Congestion and Fairness Control Mechanisms of TCP for the High-Speed Internet

Go Hasegawa

February 2000

Department of Informatics and Mathematical Science
Graduate School of Engineering Science
Osaka University

Preface

In the Internet, TCP (Transmission Control Protocol) has been widely used as its transport-layer protocol. Many Internet services such as HTTP (Hyper Text Transfer Protocol) for World Wide Web and FTP (File Transfer Protocol) are designed on the basis of TCP. A number of researchers have been extensively working on TCP's congestion control mechanisms which is an essential part to realize the effective network utilization. It has been known that the current TCP does not scale to high-speed networks directly, so that researches on TCP mechanism for high-speed networks is now one of hot topics. However, one problem of those researches is that neither stability nor fairness of TCP is fully considered; that is, most studies on TCP mechanism focus only on its effective throughput. Therefore, it is strongly needed to investigate TCP's essential characteristics in terms of fairness and stability as well as throughput. It is also an important issue how TCP can be applied to very high-speed networks, and how TCP can adapt to new Internet services, such as ADSL (Asymmetric Digital Subscriber Line) and cable modem networks.

In this thesis, we first analyze the behavior of the TCP mechanism for better understanding of its dynamical behavior. The primary objective of this part is to investigate stability and fairness of various versions of TCP mechanisms: TCP Tahoe, TCP Reno and TCP Vegas. We analyze behavior of two TCP connections sharing a same bottleneck link, and evaluate system stability and fairness between these connections. Analytic results show that TCP Tahoe/Reno can achieve good fairness among connections, but they cannot to keep stability of their window sizes because of the essential nature of their algorithms. TCP Vegas, on the other hand, can achieve good stability, but its fairness is sometimes degraded. We propose a simple modification on the congestion avoidance mechanism of TCP Vegas for achieving better fairness and stability, and validate its effectiveness through simulation experiments. The simulation results indicate that our proposed mechanism achieves better stability than TCP Tahoe and Reno, and also achieves better fairness than TCP Vegas.

We next focus on a problem in the packet retransmission mechanism of TCP. The problem occurs when a fixed amount of bandwidth is assigned to a TCP connection, or when an RTT (Round Trip Times) of a TCP connection changes frequently. In this case,

TCP frequently performs unnecessary packet retransmission (we call *mis-retransmission* in this thesis), which causes serious performance degradation. We analytically show when and why packet mis-retransmission occurs. Based on our analytic result, we propose a mechanism to detect mis-retransmission by observing an RTT for a retransmitted packet. Although our proposed mechanism cannot prevent mis-retransmission, performance degradation can be avoided by restoring the window size throttled by mis-retransmission. Effectiveness of our proposed mechanism is validated by simulation experiment and by implementation in real networks.

Based on the above discussion, we investigate on applicabilities of the TCP mechanism to various high-speed networks. First, an ATM (Asynchronous Transfer Mode) network is considered as a lower-layer protocol, and interference between two congestion control mechanisms of TCP and ATM is investigated. Through several simulation experiments, we show that UBR (Unspecified Bit Rate) service class is ill-matched to TCP because many cell losses at the ATM layer are inevitable so that the total performance is significantly degraded. We also show that ABR (Available Bit Rate) service class is well-suited to TCP and it can achieve high performance in terms of throughput and fairness among connections. However, it should be noted that such high performance cannot be achieved without appropriate tuning of ABR's control parameters. Second, an asymmetric network is considered, where bandwidths of the upstream link and the downstream link are different. Such network configurations can be found in ADSL networks and CATV networks. The problem is that if the network is asymmetry, i.e., if the difference in bandwidths of the upstream and downstream links is large, TCP acknowledgement packets are lost at the upstream link. That causes TCP sender to send some TCP packets in a burst, which results in packet losses at downstream links. We analyze the performance of TCP mechanisms in an asymmetric network, and show that as network asymmetry becomes large, performance degradation becomes large. To prevent performance degradation, we introduce the appropriate setting of network configuration parameters such as router buffer sizes of both of upstream and downstream links.

