Performance Evaluation of HTTP/TCP on Asymmetric Networks

Go Hasegawa, Masayuki Murata, and Hideo Miyahara

Department of Infomatics and Mathematical Science
Graduate School of Engineering Science, Osaka University
1-3, Machikaneyama, Toyonaka, Osaka 560-8531, Japan
Phone: +81-6-850-6588, Fax: +81-6-850-6589
E-mail: {hasegawa, murata, miyahara}@ics.es.osaka-u.ac.jp

Abstract

As the Internet users grow, new network technologies are emerging. Those include ADSL and cable modem, which essentially provide asymmetric bandwidth for uplink and downlink to the user's connection. In this paper, we investigate the behavior of HTTP/TCP protocols on such asymmetric networks, and present the analytic results of the mean throughput of TCP. The transfer time of Web documents by HTTP over TCP is also derived. In the analysis, we consider newer HTTP/TCP protocols, HTTP/1.1 and TCP Vegas, in addition to HTTP/1.0 and TCP Tahoe. We then investigate the appropriate combination of HTTP and TCP protocols on the asymmetric network. The results show that the effect of HTTP/1.1 is quite small, but TCP Vegas can improve the performance in asymmetric networks if it is appropriately modified as in our proposal.

All correspondence should be directed to:

Mr. Go Hasegawa
Department of Information and Computer Sciences
Faculty of Engineering Science, Osaka University
1-3 Machikaneyama, Toyonaka
Osaka 560, Japan
E-mail: hasegawa@ics.es.osaka-u.ac.jp
Phone: +81-6-850-6588
Fax: +81-6-850-6589

1 Introduction

As the Internet users grow, the demand for faster networks becomes larger. To meet those demands, some new network technologies are emerging, which include ADSL (Asymmetric Digital Subscriber Line) [1, 2] and cable modem. Those technologies provide *asymmetric* bandwidth for uplink (from the client to the server) and downlink (from the server to the client). For example, ADSL uses existing telephone lines, and offer $1\sim20$ Mbps for uplink, and $0.1\sim1$ Mbps for downlink. Cable modem service can provide $5\sim50$ Mbps for uplink, and $0.5\sim5$ Mbps for downlink.

Those technologies are considered to be suitable for the Internet access. It is because the user's access to the Internet is essentially asymmetric. The user usually retrieves the information from the Internet through WWW (World Wide Web) service or file transfer service. Both HTTP (Hyper Text Transfer Protocol) for WWW service and FTP (File Transfer Protocol) for file transfer service use TCP (Transmission Control Protocol) [3]; the most popular transport–layer protocol in the Internet. The problem is that TCP has not been designed for asymmetric networks, and the performance of HTTP/FTP over TCP protocols on such networks has not been investigated enough except [4]. In [4], the authors pointed out that the performance of TCP on asymmetric networks is degraded due to the traffic burstiness of the sender. However, in [4], the authors only focus on the mean throughput of TCP Tahoe and Reno [3].

In this paper, we will extensively investigate the performance of TCP on asymmetric networks. Our analytical approach is similar to the one adopted in [4], but in addition to TCP Tahoe [3], we also consider TCP Vegas [5, 6, 7], which adjusts the sending window size by observing the round trip times of the connection. Furthermore, we evaluate the performance of the Web document transfer on such asymmetric networks by analytically treating HTTP over TCP networks. For HTTP, we consider HTTP/1.1 [8] as well as HTTP/1.0. The new HTTP/1.1 can save the condition of the previous TCP connection to avoid the unnecessary connection establishment process when the next transfer request is immediately issued on the same connection. Through the analysis, we will discuss which combination of HTTP and TCP protocols are appropriate in asymmetric networks. We will also point out that the original TCP Vegas is not suitable for the asymmetric networks because it cannot fully utilize the network bandwidth. We thus propose to modify TCP Vegas with minimum change to resolve the problem specific to the asymmetric network. Finally, the design principle suitable to the asymmetric network is discussed.

This paper is organized as follows. In Section 2, we first describe the network model that we will use in the analysis and simulation, followed by introducing HTTP/TCP protocols. We next show our analytic results of HTTP over TCP in Section 3. Numerical results and discussions are then presented in Section 4. Finally, in Section 5, we make some concluding remarks.

2 Model for Asymmetric Networks

In this section, we first describe our asymmetric network model in Subsection 2.1. Then, TCP and HTTP protocols are briefly summarized in Subsections 2.2 and 2.3, respectively.

