Phase:} & \\ \textit{cwnd}(t) + m, & \text{if } \textit{cwnd}(t) < \textit{ssth}; \\ \text{Congestion Avoidance Phase:} & \\ \textit{cwnd}(t) + m^2/\textit{cwnd}(t), & \text{if } \textit{cwnd}(t) \geqslant \textit{ssth}; \end{cases} \tag{1}$$

where *ssth* is the threshold value at which TCP changes its phase from Slow Start Phase to Congestion Avoidance Phase. When segment loss is detected by timeout or fast retransmission algorithm [9], *cwnd*(t) and *ssth* are updated as follows:

$$\begin{aligned} \textit{ssth} &= \frac{\textit{cwnd}(t)}{2}, \\ \textit{cwnd}(t) &= m. \end{aligned} \tag{2}$$

That is, TCP Tahoe again enters Slow Start Phase when segment loss occurs. Therefore, the dynamics of TCP Tahoe in a simplest case is; Slow Start Phase $\rightarrow$ Congestion Avoidance Phase $\rightarrow$ segment loss $\rightarrow$ Slow Start Phase $\rightarrow \cdots$.

3.2. TCP Reno version

TCP Reno is similar to TCP Tahoe, but uses another algorithm when segment loss occurs. In Slow Start Phase and Congestion Avoidance Phase, TCP Reno also uses eq. (1) to update the window size, but when segment loss is detected by fast retransmission algorithm, the window size $cwnd(t)$ is halved. That is,

$$ssth = \frac{cwnd(t)}{2},$$
$$cwnd(t) = ssth.$$

TCP Reno then enters Fast Recovery Phase [9]. In this phase, the window size is increased by one segment when a duplicate ACK segment is received, and $cwnd(t)$ is restored to *ssth* when the non-duplicate ACK segment corresponding to the retransmitted segment is received. Figure 2(b) is a typical example of the behavior of $cwnd(t)$.

3.3. TCP Vegas version

In TCP Tahoe and Reno, the window size, *cwnd*, is increased until segment loss occurs due to congestion. Then, the window size is throttled, which leads to the throughput degradation of the connection. However, it cannot be avoided because of an essential nature of the congestion control mechanism adopted in TCP Tahoe and Reno. It can detect network congestion information only by segment loss. However, it becomes a problem since the segment may be lost when the TCP connection itself causes the congestion because of its too large window size. If *cwnd* is appropriately controlled such that the segment loss does not occur in the network, the throughput degradation due to throttled window can be avoided. This is the reason that TCP Vegas was introduced.

TCP Vegas employs another mechanism for detecting the network congestion. It controls *cwnd* by observing changes of RTTs (Round Trip Time) of segments that the connection has sent before. If observed RTTs become large, TCP Vegas recognizes that the network begins to be congested, and throttles *cwnd* down. If RTTs become small, on the other hand, TCP Vegas determines that the network is relieved from the congestion, and increases *cwnd*. Then, *cwnd* in an ideal situation becomes converged to the appropriate value as shown in figure 2(c), and the throughput is not degraded. In Congestion Avoidance Phase, the window size is updated as

$$cwnd(t + t_A) = \begin{cases} cwnd(t) + 1, & \text{if } diff < \dfrac{\alpha}{base\_rtt}, \\ cwnd(t), & \text{if } \dfrac{\alpha}{base\_rtt} \leqslant diff \leqslant \dfrac{\beta}{base\_rtt}, \\ cwnd(t) - 1, & \text{if } \dfrac{\beta}{base\_rtt} < diff, \end{cases} \tag{3}$$
$$diff = \frac{cwnd(t)}{base\_rtt} - \frac{cwnd(t)}{rtt},$$

where *rtt* is a observed round trip time, *base_rtt* is the smallest value of observed RTTs, and α and β are some constant values.

TCP Vegas has another feature in its congestion control algorithm. That is *slow* Slow Start. The rate of increasing *cwnd* in slow start phase is a half of that in TCP Tahoe and TCP Reno

$$cwnd(t + t_A) = cwnd(t) + \frac{m}{2}. \tag{4}$$

Note that eq. (3) used in TCP Vegas indicates that if RTTs of the segments are stable, the window size remains unchanged. That can be seen by figure 2(c), where the window size is converged to a fixed value in steady state. However, when two or more connections share the bottleneck link, the window sizes are not converged to an identical value as will be shown in section 4.4. Then, we will present an enhanced version of TCP Vegas to *prevent* the convergence of the window size to a fixed value in section 4.5.

4. Analysis

In this section, we analytically investigate the congestion control mechanisms of TCP in terms of stability and fairness between two connections. We mainly focus on changes of $cwnd_1(t)$ and $cwnd_2(t)$, the time-dependent behavior of the window sizes of connections.