In the above researches, we have focused on the TCP mechanisms itself. We finally investigate the effect of packet scheduling algorithms at the router buffer on *fairness* among TCP connections. In more detail, we focus on the degree of fairness provided to TCP connections. For comparison, three packet scheduling algorithms at the router are considered: FIFO (First In First Out, or Drop-Tail), RED (Random Early Detection), and DRR (Deficit Round Robin). FIFO is a traditional discipline which is deployed in most current Internet routers. RED implements probabilistic packet dropping at the FIFO buffer for better fairness and throughput. DRR is one of the derivations of WRR (Weighted Round Robin) algorithms, which employs a per-flow queueing for every con-

nection. Our concern is on *proportional fairness* among connections, where each connection should receive *proportional* throughput to the bandwidth of its input link at the router. We first show simulation results that FIFO cannot provide fairness among connections because of bursty nature of packet losses. It is next shown that RED offers better fairness than FIFO to TCP Reno connections, but it cannot keep a good fairness when the output link capacity becomes small compared to the total input link capacity. DRR can provide almost perfect fairness among connections, unless multiple TCP connections are assigned to a same Round Robin queue. We also propose DRR+ mechanisms, where RED is used at each DRR queue, and show that our DRR+ can provide better fairness than DRR through the simulation results.

TCP was first introduced in early 1970s, and has been gradually improved for traditional low-speed networks. Through all of our researches, we have found that the main problem of the current TCP is that congestion control mechanisms of TCP Tahoe and TCP Reno have not been designed to support either very high-speed networks or new-emerging Internet services. TCP Vegas is a possible solution for the future Internet, but some modifications proposed in this thesis are necessary for better performance in terms of throughput, fairness and stability. Another conclusion of this thesis is that a packet scheduling algorithm at the router's buffer must be enhanced for supporting various fairness definition such as proportional fairness. The Internet is going to change continuously now and in the future. Protocol migration is a key issue for the Internet development. Throughout this thesis, our proposals for performance improvement of TCP do not pose an introduction of a new protocol, but minor modification to the existing TCP. We believe that TCP can be seamlessly migrated to the future Internet by applying our research results.

Acknowledgements

This work has its root in the teaching, help, and inspiration of a great number of people, to whom I wish to express my gratitude.

Simply put, Prof. Masayuki Murata, my advisor, is the reason I have studied this work; the reason I embarked on this study. Heartfelt thanks go to him who has given me so much energy for research study and meaningful advice on nearly every pages of this work.

I would like to express my gratitude to Prof. Hideo Miyahara for his countless advice and continuous support. I am heartily grateful to Prof. Tohru Kikuno and Prof. Kenichi Hagihara for serving as readers of my thesis committee.

This work would not have been realized without discussions with Prof. Shinji Shimojo, Prof. Masashi Sugano, Prof. Ken-ichi Baba. There much appreciated ideas, support, and feedback have been great help for my study.

I would likewise thank my colleagues and friends in the department, for their detailed, valuable instructions, fellowship and underpinning. I am particularly thank Assistant Prof. Naoki Wakamiya, Dr. Hiroyuki Ohsaki, Mr. Shingo Ata and Mr. Kentarou Fukuda, for their expert suggestions and helpful comments.

I dedicate this thesis to my parents who have continuously loved and supported me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Backgrounds

TCP (Transmission Control Protocol) [1] is widely used in the current Internet, and many of popular Internet services, including HTTP (and World Wide Web) and FTP (File Transfer Protocol), use it as the de-facto standard transport-layer protocol. Thus, even if the network infrastructure may change in the future, TCP and its applications would be likely to be continuously used. Off course, TCP is very traditional protocol which was first designed in early 1970s, and many efforts of researchers, developments and standardization have been extensively devoted to the TCP/IP technology. The authors in [2] pointed out the importance of the congestion control in the Internet, and proposed some algorithms of TCP to avoid and control congestion in the network. The paper has brought many researchers to become aware of importance of TCP's congestion control, and an extensive literature has accumulated on it [3-7]. For example, Paxson [7] investigated end-to-end Internet dynamics including the behavior of TCP's congestion control mechanisms, and characterized it. As the result of these efforts, many RFC (Request For Comments) documents regarding TCP are announced to enhance its performance [8-10].