2.1 Network Model

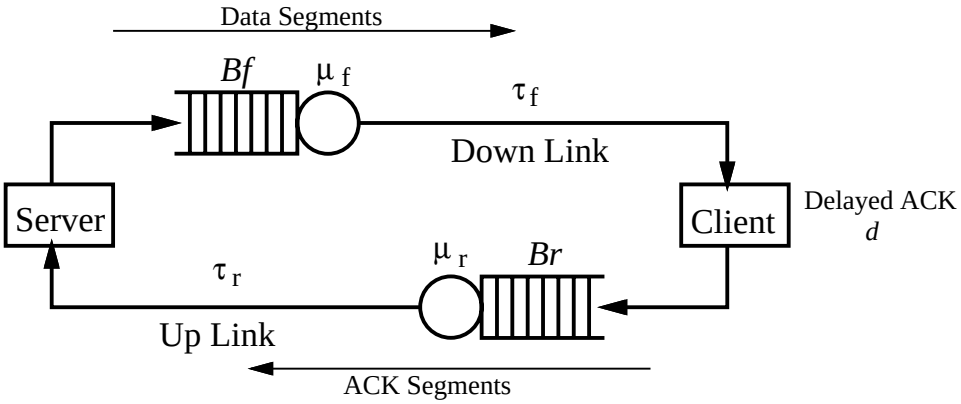


Figure 1: Network Model

The network model which we will use in the analysis and simulation is depicted in Fig. 1. The model consists of a server, a client, and two links; downlink from the server to the client and uplink from the client to the server. The uplink and the downlink have asymmetric bandwidth, denoted as μ_f [data segments/sec] and μ_r [ACK segments/sec], respectively. Note that μ_f and μ_r are represented by units of data/ACK segment, respectively. The buffer sizes are denoted as B_f [data segments] and B_r [ACK segments], and the propagation delays between the server and client are τ_f [sec] and τ_r [sec].

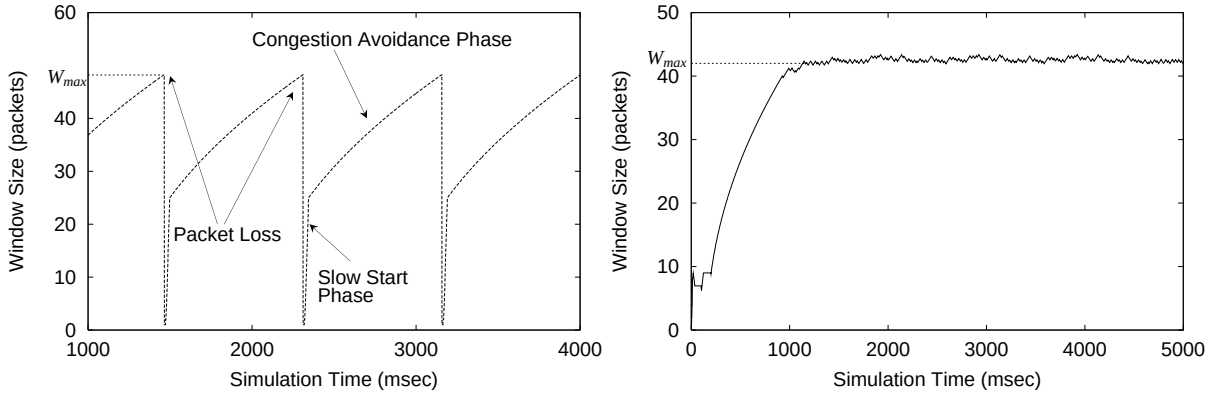
Following [4], we introduce an *asymmetry factor* k defined as μ_f/μ_r to represent the degree of network asymmetry. As an example, if the bandwidth of downlink and uplink are 16 Mbps and 160 Kbps, respectively, and the data segment size and ACK segment size are 1 Kbyte and 40 byte, we then have μ_f = 2000 [data segments/sec], μ_r = 500 [ACK segments/sec]. Then, the asymmetry factor k becomes $\mu_f/\mu_r = 4$. As the asymmetry factor k becomes large, the uplink with smaller bandwidth cannot serve all ACK segments generated by the client, and some of ACK segments are lost at the buffer of uplink. It causes the performance degradation of TCP, which will be discussed in detail in Section 4.

2.2 TCP (Transmission Control Protocol)

We consider two versions of TCP; Tahoe version (*TCP Tahoe*), the most popular one in the current Internet, and Vegas version (*TCP Vegas*) recently proposed in [5, 6, 7]. Here, we will briefly

introduce the congestion control mechanisms of Tahoe and Vegas versions since the congestion control mechanism has a significant effect on TCP throughput as shown later.

In [4], the authors investigated the case of TCP Reno [3] as well as TCP Tahoe. In this paper, we do not consider TCP Reno because the improved mechanism in TCP Reno do not affect the performance in the asymmetric network, and the performance is similar to that of TCP Tahoe as shown in [4].



(a) TCP Tahoe.

(b) TCP Vegas.

Figure 2: The Change of Window Size of two versions of TCP

2.2.1 TCP Tahoe version

TCP employs the window–based flow control mechanism to regulate segment flow injected into the network. The window size for the connection, called congestion window size (*cwnd*), is changed according to the network congestion indication. In TCP Tahoe, *cwnd* continues increasing until segment loss occurs. When segment loss occurs, TCP determines that the network is congested, and throttles *cwnd* down to the size of one segment. As a result, *cwnd* is cyclically changed. A typical example is illustrated in Fig. 2(a). This figure is obtained by the simulation using the network model depicted in Fig. 1.

As shown in Fig. 2(a), TCP Tahoe has two phases in increasing *cwnd*; Slow Start Phase and Congestion Avoidance Phase.