4.1. Analysis method

To investigate fairness between two connections, we employ the $cwnd_1$–$cwnd_2$ graph depicted in figure 3 [5]. In this graph, x-axis and y-axis represent the window sizes of connections 1 and 2, respectively. The point ($cwnd_1(t)$, $cwnd_2(t)$) represents the status observed at time t. The line labeled with "Fairness Line" corresponds to the case of $cwnd_1 = cwnd_2$, i.e., the window sizes of both connections are equivalent if

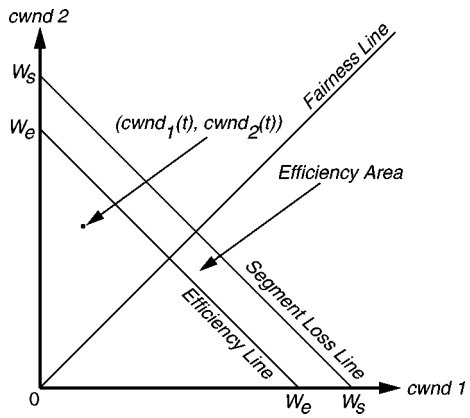


Figure 3. $cwnd_1$–$cwnd_2$ graph.

the point is on the line. By the "Efficiency Line", it is shown whether the link is fully utilized or not. All segments from both connections are served at the bottleneck link with the bandwidth of μ. Thus, if the link is fully utilized at time t, μ equals the sum of the rates at which the segments of both connections are served. By approximately representing the arrival rate of segments by $cwnd_1(t)/\tau_1$ and $cwnd_2(t)/\tau_2$, we have a relation

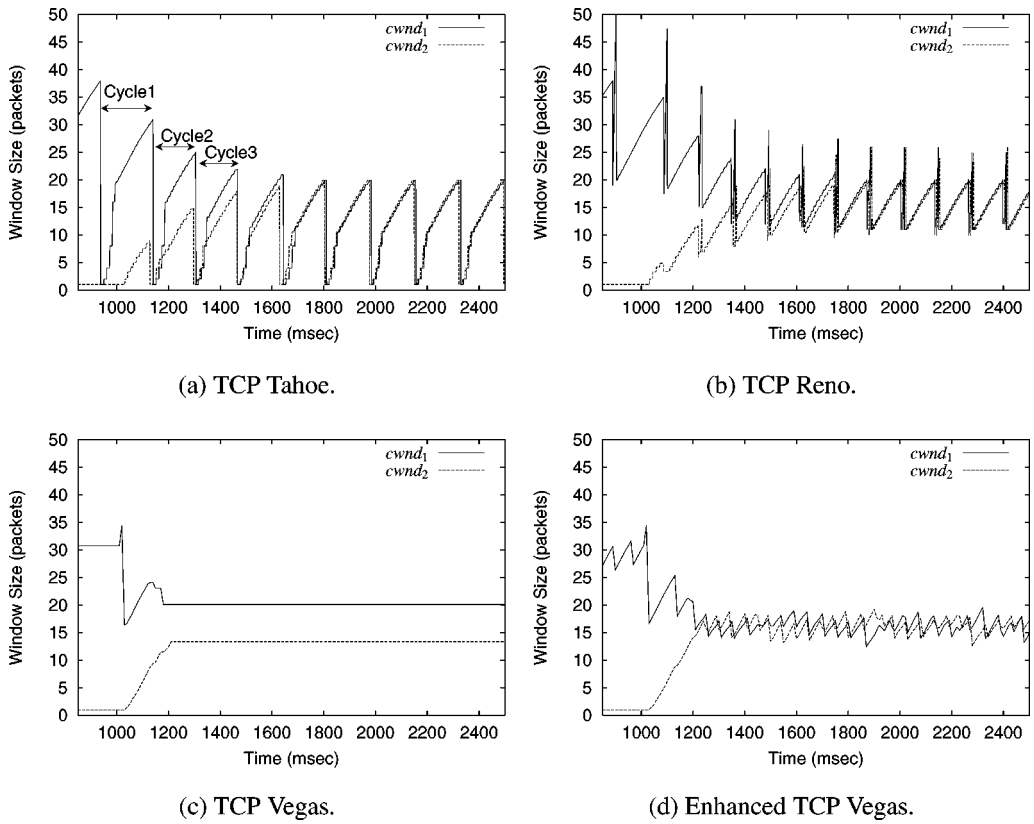
$$\mu = \frac{cwnd_1(t)}{\tau_1} + \frac{cwnd_2(t)}{\tau_2}. \tag{5}$$

We further introduce W_e as

$$W_e = cwnd_1(t) + cwnd_2(t) \tag{6}$$

such that the values of $cwnd_1(t)$ and $cwnd_2(t)$ satisfy eq. (5). Then, the Efficiency Line corresponds to W_e, which means that if the point is located lower than the Efficiency Line, the link bandwidth is not fully utilized. In the homogeneous case ($\tau_1 = \tau_2 = \tau$), eq. (6) after substituting eq. (5) becomes