On the other hand, explosive increase of the Internet population has made the Internet larger and larger, and the network infrastructure which constructs the Internet has been rapidly improved in all its aspects. In the access network from each Internet user to ISP (Internet Service Provider), the new technologies, such as ISDN (Integrated Services Digital Network), ADSL (Asymmetric Digital Subscriber Line) or CATV Internet service, have emerged, replacing the traditional analog modem access using telephone line. These technologies can provide *larger* bandwidth (128 Kbps $\sim$ 1Mbps or more), or may sometimes provide *asymmetric* bandwidth to each user, which is very different from a traditional analog modem network. When considering a backbone network,

ATM (Asynchronous Transfer Mode) [11, 12] technology has been already introduced, and other new networking technologies are also emerging including photonic networks such as WDM (Wavelength Division Multiplexing) networks, to introduce much larger bandwidth (Gbps $\sim$ Tbps). Therefore, existing research results about TCP may not be directly applied to future high-speed Internet which is developed by such networking technologies [5, 13]. Although some modifications of TCP have been already proposed (for examples, [10, 14-16]), but most of them have focused only on TCP's overall throughput and have not paid so much attention to the other performance measures, such as stability, fairness among connections, and so on.

Emergence of those new networking technologies makes it enable to realize new Internet services. The conventional Internet has only been providing the *best effort service*, and it could not offer throughput and/or delay guarantees. It also lacks of fairness guarantees; TCP connections sometimes receive unfair service in terms of, e.g., throughput. For supporting commercial network services in the Internet, however, we now need more advanced services, different from best effort service. That is, a new service should be available within the network to support the *differentiated services* among the users [17]. Following the *diff-serv* model, several kinds of network service have recently been proposed; for example, a constant throughput may be preferred to some connections, or QoS (Quality of Service) support is necessary for real-time applications. For example, in [18], the authors have proposed an *Explicit Capacity* framework for allocating the network capacity to users in a controlled way even in congestion periods. In that approach, the network defines service profiles in advance, and incoming packets are tagged when entering the network. The tagged packets are dropped first when congestion occurs in the network.

Another important service that the next-generation Internet should support is *fair allocation of the bandwidth.* It is one of most desired features for elastic applications, but not supported in the current Internet, and we believe that fairness may be as important as network efficiency. One service found in the literature is the USD (User Share Differentiation) scheme [19], where users are provided different service qualities from ISPs based on the contracts. However, the authors in [19] do not provide a quantitative evaluation of USD to show how the users are *differentiated.*

One convincing way to realize such service differentiation is to be per-flow queueing at the router in the network, and some algorithms have been proposed in the literature such as CBQ (Class Based Queueing)[20], WRR (Weighted Round Robin) [21], and DRR (Deficit Round Robin) [22] that is one of the derivations of WRR. However, the effectiveness of these algorithms have been confirmed under Poisson arrivals of packets from each connection, and the behavior of the upper-layer protocol, i.e., TCP, has not been considered to evaluate their performance. On the other hand, some new transport-

layer protocols have been developed for a new underlying network. Examples are XTP (Xpress Transport Protocol) [23] and the one in [24] for ATM networks. However, TCP has been already widely used in the current Internet, and many of the current Internet services rely on TCP. Therefore, it is unrealistic proposal to replace TCP, with a new transport-layer protocol immediately. Therefore, for gradual migration of TCP to the next-generation Internet, we believe that it is necessary to modify TCP's congestion control mechanisms to support various types of services in the high-speed Internet.