In TCP Tahoe, 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 of TCP Tahoe. It can detect the network congestion information only by segment loss. The problem is that the segment may be lost when the TCP connection itself causes the congestion due to its too large window. If we appropriately control *cwnd* such that the segment loss does

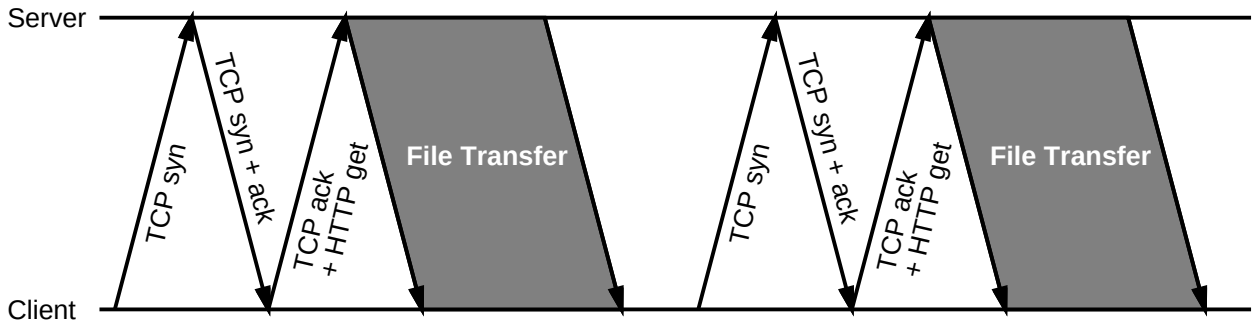
not occur in the network, the throughput degradation due to throttled window size can be avoided. This is the reason that TCP Vegas was introduced.

2.2.2 TCP Vegas version

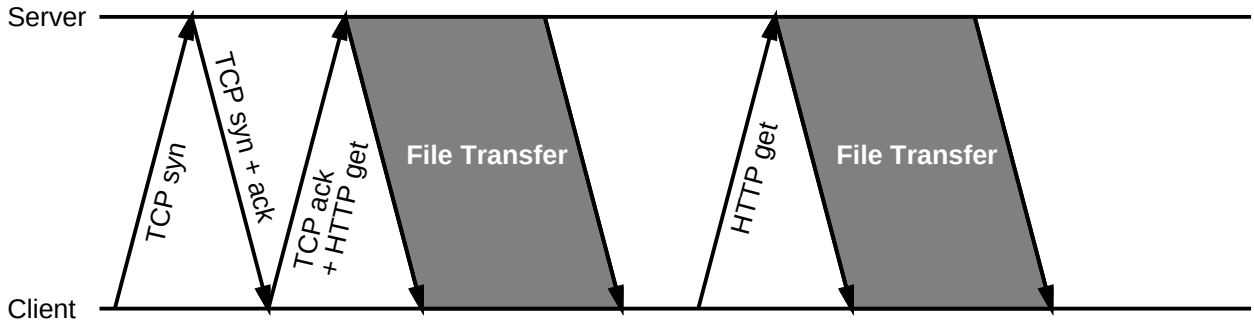
Different from TCP Tahoe, TCP Vegas has another mechanism for detecting the network congestion. It controls *cwnd* by observing changes of RTTs (Round Trip Time) of segments that the connection has sent before. If observed RTT becomes large, TCP Vegas recognizes that the network begins to be congested, and throttles *cwnd* down. If RTTs becomes 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 Fig. 2(b), and the throughput is not degraded.

TCP Vegas has another feature in its congestion control algorithm. It is *slow* Slow Start. The rate of increasing *cwnd* in slow start phase is a half of that of TCP Tahoe. It affects the performance of TCP Vegas especially in the asymmetric network as will be shown in Section 4.

2.3 HTTP (Hyper Text Transfer Protocol)



(a) HTTP/1.0 Transfer



(b) HTTP/1.1 Transfer

Figure 3: Web File Transfer by two versions of HTTP

Figure 3(a) shows a time chart of typical Web document transfer using HTTP/1.0. When the client requests the Web document to the server, a new TCP connection is always established between the client and server. The client first sends a HTTP request command to the server. Then, the server begins to transfer the Web document using TCP. When the document is completely transfered, the TCP connection is immediately closed. It always takes one and a half of RTT (Round Trip Time) before the document transfer is actually started.

In HTTP/1.1, on the other hand, the client saves the information of previously established TCP connections. It can be used when the successive request is destined for the same server. The overhead of document transfer then becomes smaller than that of HTTP/1.0 because another connection establishment phase can be omitted as shown in Fig 3(b).

3 Analysis

In this section, we present the analysis of Web document transfer delay in asymmetric networks. As shown in Fig. 3, there are two phases in the Web document transfer; connection setup phase and document transfer phase. We will consider the analysis of Web document transfer delay by dividing it into these two phases.

3.1 Connection Setup Phase

As has been described in Subsection 2.3, it takes one and a half of RTT (Round Trip Time) for the server to establish a new TCP connection in HTTP/1.0. In HTTP/1.1, it takes the same time as in HTTP/1.0 for the first document request, but the following transfers does not need to establish other TCP connections so that it takes only a half of RTT to start the document transfer (Fig. 3(b)).

Therefore, the connection setup time T_{setup} is,

$$T_{setup} = \begin{cases} \frac{3}{2}rtt, & (\text{a transfer in HTTP}/1.0 \text{ and the first transfers in HTTP}/1.1) \\ \frac{1}{2}rtt, & (\text{the second and later transfer in HTTP}/1.1) \end{cases} \quad (1)$$

where rtt is the round trip time of the connection.

3.2 Document Transfer Phase

In this subsection, we will obtain time dependent behavior of the window size *cwnd*, $cwnd(t)$, and then obtain the mean throughput in each version of TCP.