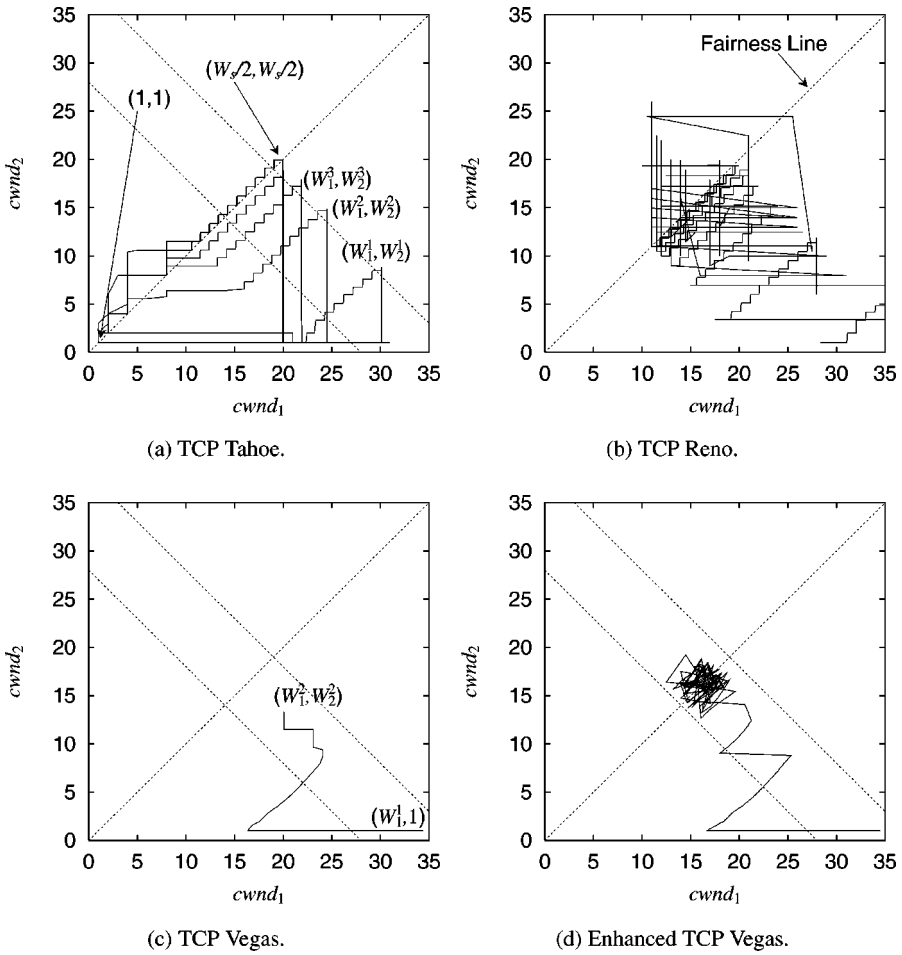
$$W_e = 2\tau\mu.$$



(a) TCP Tahoe. (b) TCP Reno.

(c) TCP Vegas. (d) Enhanced TCP Vegas.

Figure 4. Simulation results of homogeneous case.



(a) TCP Tahoe. (b) TCP Reno.

(c) TCP Vegas. (d) Enhanced TCP Vegas.

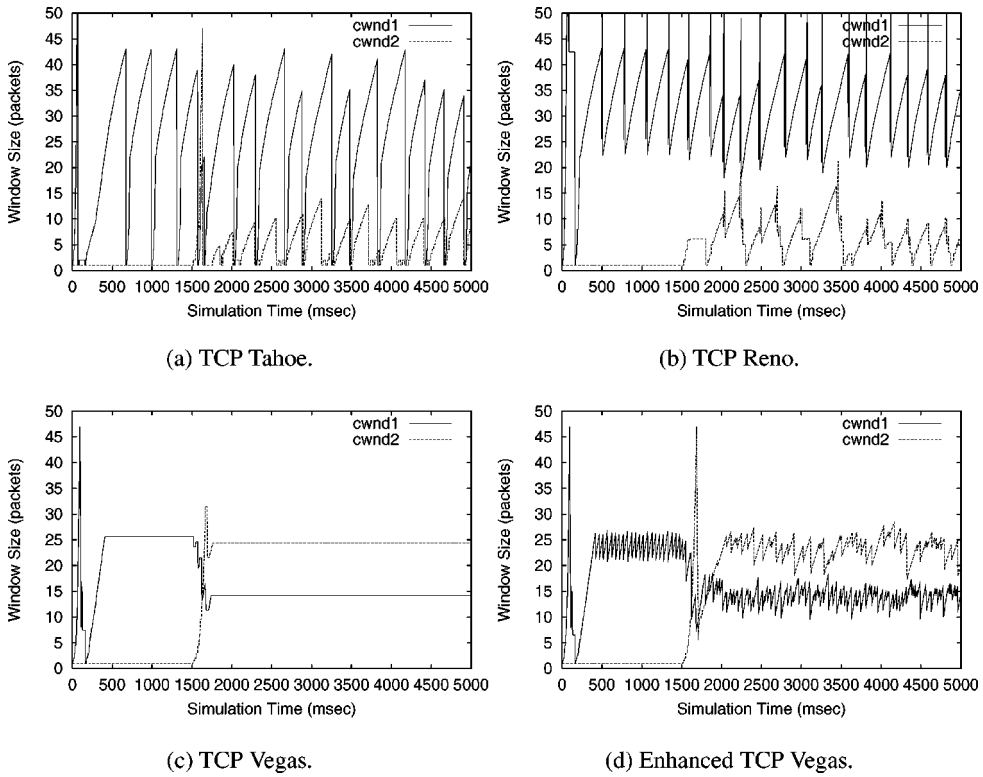
Figure 5. $cwnd_1$–$cwnd_2$ graph of homogeneous case.

The Segment Loss line in the figure represents $cwnd_1 + cwnd_2 = W_s$ [segments], where W_s is the sum of W_e and the buffer size of the intermediate bottleneck switch:

$$W_s = W_e + B.$$

Thus, segment loss occurs if the point is beyond the "Segment Loss Line". If the points are located between "Efficiency Line" and "Segment Loss Line", it can be said that TCP offers an ideal control mechanism in the sense that the network bandwidth is fully utilized and no segment loss occurs. When the fairness is also important, the points should be kept around the "Fairness Line".