1.2 Congestion Control Mechanisms of TCP

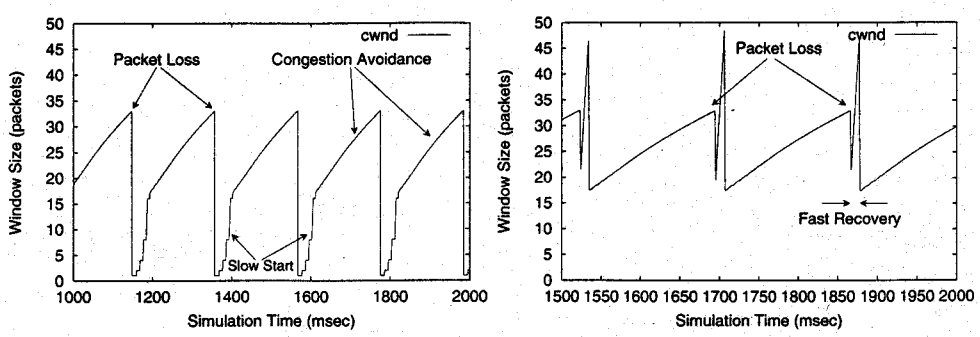
Most of the current TCP implementations are based on TCP Tahoe and TCP Reno, which was first implemented in BSD UNIX [25]. However, there have been several versions of TCP in the literature [4, 16, 26, 27]. For instance, TCP Vegas version [28, 29] has been proposed in 1995, which improves several drawbacks of TCP Reno. TCP Vegas can achieve much higher throughput than TCP Tahoe/Reno mainly because of its improved congestion avoidance mechanism, which is why we focus on three versions of TCP Tahoe, Reno and Vegas in this thesis. In what follows, we briefly explain operational algorithms of TCP Tahoe, Reno, and Vegas. Refer to [29, 30] for detailed explanation.

The congestion avoidance mechanism of 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.

The notable difference in various versions of TCP is in their algorithms to change the window size. Hence, we first explain how the window size of the source terminal is changed. For easier understanding, typical evolutions of window sizes in TCP Tahoe, Reno, and Vegas are shown in Figure 1.1. These figures clearly illustrate the notable difference in TCP Tahoe, Reno, and Vegas. In what follows, we denote the window size at time t [sec] by $cwnd(t)$ [packets].

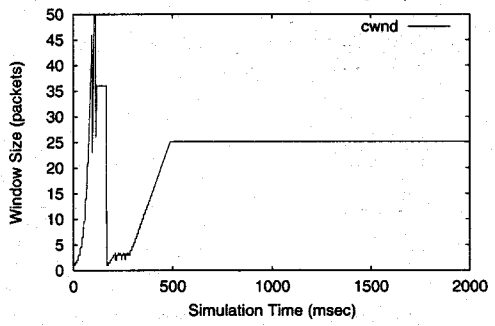
1.2.1 TCP Tahoe

In TCP Tahoe, the window size *cwnd* is cyclically changed as indicated in Figure 1.1(a). The window size continues to be increased until packet loss occurs. When it occurs, TCP knows that the network is congested, and throttles its window size to the size to 1 [packet]. TCP Tahoe has two phases of operation mode: Slow Start phase and Con-



(a) TCP Tahoe.

(b) TCP Reno



(c) TCP Vegas

Figure 1.1: Evolutions of Window Sizes in TCP Tahoe, Reno, and Vegas

gestion Avoidance phase. When an ACK packet is received by TCP at the sender side at time $t + t_A$ [sec], $cwnd(t + t_A)$ is changed from $cwnd(t)$ as follows (see, e.g., [30]);

$$cwnd(t + t_A) = \begin{cases} \text{Slow Start phase :} & \\ cwnd(t) + 1, & \text{if } cwnd(t) < ssth \\ \text{Congestion Avoidance phase :} & \\ cwnd(t) + \dfrac{1}{cwnd(t)}, & \text{if } cwnd(t) \geq ssth \end{cases} \tag{1.1}$$

where $ssth$ [packets] is a threshold value at which TCP changes its phase from Slow Start phase to Congestion Avoidance phase. TCP detects packet loss in two ways; time-out and duplicate ACKs [30]. Sender TCP sets RTO (Retransmission TimeOut) timer at every packet sent and if RTO timer expires before the corresponding ACK packets is received, the sender determines the packet is lost in the network. The length of the RTO timer is calculated from RTTs (Round Trip Times) of sending packets.

Since the RTO timer is usually set to a much larger value than the RTT, it takes long time for the sender TCP to detect packet losses. Therefore, an another mechanism called as *fast retransmit* is provided at TCP Tahoe. The source terminal detects packet losses in the network by receiving duplicate ACKs; that is, if the source terminal receives three

ACKs with the same sequence number, it probably indicates packet losses. In this case, the source terminal is allowed to retransmit the (possibly) lost packet before the RTO timer expires.