3.2.1 TCP Tahoe Version

As shown in Fig. 2(a), in Tahoe version, *cwnd(t)* has cycles. We assume that one cycle starts on time $t = 0$. 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;

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

where m is a TCP segment size and $ssth$ is the threshold value at which TCP shifts its phase from Slow Start Phase to Congestion Avoidance Phase. When segment loss is detected, the value of $ssth$ at time t is updated as;

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

From Eq.(2), we obtain

$$\frac{d\,cwnd(t)}{d\,ack(t)} = \begin{cases} m, & \text{if } cwnd(t) < ssth; \\ \dfrac{m^2}{cwnd}, & \text{if } cwnd(t) > ssth; \end{cases} \tag{4}$$

where $ack(t)$ is the accumulated number of ACK segments received by the server from $t = 0$. The rate of receiving ACK segments by the server TCP depends on whether the uplink (from the client to the server) can serve all ACK segments generated by the client TCP. Then, we can derive $\frac{d\,ack(t)}{d\,t}$ as follows;

$$\frac{d\,ack(t)}{d\,t} = \begin{cases} \dfrac{cwnd(t)}{rtt}, & \text{if } \frac{cwnd(t)}{rtt} \leq \mu_r; \\ \mu_r, & \text{if } \frac{cwnd(t)}{rtt} > \mu_r; \end{cases} \tag{5}$$

where rtt is RTT (Round Trip Time) for the case where the downlink buffer is empty, i.e.,

$$rtt = \frac{1}{\mu_f} + \tau_f + \frac{1}{\mu_r} + \tau_r. \tag{6}$$

Finally, $cwnd(t)$ can be obtained from Eqs.(4) and (5) by utilizing the following relationship;

$$\frac{d\,cwnd(t)}{d\,t} = \frac{d\,cwnd(t)}{d\,ack(t)} \frac{d\,ack(t)}{d\,t}. \tag{7}$$

The instantaneous throughput at time t, $\rho(t)$, is then obtained by Eq.(7) as;

$$\rho(t) = \begin{cases} \dfrac{cwnd(t)}{rtt}, & \text{if } \frac{cwnd(t)}{rtt} \leq \mu_f; \\ \mu_f, & \text{if } \frac{cwnd(t)}{rtt} > \mu_f; \end{cases} \tag{8}$$

One cycle of TCP Tahoe terminates when segment loss occurs. Let W_{max} denote the window size at time when segment loss occurs. To obtain W_{max}, we need to consider how the segment loss occurs. There are two reasons;

Case 1: the window size exceeds the bandwidth–delay product of the connection. Here, the buffer sizes at both uplink and downlink are also included.

Case 2: the burst size (the number of segments that the server TCP sends continuously) exceeds the buffer size on the downlink. If the network has some asymmetry (i.e., $k > 1$), the uplink cannot serve all ACK segments since the rate of returning ACK segments exceeds the uplink capacity. It results in losses of ACK segments at the uplink buffer. In TCP, each ACK segment is labeled the largest number of data segment successfully received at the client. Thus, the server receives non–sequential ACK segments if some of ACK segments are lost. It results in that the server continuously emits several data segments according to ACK segments that the server has received, which we call the burst of the data segments. The length of burst (i.e., the number of consecutively transmitted data segments) depends on how many ACK segments are lost at the uplink buffer, and is given k since the uplink can serve only $1/k$ of all ACK segments the client generates.

Thus, two cases in the above are determined by the relation between the size of the downlink buffer, B_f, and the asymmetry factor of the network, k, as follows.

Case 1: The asymmetry factor k is smaller than the downlink buffer size ($k \leq B_f$)

In this case, the burst length (the number of data segments generated by the server) is smaller than the downlink buffer size. Then segment loss occurs when the window size exceeds the bandwidth–delay product of the connection. Therefore, W_{max} is given as

$$\begin{aligned} W_{max} &= \mu_f \left(\tau_f + \frac{1}{\mu_f} \right) + B_f + \frac{\mu_f}{\mu_r} \left[\mu_r \left(\tau_r + \frac{1}{\mu_r} \right) + B_r \right] \\ &= \mu_f T + B_f + k B_r. \end{aligned}$$

Case 2: The asymmetry factor k is larger than the downlink buffer size ($k > B_f$)

In this case, segment loss occurs when the burst size exceeds the buffer size. Once the uplink is fully utilized, the server receives ACK segments with rate of μ_r. The corresponding window size, W_r, is equal to the sum of the uplink capacity (bandwidth–delay product) and

its buffer size, i.e.,

$$W_r \;=\; \mu_r \cdot rtt + B_r$$

Let $w_r = \lfloor W_r/m \rfloor$ be the number of bursts on the connection (including both of uplink and downlink). Further, we introduce b_i to represent the number of segments in the ith burst ($i = 1, \cdots w_r$). Then, the *current window size*, W, is given by $\sum_{i=0}^{w_r} b_i$ in the current case. When the window size reaches W_r, b_i for all i is equal to one because no ACK segment is lost at that time. After that, the length of the burst is incremented in turn as the server receives an ACK segment and the window size is increased by one segment. Since in Congestion Avoidance Phase, the window size is increased at rate of m/rtt, the size of jth burst ($j = (i+W+1) \bmod w_r$) is increased in ith increment of the window size after the window size reaches W_r. Then, the segment loss occurs if the size of some burst exceeds the buffer size of the downlink B_f, and one cycle terminates.