Before presenting the analytic results, we illustrate simulation results in figure 4 to give some feeling on the behavior of TCP. We will use it as an illustrative example for deriving analytic results. Note that discussions on the results will also presented in the following subsections. In the figure, the changes of the window sizes of two



(a) TCP Tahoe. (b) TCP Reno.

(c) TCP Vegas. (d) Enhanced TCP Vegas.

Figure 6. Simulation results of heterogeneous case.

connections as a function of time are shown for the homogeneous case where two connections have same propagation delays. In simulation, connection 2 joins the network at time $t = 1000$ [ms]. We set $\mu = 20$ [Mbps], $\tau_1 = \tau_2 = 5$ [ms], $B = 10$ [segments] and $m = 1$ [Kbytes] for parameters of the model shown in figure 1. The other parameters are set as $\alpha = 2$, $\beta = 4$ for TCP Vegas, and $\delta = 3$ for enhanced TCP Vegas. Figure 5 shows the $cwnd_1$–$cwnd_2$ graph obtained from figure 4.

In figures 6 and 7, we show the heterogeneous case where the propagation delays of the two connections are different. In simulation, we set $\mu = 20$ [Mbps], $B = 10$ [segments], $\tau_1 = 4$ [ms], $\tau_2 = 8$ [ms], $m = 1$ [Kbytes], and $\alpha = 2$, $\beta = 4$, $\delta = 3$ for parameters of TCP Vegas and enhanced TCP Vegas, and connection 2 joins the network at time $t = 1500$ [ms]. We will explain the effect of propagation delay on the congestion control mechanisms of TCP by using figures 6 and 7 and our analytical results.

4.2. TCP Tahoe

In TCP Tahoe, the change of the window size is cyclic as shown in figure 2(a) where the single connection utilizes the link. It is also true when two connections with identical propagation delays share the link (figure 4(a)) since segments from two

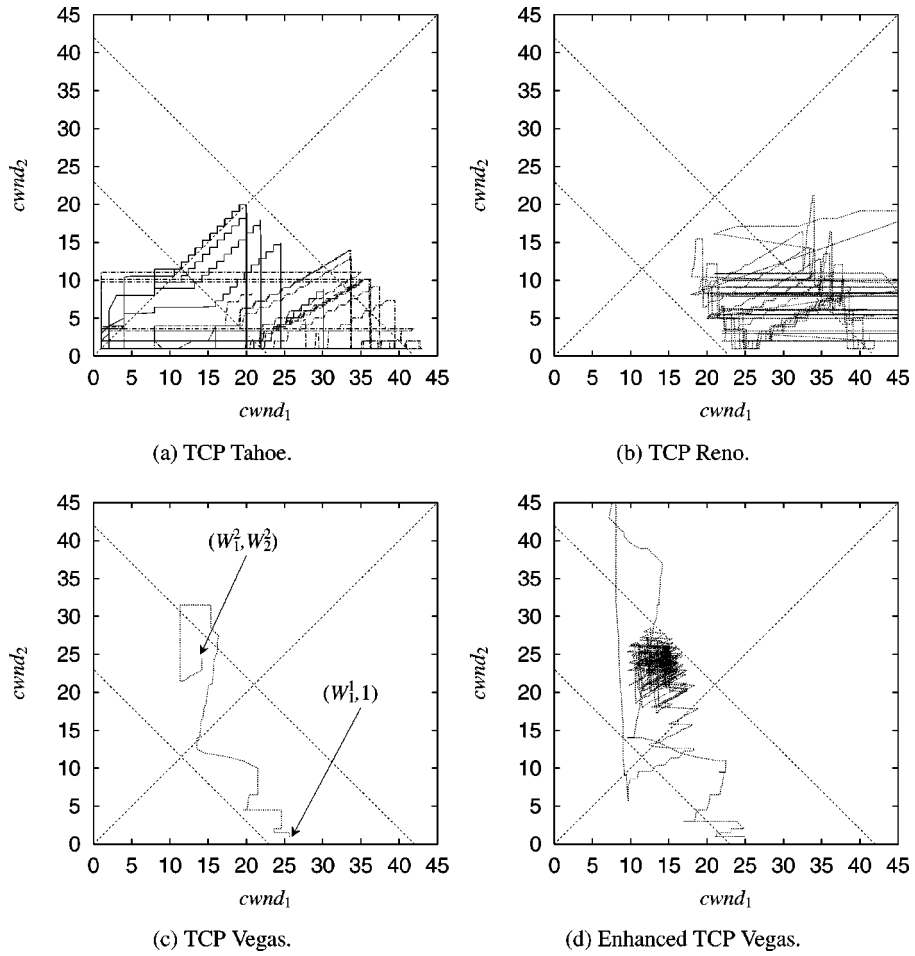


Figure 7. $cwnd_1$–$cwnd_2$ graph of heterogeneous case.

connections are lost at the end of the cycle. It is explained as follows. Suppose that both of two TCP senders open the window at same speed in Congestion Avoidance Phase. Each connection increments its window size by one segment simultaneously, and injects a new segment into the network. Finally, the sum of the window sizes of both connections becomes equal to W_s, the sum of *bandwidth-delay products* of the link (W_e) and the buffer size at switch (B). Then new segments from both connections are likely to be dropped at the switch buffer because the sum of the window sizes exceeds the network capacity by two segments. It is true that we treat a special case for the network configuration, but the problem described in the above is inherent in TCP Tahoe.