When packet loss is detected by timeout or duplicate ACKs, $cwnd(t)$ and $ssth$ are updated as follows;

$$\begin{aligned} ssth &= \frac{cwnd(t)}{2} \\ cwnd(t) &= 1 \end{aligned} \tag{1.2}$$

That is, TCP Tahoe enters Slow Start phase when packet loss occurs. Thus, TCP Tahoe indefinitely switches its operation mode between Slow Start phase and Congestion Avoidance phase.

1.2.2 TCP Reno

The operation algorithm of TCP Reno is equivalent to TCP Tahoe except that a packet is retransmitted differently when a packet loss is detected by duplicate ACKs. Namely, TCP Reno changes its window size according to Equation (1.1) in its Slow Start and Congestion Avoidance phases. However, when a packet loss is detected by receiving duplicate ACKs, TCP Reno halves its window size as following Equations;

$$\begin{aligned} ssth &= \frac{cwnd(t)}{2} \\ cwnd(t) &= ssth \end{aligned} \tag{1.3}$$

TCP Reno then enters an another operation mode called as "Fast Recovery phase" [30]. In this phase, change in the window size is determined by whether duplicate ACKs are received or not. Namely, when duplicate ACKs are received, the window size is increased by one packet. Otherwise, a non-duplicate ACK corresponding to the retransmitted packet is received, the window size is changed to $ssth$. Figure 1.1(b) shows a typical operation of TCP Reno.

1.2.3 TCP Vegas

In TCP Tahoe and Reno, the window size is increased until packet 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 packet loss. However, it becomes a problem since

the packet may be lost when the TCP connection itself causes the congestion because of its too large window size. If the window size is controlled appropriately such that a packet loss never occurs in the network, the throughput degradation due to throttled window could be avoided. This is the key idea of the congestion avoidance mechanism of TCP Vegas.

TCP Vegas employs an another approach for gauging how the network is congested. TCP Vegas controls its window size based on measurement of the actual RTTs of packets sent before. If observed RTTs become large, TCP Vegas recognizes that the network begins to be congested, and throttles its window size. If RTTs become small, on the other hand, TCP Vegas determines that the network is relieved from the congestion, and increases its window size. Then, the window size in an ideal situation becomes converged to a certain value as shown in Figure 1.1(c), and the throughput is not degraded. In Congestion Avoidance phase, the window size is updated as;

$$
\begin{aligned}
cwnd(t+t_A) &= \begin{cases} cwnd(t)+1, & \text{if } diff < \frac{\alpha}{base\_rtt} \\ cwnd(t), & \text{if } \frac{\alpha}{base\_rtt} < diff < \frac{\beta}{base\_rtt} \\ cwnd(t)-1, & \text{if } \frac{\beta}{base\_rtt} < diff \end{cases} \\
diff &= \frac{cwnd}{base\_rtt} - \frac{cwnd}{rtt}
\end{aligned}
\tag{1.4}
$$

where rtt [sec] is the 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 ACK packet is received.

Note that Equation (1.4) used in TCP Vegas indicates that if observed RTTs of the packets are identical, the window size remains unchanged. That can be seen by Figure 1.1(c), where the window size is converged to a fixed value in steady state.

1.3 Outline of Thesis

As discussed in Section 1.1, it is necessary to consider how we can modify the congestion control mechanisms of TCP to adjust to the new Internet services on the future high-speed network. In addition, it is also important to investigate the performance of TCP's congestion control mechanisms in more detail. In this thesis, therefore, we focus on the following two objectives;

- Investigation of TCP behavior through mathematical analysis

- Study of TCP's applicability to the future high-speed Internet

In the rest of this Section, we summarize the objectives of this thesis and refer to other related works in the literature.