In [4], the authors used an approximate method to calculate W_{max}. However, if we calculate the increment process of the window size repeatedly, we can accurately obtain the value of the window size at time when the maximum length of w_r bursts exceeds B_f. That is,

$$W_{max} \;=\; \sum_i b_i \;\; \text{when} \;\; \max_i(b_i) = B_f + 1. \tag{9}$$

That is, W_{max} can be determined for given values of k and B_f. It is then used to obtain the length of one cycle, T, by solving the following equation;

$$cwnd(T) \;=\; W_{max}.$$

The mean TCP throughput, $\overline{\rho}$, is finally obtained as;

$$\overline{\rho} = \frac{1}{T}\int_0^T \rho(t)\, dt. \tag{10}$$

3.2.2 TCP Vegas Version

In TCP Vegas, evolution of the window size does not have any cycle. See Fig. 2(b). We first consider Case 1' ($k \leq B_f$) which corresponds to Case 1 of TCP Tahoe. In TCP Vegas, after the window size reaches W_{max}, the window size is tried to be converged around W_{max}, and the downlink can be fully utilized. In what follows, we will determine W_{max}.

The convergence of TCP Vegas is achieved by the following algorithm. TCP Vegas observes RTT of each segment, and controls the window size according to RTTs. The window size is kept

unchanged if the following condition is satisfied [5, 6, 7];

$$\frac{m\,\alpha}{base\_rtt} < \frac{cwnd}{base\_rtt} - \frac{cwnd}{rtt} < \frac{m\,\beta}{base\_rtt}, \tag{11}$$

where rtt is the observed round trip time, $base\_rtt$ is the smallest value of observed RTTs, m is the TCP segment size, and α and β are some constant values. By letting q_l be the number of segments queued in the downlink buffer when the window size, $cwnd$, reaches W_{max}, we have rtt, $base\_rtt$ and $cwnd$ as follows;

$$\begin{aligned} rtt &= \tau_f + \tau_r + \frac{B_r}{\mu_r} + \frac{q_l}{\mu_f} \\ base\_rtt &= \tau_f + \tau_r + \frac{B_r}{\mu_r} \\ cwnd &= \mu_f \cdot rtt. \end{aligned}$$

By utilizing the above relations, Eq.(11) can be simplified as;

$$\alpha < q_l < \beta.$$

To simplify the analysis, we assume that q_l equals $(\alpha + \beta)/2$. It is an appropriate simplification since q_l is varied between α and β. Then we obtain W_{max} as follows;

$$W_{max} = \mu_f \left(\tau_f + \frac{\alpha + \beta}{2\mu_f} \right) + \mu_r \left(\tau_r + \frac{B_r}{\mu_r} \right).$$

When the window size reaches W_{max}, the downlink is fully utilized without segment loss. We thus have

$$\rho(t) = \mu_f.$$

We next consider Case 2' ($k > B_f$) corresponding to Case 2 of TCP Tahoe. Since the server emits several data segments continuously as is the case of TCP Tahoe, TCP Vegas cannot also avoid segment losses. Therefore, the window size has cycles like that of TCP Tahoe (Fig. 2(a)). The analysis in this case is almost same as that of TCP Tahoe (Case 2), except that TCP Vegas employs *slow* Slow Start described in Section 2.2.2. That is, in TCP Vegas, $cwnd(t)$ is updated

when it receives an ACK segment as follows;

$$cwnd(t+t_A) = \begin{cases} (\text{Slow Start Phase :}) \\ cwnd(t)+m/2, & \text{if } cwnd(t) < ssth; \\ \\ (\text{Congestion Avoidance Phase :}) \\ cwnd(t)+\dfrac{m^2}{cwnd(t)}. & \text{if } cwnd(t) \geq ssth; \end{cases} \tag{12}$$

3.3 Derivation of Web Document Transfer Delay

We finally obtain the total of Web document transfer delay T_{total} by summing up connection setup time obtained in Section 3.1, and the actual document transfer time, where is obtained from $\rho(t)$ in Section 3.2 by solving the following equation;

$$\int_0^{T_{transfer}} \rho(t)\, dt = S, \tag{13}$$

where S is the size of Web document. Finally, T_{total} can be obtained from Eqs.(1) and (13) as;

$$T_{total} = T_{setup} + T_{transfer}. \tag{14}$$

4 Numerical Examples and Discussion

In this section, we will show some numerical examples based on our analysis presented in Section 3, and discuss on the mean TCP throughput in asymmetric networks and Web document transfer times via HTTP over TCP. For TCP, we use two versions; TCP Tahoe and Vegas. HTTP 1.0 and 1.1 are also considered for HTTP. Then, we will discuss which combination of HTTP and TCP is suitable for asymmetric networks. The parameters displayed in Table 1 are used in this section unless otherwise stated.