When propagation delays of two connections are different, on the other hand, the above discussions never be directly applicable. However, we can confirm that even if two connections have different propagation delays, the segment losses of both

connections are likely to occur simultaneously in figure 6(a). Therefore, in the analysis, we will assume that segment losses of the two connections take place simultaneously.

We introduce the following notations. Cycle i starts at the time when $(i-1)$th segment is lost, and terminates at ith segment loss. W_1^i and W_2^i are window sizes of connections 1 and 2 when ith segment of two connections are lost. Similarly, $ssth_1^i$ and $ssth_2^i$ are defined as *ssth* of cycle i for two connections, respectively. Let us assume that cycle i begins at time $t = 0$. From eq. (2), we obtain

$$ssth_j^i = \frac{W_j^{i-1}}{2}, \quad j = 1, 2. \tag{7}$$

When segment loss occurs, the window size is reset to one segment (since fast retransmit is not used in TCP Tahoe). Then, the window size increases according to Slow Start Phase until $cwnd_j(t)$ reaches $ssth_j^i$. Afterwards, the window size increases according to Congestion Avoidance Phase as follows (see eq. (1)):

$$cwnd_j(t) = \frac{(t - {}_{ssth}t_j^i)}{2\tau_j} + ssth_j^i, \quad j = 1, 2, \tag{8}$$

where ${}_{ssth}t_j^i$ is the time when Slow Start Phase terminates, that is, when $cwnd_j(t)$ reaches $ssth_j^i$. At the end of cycle i, ith segment loss takes place in both connections since the sum of the window sizes of both connections reaches W_s (defined in eq. (7)), i.e.,

$$W_s = cwnd_1\left(t_{loss}^i\right) + cwnd_2\left(t_{loss}^i\right). \tag{9}$$

We can obtain t_{loss}^i, the time when ith segment loss occurs, from eqs. (7) and (8) as follows:

$$t_{loss}^i = \frac{\tau_1\tau_2}{\tau_1 + \tau_2} W_s. \tag{10}$$

Finally, W_j^i is obtained from eqs. (7)–(10) as

$$\begin{aligned} W_j^i &= \frac{W_j^i}{2} + \frac{1}{\tau_j}\frac{\tau_1\tau_2}{2(\tau_1+\tau_2)} W_s \\ &= \frac{1}{\tau_j}\frac{\tau_1\tau_2}{2(\tau_1+\tau_2)} W_s - \left(\frac{1}{2}\right)^{i-1}\left(\frac{1}{\tau_j}\frac{\tau_1\tau_2}{2(\tau_1+\tau_2)} W_s - W_j^1\right). \end{aligned} \tag{11}$$

The above result implies that the window sizes of both connections are exponentially converged as $i \to \infty$, and the converged value is in proportion to the inverse of the propagation delays. It is then clear that if the propagation delays are equivalent, TCP Tahoe provides fair service between connections. We can also see that the congestion control of TCP Tahoe lacks in an ability to stabilize the window sizes in the sense that the window size oscillates as a function of time as shown in figures 2(a), 4(a) and 6(a).

However, it is also observed that in the heterogeneous case of different propagation delays, the window sizes of two connections become different in TCP Tahoe, and

the connection with longer propagation delay suffers from the small window size. This can be observed in figures 6(a) and 7(a), in which connection 2 with longer propagation delay has a small window size during the simulation, and the network status point ($cwnd_1(t)$, $cwnd_2(t)$) is always lower than the "Fairness Line". One may think that the fairness measure should be defined by taking account of the propagation delays, and that it is natural that the connection with the longer propagation delay achieves the less throughput. It may be true, but our point is that in TCP Tahoe, the throughput is not proportional to the propagation delay. We introduce A_j^i as the number of segments transmitted in cycle i of the connection j, and S_j as throughput for the connection j. That is, the following relation holds:

$$A_j^i / t_{\text{loss}}^i. \tag{12}$$

Thus, by utilizing eqs. (7)–(11), we have

$$S_j = \frac{A_j^i}{t_{\text{loss}}^i} = \frac{\int_0^{t_{\text{loss}}^i} cwnd_j(t)\,\mathrm{d}t}{t_{\text{loss}}^i} = \frac{3W_j^i}{4\tau_j}.$$

By letting $i \to \infty$, we have

$$S_j \to \frac{3}{\tau_j^2}\frac{\tau_1\tau_2}{2(\tau_1+\tau_2)}W_{\text{s}}. \tag{13}$$

That is, the throughput becomes proportional to the inverse of the square of the propagation delay in the heterogeneous case.

4.3. TCP Reno

As described in section 3, the congestion control mechanism of TCP Reno is similar to that of TCP Tahoe, except that TCP Reno has a Fast Recovery Phase to be able to react the random segment loss quickly. That is, in the Fast Recovery Phase, the window size is *temporarily* inflated until non-duplicate ACK is received, and it is restored to *ssth*. After that, the Congestion Avoidance Phase begins as in TCP Tahoe. Therefore, if we ignore the temporary inflation of the window size in Fast Recovery Phase, TCP Reno controls the window size as if Slow Start Phase were eliminated from the change of the window size of TCP Tahoe. As a result, the transition of network status point ($cwnd_1(t)$, $cwnd_2(t)$) follows eq. (11). That is, TCP Reno also has an ability to keep a fair service among connections in the homogeneous case, but it cannot keep fair service among connections in heterogeneous case, as in the case of TCP Tahoe. See figures 4(b) and 5(b) for the homogeneous case and figures 6(b) and 7(b) for the heterogeneous case.