1.3.1 Investigation of TCP Behavior through Mathematical Analysis

Fairness and Stability of the Congestion Control Algorithm of TCP [31-35]

As described above, inapplicability of the traditional transport-layer protocols such as TCP to the future high-speed network have been repeatedly claimed in the literature. Therefore, many researchers have been studying about TCP for high speed data transfer. Some of them have focused on packet buffering algorithm at the internal router, and some new algorithms have been proposed to achieve higher throughput and better fairness among connection at the router buffer [36, 37]. The other approach is to modify the congestion control algorithm of TCP to be appropriately applied to high-speed networks [6, 26, 38, 39]. However, most of past studies have concentrated on the throughput of TCP in spite of the fact that stability and fairness are other important issues, and those sometimes become more essential than effectiveness [40].

Therefore, in Chapter2, we focus on fairness and stability of three versions of TCP through mathematical analysis. Here, by "fairness", we mean that by dynamically adjusting window size of TCP, throughputs of connections sharing the bottleneck bandwidth is close. 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, and present some findings through the analytic approach. We consider two cases of network configuration: the homogeneous case, where two TCP connections have identical propagation delays, and the heterogeneous case where two connections have different propagation delays in order to make clear the effect of the difference of RTT (Round Trip Time) on fairness among connections. In addition to TCP Tahoe and Reno versions, We also evaluate the performance of TCP Vegas version [28, 29], since there is few researches which focuses on fairness aspects of TCP Vegas. In the analysis, we use the similar method to that appeared in [41] to evaluate fairness, and focus on the changes of the two connections' window sizes. Finally, we propose the enhance mechanisms of TCP Vegas' congestion control algorithm and evaluate its effectiveness in both homogeneous and heterogeneous cases.

Improvement of the Congestion Control Mechanism of TCP to Avoid Mis-retransmission [42-44]

In Chapter 3, we investigate the TCP behavior in the network where fixed amount of bandwidth is assigned to each connection. As discussed in Section 1.1, new networking technology makes it possible for each connection to be provided constant bandwidth to accommodate new Internet services. One of these services is ER (Explicit Rate) mode within ATM ABR (Available Bit Rate) service class. In the ER mode, the intermediate switch explicitly specifies the cell emission rate of the source end systems dependent on the number of active connections [45, 46], and each connection is guaranteed a fixed amount of the bandwidth when the number of active connections remains fixed. Although ABR service class is thought to be used by data transfer using TCP, TCP is essentially designed for sharing the network bandwidth among multiple connections without bandwidth assignment. It assumes that the packet transfer delay fluctuates as a function of time as in Ethernet, and most of literature regarding TCP assumes such networks.

We point out, in Chapter 3, that in such networks TCP may unnecessarily retransmit some packets which are not lost in the network, because RTT becomes fixed to a certain value. We call this wrong retransmission *mis-retransmit*. We also make clear that mis-retransmission may occur even in the current Internet where no bandwidth guarantee takes place, and that degrades TCP performance significantly. We evaluate this performance degradation analytically, and propose an enhancement technique to the congestion control mechanism of TCP to avoid performance degradation caused by mis-retransmissions.

1.3.2 Study of TCP's Applicability to the Future High-Speed Internet

Performance Evaluation and Parameter Tuning of TCP over ATM Networks [47-52]

The ATM (Asynchronous Transfer Mode) networks has been thought to provide large bandwidth by preparing effective traffic management mechanisms [53]. ATM has four service classes: CBR (Constant Bit Rate; or Deterministic Bit Rate in ITU-T terminology), VBR (Variable Bit Rate; or Statistical Bit Rate), UBR (Unspecified Bit Rate), and ABR (Available Bit Rate) service classes [54, 55], dependent on traffic characteristics and QoS (Quality of Service) demands. To meet the QoS requirements, essential for stable and efficient operation of ATM networks is congestion control. When we apply data communications to the ATM layer, two service classes are considered to be available: UBR and ABR service classes. The rate-based congestion control adopted in the ABR service classes is suitable to data communications, especially for the existing LAN traf-

fic [55-58]. The service offered by the ABR service class is sometimes referred to as "best-effort" since it cannot provide QoS in terms of, e.g., cell transfer delay. However, an appropriate control parameter setting of the rate-based congestion control can assure the cell loss ratio to be almost zero [57], which is very different from the UBR service class in which no congestion control mechanism is provided.