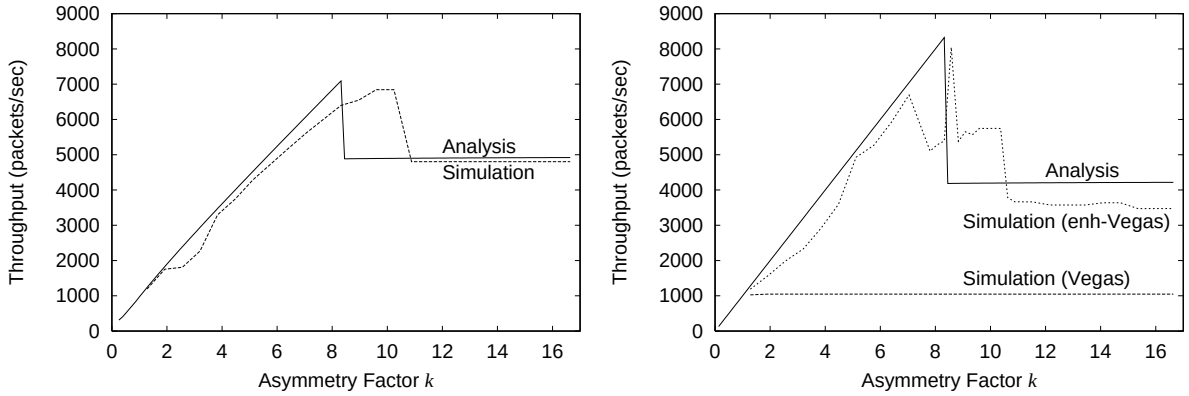
4.1 Comparisons of TCP Mean Throughputs

In this subsection, we only consider TCP level performance. Figure 4 shows the throughput of two versions of TCP as a function of the asymmetry factor of the network, $k = \mu_f/\mu_r$. Here, we change μ_f while fixing μ_r to determine k. Therefore, a larger value of k means larger downlink bandwidth μ_f. In the figure, we also plot simulation results to confirm the accuracy of our analysis presented in the previous section.

Figure 4(a) is the case of the TCP Tahoe version. We can see from this figure that when the asymmetry factor k is small, the throughput increases in proportion to k. Then, it is suddenly

Table 1: Parameter Set Used in Numerical Examples

Variable		Default Value
TCP data segment size	m	1 Kbyte
TCP ACK segment size	m_{ack}	40 byte
Downlink buffer size	B_f	8 segments
Downlink bandwidth	μ_f	4000 segments/sec
Downlink propagation delay	τ_f	1 msec
Uplink buffer size	B_r	5 segments
Uplink bandwidth	μ_r	1000 segments/sec
Uplink propagation delay	τ_r	1 msec



(a) TCP Tahoe case.

(b) TCP Vegas case.

Figure 4: Throughput vs. Asymmetry Factor k

decreased and kept constant when k is large. These two regions correspond to Cases 1 and 2 presented in Section 3.2.1, respectively. That is, if k is smaller than the downlink buffer size B_f, segment loss occurs only when the window size exceeds the sum of bandwidth–delay product of the connection and the uplink and downlink buffer sizes. In other words, the throughput increases in proportion to the downlink bandwidth, μ_f. If k is larger than B_f, on the other hand, segment loss takes place when the size of the burst generated by the server exceeds the downlink buffer size. It is independent of μ_f, and therefore the throughput is kept constant. Our analysis gives good approximation except that the throughput falls down as the asymmetry factor k gets large. The difference is due to the fact that some of bursts generated by the server are divided in two in simulation and W_{max} becomes larger than the analysis result.

We next see the case of TCP Vegas. As can be observed in Fig. 4(b), it is noticeable that the simulation result of TCP Vegas shows quite low throughput (the line labelled as "Simulation (Vegas)"). This can be explained as follows; as described in Section 2.2.2, TCP Vegas controls the window size according to the change of RTTs. Namely, it stops increasing the window size

when RTT gets slightly larger than the smallest RTT value ($base\_rtt$). Since the increase of RTT is caused by queueing at intermediate buffers, ACK segments passing through the uplink is first delayed at the uplink buffer in asymmetric networks. As a result, the window size is converged to the capacity of the uplink, but it is small in the current case. It leads to very low throughput of TCP Vegas.

Of course, TCP Vegas was originally designed not for asymmetric networks, and we made a small change in the algorithm of TCP Vegas to resolve this problem. In the original TCP Vegas, the change of RTTs is found by comparing the observed RTT with a minimum RTT, $base\_rtt$. Our change is to update $base\_rtt$ at a regular interval so that TCP Vegas could set $base\_rtt$ to the value which includes the queuing delay at the uplink buffer. The line labelled by "Simulation (enh-Vegas)" in Fig. 4(b) shows the case of our modified Vegas, where the update interval is set to be 100 msec. The throughput is remarkably improved. Since our analysis assumes that TCP Vegas can fully utilize the capacity of the network, analysis results give a good accuracy when compared with the simulation results. Namely, our enhanced version of TCP Vegas can set the window size to an appropriate value. We will use this enhanced Vegas version in the numerical results shown in the the following subsection.

It is clear by comparing Figs. 4(a) and 4(b) that the mean throughput of TCP Vegas is higher than that of TCP Tahoe for small k, but lower for large k. If $k \leq B_f$, TCP Vegas can appropriately control the window size like Fig. 2(b), and no segment loss occurs. Therefore, TCP Vegas can achieve higher throughput than TCP Tahoe. When k becomes larger than B_f, however, segment loss occurs even in TCP Vegas and the change of the window size has cycles like TCP Tahoe (Fig. 2(b)). It is because even TCP Vegas cannot avoid segment losses caused by the bursty generation of data segments by the server. Furthermore, at the beginning of the cycle, the window size of TCP Vegas is not increased as fast as that of TCP Tahoe according to its *slow* Slow Start discipline as described in Section 2.2.2. Then, the throughput of TCP Vegas becomes lower than even that of TCP Tahoe.

4.2 Web Document Transfer Delay through HTTP over TCP

In this section, we will show analytic results of Web document transfer delay when using HTTP over TCP. We consider four combinations of HTTP over TCP; HTTP/1.0 over TCP Tahoe (*http–tahoe*), HTTP/1.0 over TCP Vegas (*http–vegas*), HTTP/1.1 over TCP Tahoe (*http1.1–tahoe*), and HTTP/1.1 over TCP Vegas (*http1.1–vegas*).