4.4. TCP Vegas

In TCP Vegas, it is noticeable that the window sizes of both connections remains constant at different values as shown in figures 4(c) and 6(c). It can also be observed

in the $cwnd_1$–$cwnd_2$ graph in figures 5(c) and 7(c) where the network status point ($cwnd_1(t)$, $cwnd_2(t)$) first moves from $(W_1^1, 1)$ to (W_1^2, W_2^2), and is converged at that point. In this subsection, we analytically derive W_1^1 (the window size of connection 1 when connection 2 is activated), and W_1^2, W_2^2 (converged values of window sizes of connections 1 and 2) to make clear the characteristics of the congestion control mechanism of TCP Vegas.

Let l_1 and l_2 be the mean numbers of segments queued in the switch buffer before and after connection 2 joins the network, respectively. Since the window size in TCP Vegas converges to a fixed value in steady state, l_1 and l_2 should also be converged to some values. We first consider the situation where only connection 1 is active in the network. When the window size of connection 1 becomes stable, the following inequalities should be satisfied from eq. (3):

$$\frac{\alpha}{base\_rtt_1^1} < \frac{W_1^1}{base\_rtt_1^1} - \frac{W_1^1}{rtt_1} < \frac{\beta}{base\_rtt_1^1}, \tag{14}$$

where $base\_rtt_1^1$ is *base_rtt* of connection 1, being equal to the round trip time without queueing delays at the switch buffer. That is,

$$base\_rtt_1^1 = 2\tau + \frac{1}{\mu} \tag{15}$$

and rtt_1 is the round trip time in steady state, i.e.,

$$rtt_1 = 2\tau + \frac{l_1 + 1}{\mu}. \tag{16}$$

From eqs. (15) and (16), eq. (14) can be rewritten as

$$\{2\tau\mu + (l_1 + 1)\}\frac{\alpha}{l_1} < W_1^1 < \{2\tau\mu + (l_1 + 1)\}\frac{\beta}{l_1}. \tag{17}$$

W_1^1 can also be obtained by summing the *bandwidth-delay products* of the shared link ($2\tau\mu$) and the number of segments in the switch buffer (l_1);

$$W_1^1 = 2\tau\mu + l_1. \tag{18}$$

By substituting eq. (18) into eq. (17), we simply have

$$\alpha < l_1 < \beta. \tag{19}$$

Also, eq. (19) can be written by using eq. (18) as

$$2\tau\mu + (\alpha + 1) < W_1^1 < 2\tau\mu + (\beta + 1). \tag{20}$$

From the above equations, we observe that in steady state, the mean number of segments in the switch buffer is kept stable between α and β, and the link bandwidth is always fully utilized.

We next observe the TCP behavior after connection 2 joins the network. When connection 2 starts to transmit segments into the network, the number of segments

queued in the switch buffer increases. Then the round trip time of connection 1 increases, and its window size is decreased to satisfy the condition that the window size should be stable. See eq. (3). Since all segments of both connections are served at the bottleneck link with the bandwidth of μ [segments/s], the following equation is satisfied:

$$\frac{W_1^2}{rtt_1} + \frac{W_2^2}{rtt_2} = \mu. \tag{21}$$

The window size of each connection changes according to eq. (3) as follows:

$$\frac{\alpha}{base\_rtt_1} < \frac{W_1^2}{base\_rtt_1} - \frac{W_1^2}{rtt_1} < \frac{\beta}{base\_rtt_1}, \tag{22}$$

$$\frac{\alpha}{base\_rtt_2} < \frac{W_2^2}{base\_rtt_2} - \frac{W_2^2}{rtt_2} < \frac{\beta}{base\_rtt_2}, \tag{23}$$

where rtt_1, rtt_2, $base\_rtt_1$, $base\_rtt_2$ are Round Trip Time and *base_rtt* of connection 1 and connection 2, respectively, and can be obtained as follows:

$$rtt_1 = 2\tau_1 + \frac{l_2}{\mu}, \tag{24}$$

$$rtt_2 = 2\tau_2 + \frac{l_2}{\mu}, \tag{25}$$

$$base\_rtt_1 = 2\tau_1 + \frac{1}{\mu}, \tag{26}$$

$$base\_rtt_2 = 2\tau_2 + \frac{l_1}{\mu}. \tag{27}$$

By substituting eqs. (24) from (27) into (22) and (23), and after some manipulation, we have

$$(2\tau_1\mu + l_2)\frac{\alpha}{l_2} < W_1^2 < (2\tau_1\mu + l_2)\frac{\beta}{l_2}, \tag{28}$$

$$(2\tau_2\mu + l_2)\frac{\alpha}{l_2 - l_1} < W_2^2 < (2\tau_2\mu + l_2)\frac{\beta}{l_2 - l_1}. \tag{29}$$

Namely, network status point ($cwnd_1(t)$, $cwnd_2(t)$) converges to the values satisfying eqs. (28), (29) and (21). Furthermore, from the condition that network status point ($cwnd_1(t)$, $cwnd_2(t)$) that satisfying eqs. (28), (29) and (21) exists, we can determine the range of l_2, the numbers of segments queued in the switch buffer after connection 2 joins the network. It is obtained by solving eq. (28) and (29) for l_2 as follows:

$$\frac{3+\sqrt{5}}{2}\alpha < l_2 < \frac{3+\sqrt{5}}{2}\beta.$$

We can observe from eqs. (28) and (29) that the window sizes of both connections converge in *almost* proportion to the propagation delay. It means that if the propagation delays of connections are equivalent, the window sizes should become identical. However, it is also observed in eqs. (28) and (29) that W_1^2 and W_2^2 have some *ranges*, and the real convergence point is determined arbitrarily. The range of the convergence is dependent on the congestion control algorithm of TCP Vegas itself, which is the condition that the window size remains unchanged has a some *range* as specified in eq. (3).