There have been much literature about UBR/ABR services for data transmission (see, e.g., [59-62] and references therein), but their main concern was ATM layer performance. As a next step, of course, we need to study the upper layer protocols to be applied to the ABR service class. Such a study is especially important when the existing TCP/IP network is migrated to ATM networks. In Chapter 4, we investigate performance of TCP over ABR/UBR networks for data transmission, and show some results that explain the effectiveness of ABR service class for TCP traffic, and inapplicability of UBR service class.

Performance Evaluation of HTTP/TCP on Asymmetric Networks[63-67]

Different from the traditional analog modem networks, some of new network technologies regarding access networks, which connects each users to ISP, may provide *asymmetric* bandwidth for upstream (from the client to the server) and downstream (from the server to the client). For example, ADSL (Asymmetric Digital Subscriber Line) [68, 69] uses existing telephone lines, and offer 1$\sim$20 Mbps for upstream, and 0.1$\sim$1 Mbps for downstream. Cable modem service can provide 5$\sim$50 Mbps for upstream, and 0.5$\sim$5 Mbps for downstream. Those technologies are considered to be suitable for the Internet access, 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. 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 [14]. In [14], the authors pointed out that the performance of TCP on asymmetric networks is degraded due to the traffic burstiness of the sender. However, in [14], the authors only focus on the mean throughput of TCP Tahoe and Reno.

In Chapter 5, we extensively investigate the performance of HTTP/TCP on asymmetric networks with analytical and simulation approach. Our analytical approach is similar to the one adopted in [14], but in addition to traditional HTTP/1.0 and TCP Tahoe, we consider new HTTP/TCP protocols; HTTP/1.1 [70] and TCP Vegas. Through the analysis, we discuss which combination of HTTP and TCP protocols is appropriate in asymmetric networks. We also point out that the original TCP Vegas is not suitable for the asymmetric networks because its essential congestion control mechanisms make

it not to fully utilize the downstream bandwidth. We thus propose to modify TCP Vegas with minimum change to resolve the problem specific to the asymmetric network.

Comparisons of Packet Scheduling Algorithm for Fair Service among Connections[71-75]

As described before, it is necessary to provide commercial network services by the Internet, which only provides the best effort service currently. That is, a new service should be available within the network to support the *differentiated services* among the users [17, 18], where a constant throughput may be preferred to some connections, or QoS support is necessary for real-time applications. Another important service that the next-generation Internet should support is *fair allocation of the bandwidth*, which is our main subject of Chapter 6. It is one of most desired features for elastic applications, but not supported by the current Internet, and we believe that network fairness is as important as network efficiency. One of these services found in the literature is the USD (User Share Differentiation) scheme described in [19], where users are provided different service qualities from ISPs (Internet Service Providers) based on the contracts. However, the authors in [19] do not provide a quantitative evaluation of USD to show how the users are *differentiated*.

One way to realize such service differentiation seems to be DRR (Deficit Round Robin) presented in [22] where the WRR (Weighted Round Robin) scheduling is performed among active connections. In [22], an extensive evaluation of the DRR algorithm is provided, but they assume Poisson arrivals of packets from each connection. That is, the authors do not consider the behavior of the upper-layer protocol, i.e., TCP.

In Chapter 6, therefore, we focus on the degree of *fairness* provided to TCP connections by comparing three packet scheduling algorithms at the router, those are FIFO (First In First Out), RED (Random Early Detection), and DRR. For TCP, we consider the Reno version, which has widely been used in the current Internet, and TCP Vegas version, adopting a different congestion control mechanism from TCP Reno for larger performance gain. Although the packet scheduling algorithms and TCP versions that we use in Chapter 6 are not new. The main objective is that the fairness properties of three packet scheduling algorithms are also shown through analytical results.