First, we investigate the case of a single Web document transfer. In this case, there is no difference between HTTP/1.0 and HTTP/1.1 in our performance study because the advantage of HTTP/1.1 is to omit the connection setup phase in transmitting two or more documents consecutively. Therefore, only a choice of TCP Tahoe or Vegas affects the performance. Figure 5 shows

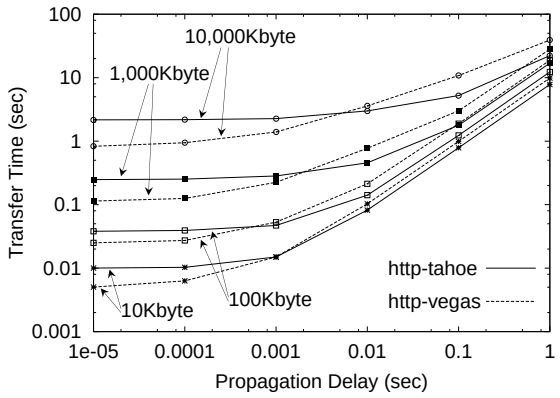


Figure 5: Document Transfer Time vs. Propagation Delay

the Web document transfer delay as a function of the propagation delay between the server and the client. Here, we set $\tau_f = \tau_r$. The asymmetric factor k is fixed at 3.75 (i.e., $\mu_f = 40$ Mbps). In the figure, four cases of the document size are considered; 10 Kbyte, 100 KByte, 1,000 Kbyte and 10,000 Kbyte. A typical value of the document size on the existing Web server is 10 Kbyte, but we can find 1,000 or 10,000 Kbyte documents of audio and movies. We can observe from this figure that when the propagation delay becomes large, the performance of TCP Vegas gets worse than that of TCP Tahoe. Furthermore, as the document size becomes smaller, the performance of TCP Vegas gets worse than that of TCP Tahoe with the smaller propagation delays. It is due to the *slow* Slow Start of TCP Vegas. It is true that TCP Vegas could achieve higher throughput than TCP Tahoe in steady state, but it takes more time to reach the steady state due to its *slow* Slow Start because the increase of TCP window size is triggered by reception of ACK segments. Therefore, when the propagation delay is large, the ill effect of *slow* Slow Start becomes more significant. Because of the same reason, as the document size becomes small, TCP Vegas gives larger transfer delay than TCP Tahoe by the smaller propagation delays. It is because the document transfer tends to be terminated before TCP Vegas reaches the steady state.

Another comparative result of TCP Tahoe and Vegas is shown in Fig. 6 where the document transfer delays are plotted as a function of the asymmetry factor k. As one may expect, when k becomes large, the transfer delay of TCP Vegas becomes worse than that of TCP Tahoe. It is because the throughput of TCP Vegas is degraded by larger k as having been shown in Section 4.1 (see Fig. 4). Since typical values of the asymmetry factor in ADSL networks is 1 to 10, the performance of TCP Vegas is not so high in ADSL networks. Or, if TCP Vegas is used in the asymmetric networks, the downlink buffer size, B_f, should be large enough in order to achieve high performance.

To investigate the effect of HTTP/1.1, we next consider multiple Web document transfer. Here we assume that the multiple documents are consecutively requested by the same client. Figure 7

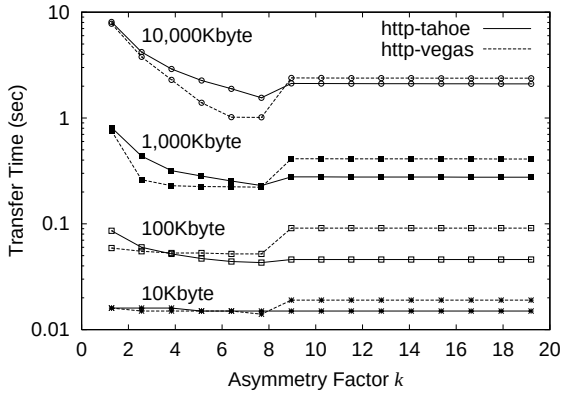
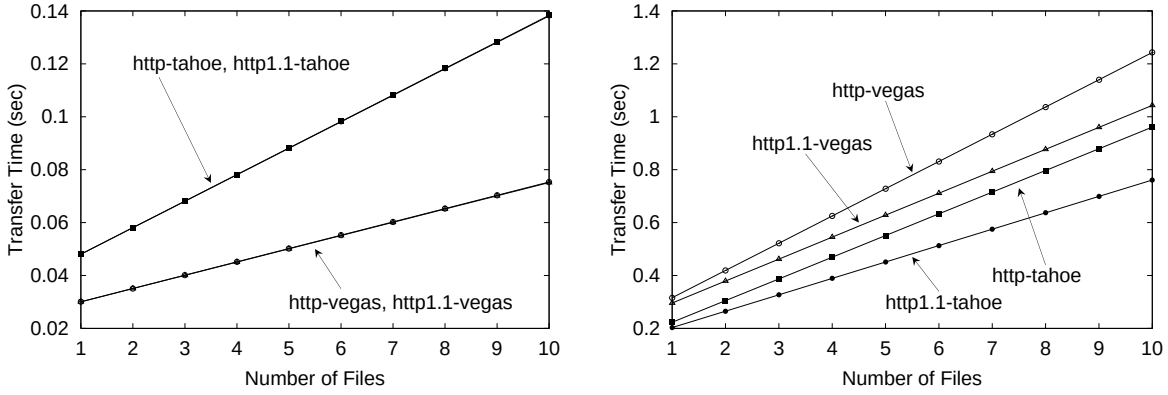