There is another reason why TCP Vegas can not achieve fairness between connections. That is caused by the difference of *base_rtt*'s of two connections (eqs. (26) and (27)) even in the homogeneous case with identical propagation delays. When connection 2 joins the network, the switch buffer is occupied by several segments of connection 1. Thus, *base_rtt* of connection 2 includes some buffering delay at the switch and it becomes larger than that of connection 1. Therefore, the window size of connection 2 becomes lower to satisfy the second equation of eq. (3). This cannot be avoided in TCP Vegas if the number of segments at the switch buffer is not much changed in steady state.

The unfairness of TCP Vegas explained above was confirmed by comparing with simulation. The results shown in figures 4(c) and 5(c) is one example, but by repeating the simulation experiments, we observe that the values of W_1^2/W_2^2 range from 1.03 to 1.58. On the other hand, eqs. (28) and (29) show that the upper and lower values of W_1^2/W_2^2 are 0.95–2.12.

On the other hand, the window size in the heterogeneous case becomes almost proportional to the propagation delay of each connection as indicated by eqs. (26) and (27). Thus, the throughput defined as (window size)/(propagation delay) becomes identical, and we may say that the fairness becomes better in the heterogeneous case. However, the obtained throughput has some range dependent on the chosen parameters α and β in TCP Vegas, which causes the unfairness between connections. It can also be confirmed by eqs. (26) and (27).

In summary, TCP Vegas can improve fairness between connections to some extent, but there still be some unfairness due to the *range* of the convergence point. In the next subsection, we will explain our enhanced TCP Vegas, and show some analytic results to confirm the effectiveness of our proposed method.

4.5. Enhanced TCP Vegas

Equation (3) used in TCP Vegas indicates that if RTTs of the segments are stable, the window size remains unchanged. The range that the network is viewed as "stable" was derived in the previous subsection. It is a fundamental problem of TCP Vegas, and our solution is to eliminate the condition of unchanging the window size. The following algorithm is used in our enhanced TCP Vegas to *prevent* the convergence of

the window size:

$$cwnd(t+t_A) = \begin{cases} cwnd(t)+1, & \text{if } diff < \dfrac{\delta}{base\_rtt}, \\ cwnd(t)-1, & \text{if } \dfrac{\delta}{base\_rtt} \leqslant diff, \end{cases} \tag{30}$$

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

where δ is a some small constant value. The same algorithm can be obtained by setting $\alpha = \beta$ in TCP Vegas (eq. (3)), which clearly shows that the condition of unchanging the window size is eliminated. Figure 2(d) shows a typical example of our enhanced Vegas version. We can see from the figure that the window size is oscillated around the appropriate value. By using the algorithm above, we can overcome the unfairness problem observed in TCP Vegas.

We now explain why it can achieve the fairness. The window size of each connection oscillates as a function of time. The point around which the window sizes are oscillated is determined as follows. From eq. (30), we first have

$$\frac{cwnd}{base\_rtt} - \frac{cwnd}{rtt} = \frac{\delta}{base\_rtt}. \tag{31}$$

Let W_1^2 and W_2^2 be the central points of oscillations of the window sizes of connections 1 and 2, respectively. By applying eq. (31) to connection 1 and 2, the following equations can be obtained:

$$\frac{W_1^2}{base\_rtt_1} - \frac{W_2^2}{rtt_1} = \frac{\delta}{base\_rtt_1}, \tag{32}$$

$$\frac{W_2^2}{base\_rtt_2} - \frac{W_2^2}{rtt_2} = \frac{\delta}{base\_rtt_2}, \tag{33}$$

where rtt_1, rtt_2, $base\_rtt_1$ and $base\_rtt_2$ are determined from

$$rtt_1 = 2\tau_1 + \frac{l_2}{\mu}, \tag{34}$$

$$rtt_2 = 2\tau_2 + \frac{l_2}{\mu}, \tag{35}$$

$$base\_rtt_1 = 2\tau_1 + \frac{1}{\mu}, \tag{36}$$

$$base\_rtt_2 = 2\tau_2 + \frac{1}{\mu}. \tag{37}$$

Note that eq. (37) is different from that of TCP Vegas (eq. (27)). It is because our enhanced method simply prevents the convergence of the window size as shown in eq. (30). Then the window sizes of both connections are changed dynamically. It also leads to the fluctuation of the number of segments at the switch buffer, and thus *base_rtt*'s of connections 1 and 2 become converged to the same value.