Figure 6: Document Transfer Time vs. Asymmetric Factor k



(a) $\tau_f = \tau_r = 0.01$ msec

(b) $\tau_f = \tau_r = 10$ msec

Figure 7: Throughput vs. Asymmetry factor k

presents the total transfer delay as a function of the number of Web documents. In the figure, four combinations of HTTP and TCP are shown. Figures 7(a) and 7(b) show cases of the small propagation delay ($\tau_f = \tau_r = 0.01$ msec) and the large propagation delay ($\tau_f = \tau_r = 10$ msec), respectively. The asymmetric factor k is set to be 3.75. In obtaining these figures, we assume that the size of the first document is 100 Kbyte while those of following documents are 10 Kbyte. From these figures, we can observe that when the propagation delay is small, the effect of introducing HTTP/1.1 is quite small (Fig. 7(a)). Even if the propagation delay becomes large, the effect of HTTP/1.1 is limited, and the selection of TCP becomes more important (Fig. 7(b)).

We last consider another case for the document size distribution. The probability distribution of the document that we tested is shown in Table 2, which we brought from [9]. Figure 8 shows the mean document transfer delay as a function of the propagation delay between the server and the client. It can be observed from this figure that if the propagation delay is small (100 μsec$\sim$1 msec) such as the case of ADSL networks, it is a good choice to use TCP Vegas. On the other hand,

Table 2: Probability Distribution of Web Documents

Probability	File Size(s)
40%	2 Kbyte,3Kbyte
25%	1 Kbyte,5Kbyte
15%	4 Kbyte,6Kbyte
5%	7 Kbyte
4%	8 Kbyte,9 Kbyte,10 Kbyte,11 Kbyte
4%	12 Kbyte,14 Kbyte,15 Kbyte,17 Kbyte,18 Kbyte
6%	33 Kbyte
1%	200 Kbyte

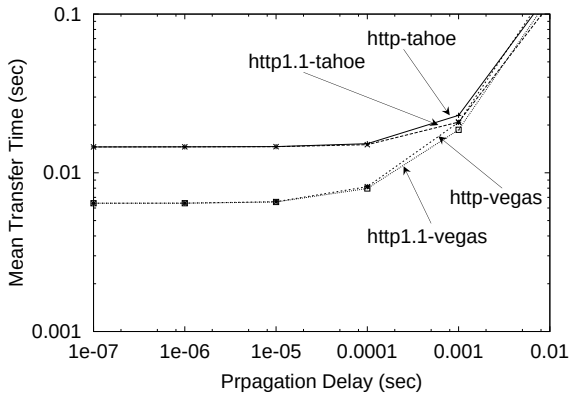


Figure 8: Mean Document Transfer Time vs. Propagation Delay

HTTP/1.1 does not give performance improvement in asymmetric networks.

5 Conclusion

In this paper, we have analytically investigated of the performance of Web document transfer on HTTP over TCP in asymmetric networks. Our conclusions are summarized as follows;

TCP Vegas: TCP Vegas may show very low throughput because it controls its window size according to the smallest bandwidth of the connection, which corresponds the downlink in the asymmetric network. This can be avoided by applying the modification as we have shown. However, the throughput may be degraded in some cases; e.g., the propagation delay is small and/or the size of documents is small. Therefore, TCP Vegas is not necessary in the asymmetric network.

HTTP/1.1: HTTP/1.1 can improve the document transfer delay when the multiple Web files are transferred as it expects. However, the effect of HTTP/1.1 is limited and an appropriate choice of TCP is more important in asymmetric networks.

HTTP and TCP: When we consider the asymmetric network like the ADSL network, it is not mandatory to adopt HTTP/1.1 because its effect is not large. On the other hand, TCP should be chosen carefully because TCP Vegas is sometimes inferior to TCP Tahoe. It depends on the asymmetric factor of the network and the distribution of transfer documents.

As future works, we plan to confirm the observations presented in this paper in the actual asymmetric network using ADSL networks. We also plan to modify the congestion control algorithm of TCP Vegas to achieve higher throughput than TCP Tahoe or TCP Reno in all situations.

References

[1] A. Forum, "ADSL tutrial: Twisted pair access to the information highway," *Available from http://www.adsl.com/adsl_tutrial.html*, 1996.

[2] W. Y. Chen and D. L. Waring, "Applicability of ADSL to support video dial tone in the copper loop," *IEEE Communication Magazine*, pp. 102–109, May 1994.

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

[4] T. V. Lakshman, U. Madhow, and B. Suter, "Window–based error recovery and flow control with a slow acknowledgement channel: a study of TCP/IP performance," *Proceedings of IEEE INFOCOM'97*, pp. 1201–1211, April 1997.

[5] L. S. Brakmo, S. W.O'Malley, and L. L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance," *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. S. Ahn, P. B. Danzig, Z. Liu, and L. Yan, "Evaluation with TCP Vegas: Emulation and experiment," *ACM SIGCOMM Computer Communications Review*, vol. 25, pp. 185–195, August 1995.

[8] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee, "Hypertext transfer protocol – HTTP/1.1," *Request for Comments 2068*, January 1997.

[9] Silicon Graphics Inc., "WebStone: World wide web server benchmarking," *Available from http://www.sgi.com/Products/WebFORCE/Resources/res_webstone.html*, 1996.