From eq. (21), eqs. (32) and (33) can be solved as follows:

$$W_1^2 = (2\tau_1\mu + l_2)\frac{\delta}{l_2}, \tag{38}$$

$$W_2^2 = (2\tau_2\mu + l_2)\frac{\delta}{l_2}. \tag{39}$$

Furthermore, in similarly to TCP Vegas, we can obtain l_2 from eqs. (21), (38) and (39)

$$l_2 = 2\delta. \tag{40}$$

We note that l_2 in the above equation is a converged value, and actually the queue size at the switch buffer is fluctuated in some range. However, there is a significant difference between TCP Vegas and enhanced TCP Vegas. In TCP Vegas, converged window sizes of both connections (eqs. (28) and (29)) may be different because it has the *range* in the condition that the window size remains unchanged (eq. (3)). On the other hand, in enhanced TCP Vegas, it is avoided by oscillating the window size.

From eqs. (38) and (39), we can confirm that the central point of oscillation becomes completely proportional to the propagation delay. It means that our enhanced TCP Vegas can provide good fairness even in terms of throughput defined as (window size)/(propagation delay). These results are quite different from those of TCP Vegas, which sometimes fails in obtaining fairness between connections as having been described in section 4.4. Our enhanced TCP Vegas discards the ability of the stable operation which is intended in the original TCP Vegas. However, the oscillation range of the window size is small. The example can be seen in the $cwnd_1$–$cwnd_2$ graph (figures 5(d) and 7(d)). The network status points ($cwnd_1(t)$, $cwnd_2(t)$) oscillate around the "Fairness Line", and the range of oscillation falls between "Effectiveness Line" and "Segment Loss Line". That is, in our enhanced version of TCP Vegas, the throughput can be kept high as in the original one, and the unfairness problem is resolved at the expense of the *stable* operation of the window sizes.

5. Conclusion

In this paper, we have focused on stability and fairness properties of TCP through an analytic approach and have made clear the basic characteristics of four versions of TCP; TCP Tahoe, TCP Reno, TCP Vegas and our proposed enhanced TCP Vegas. We have obtained the following results through analysis and simulation.

- Homogeneous case:
  - ∗ TCP Tahoe and TCP Reno can provide the fairness between connections at the expense of stability and throughput.
  - ∗ TCP Vegas can achieve higher throughput than TCP Tahoe and TCP Reno, but lacks in fair share of the link.
  - ∗ Enhanced TCP Vegas can improve fairness without throughput degradation. It is due to fluctuated window sizes.

- Heterogeneous case:
  - ∗ In TCP Tahoe and TCP Reno, the connection with longer propagation delays suffers from very lower throughput.
  - ∗ In TCP Vegas, fairness between connections can be improved to some extent. However, unfairness is not perfectly resolved in the heterogeneous case.
  - ∗ Our enhanced TCP Vegas can achieve a good fairness between connections while keeping high throughput at the expense of stability.

In the current work, we have only focused on the simple network topology, a single-hop network with two connections. For future work, we need to study the more general network topology, which has multi-hop connection between sender and receiver to investigate the effect of the number of congested links of the connections on the congestion control mechanisms of various versions of TCP. More importantly, we have assumed that TCP connections follow the pre-specified congestion control algorithm. In recent papers such as [6,8], researchers focused on isolation of *ill-behaved* flows emitting segments independently on the congestion level of the network to occupy the link bandwidth unfairly. The proposed scheduling algorithms at the switch can offer the *fair service* according to the pre-determined weights of flows, but have a limit since incorporation of the propagation delays is not considered. We feel that our results can contribute to the extension of the proposed method, but it requires a further research.

References

[1] J.S. Ahn, P.B. Danzig, Z. Liu and L. Yan, Evaluation with TCP Vegas: Emulation and experiment, ACM SIGCOMM Computer Communications Review 25 (August 1995) 185–195.
[2] R. Ahuja, S. Keshav and H. Saran, Design, implementation, and performance of a native mode atm transport layer, in: *Proc. of IEEE INFOCOM '96* (March 1996) pp. 206–214.
[3] L.S. Brakmo, S.W. O'Malley and L.L. Peterson, TCP Vegas: New techniques for congestion detection and avoidance, in: *Proc. of ACM SIGCOMM'94* (October 1994) pp. 24–35.
[4] L.S. Brakmo and L.L. Peterson, TCP Vegas: End to end congestion avoidance on a global Internet, IEEE Journal on Selected Areas in Communications 13 (1995) 1465–1480.
[5] D.-M. Chiu and R. Jain, Analysis of the increase and decrease algorithms for congestion avoidance in computer networks, Computer Networks and ISDN Systems 17 (1989) 1–14.
[6] S. Floyd and V. Jacobson, Random early detection gateways for congestion avoidance, IEEE/ACM Transactions on Networking 1 (1993) 397–413.
[7] J.C. Hoe, Improving the start-up behavior of a congestion control scheme of TCP, ACM SIGCOMM Computer Communications Review 26 (October 1996) 270–280.
[8] M. Shreedhar and G. Varghese, Efficient fair queuing using deficit round robin, IEEE/ACM Transactions on Networking 4 (1996) 375–385.
[9] W.R. Stevens, *TCP/IP Illustrated, Vol. 1: The Protocols* (Addison-Wesley, Reading, MA, 1994).
[10] A.S. Tanenbaum, *Computer Networks*, 3rd ed. (Prentice-Hall, Upper Saddle River, NJ, 1996).
[11] G.R. Wright and W.R. Stevens, *TCP/IP Illustrated, Vol. 2: The Implementation* (Addison-Wesley, Reading, MA, 1995).
[12] XTP home page, `http://www.ca.sandia.gov/xtp/xtp.html`.