Chapter 2

Fairness and Stability of Congestion Control Mechanisms of TCP

Although many researchers have studied about TCP for high speed data transfer, 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. In this chapter, we focus on stability and fairness of several versions of TCP; TCP Tahoe, Reno and Vegas through a mathematical analysis. We consider two cases of network model: homogeneous case where two TCP connections have identical propagation delays, and heterogeneous case where two connections have different propagation delays. Through the mathematical analysis and simulation, we point out the instability of TCP Tahoe and TCP Reno, and the unfairness of TCP Vegas. Based on the analysis results, we finally propose the improvements of TCP Vegas for fairness enhancement, and evaluate its effectiveness both in the homogeneous and heterogeneous cases.

2.1 Network Model

The network model that we will use in the analysis and simulation in this chapter is depicted in Figure 2.1. 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 [packet/sec]. The buffer size of SW1 is B [packets]. The propagation delays between SESi and DESi are τ_i ($i = 1, 2$).

In analysis and simulation, we consider the situation that Connection 1 starts to trans-

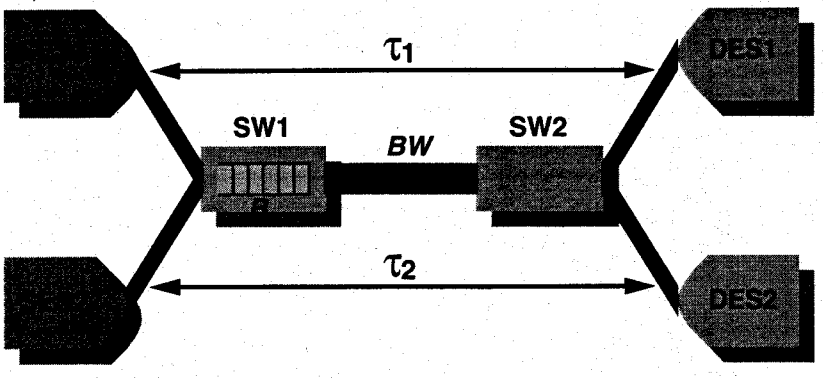


Figure 2.1: Network Model

fer data packets at first, and Connection 2 joins the network afterward. Each SES transmits data packets 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 packet 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 $cwnd_2(t)$.

2.2 Analysis and Evaluation

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.

2.2.1 Analysis Method

To investigate fairness between two connections, we employ the $cwnd_1$-$cwnd_2$ graph depicted in Figure 2.2 [41]. 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 the point is on the line. By the "Efficiency Line", it is shown whether the link is fully utilized or not. All packets 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 packets of both connections are served. By approximately representing the arrival

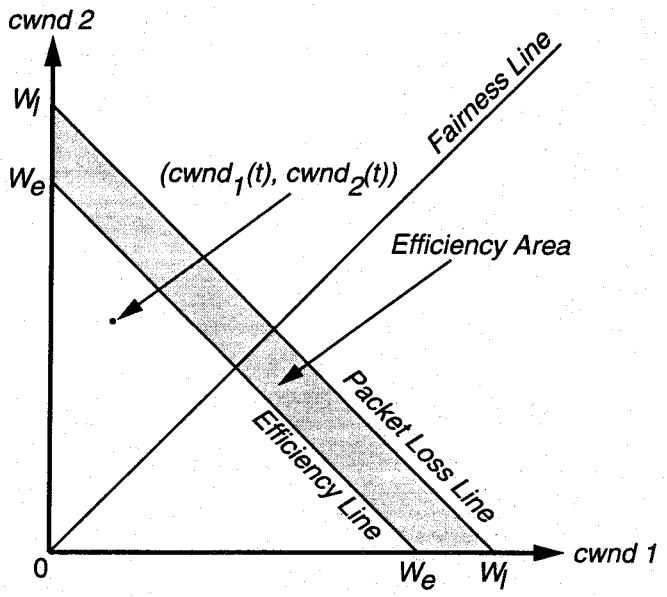


Figure 2.2: $cwnd_1$-$cwnd_2$ Graph

rate of packets by $cwnd_1(t)/\tau_1$ and $cwnd_2(t)/\tau_2$, we have a relation

$$\mu = \frac{cwnd_1(t)}{\tau_1} + \frac{cwnd_2(t)}{\tau_2} \tag{2.1}$$

We further introduce W_e as

$$W_e = cwnd_1(t) + cwnd_2(t) \tag{2.2}$$

such that the values of $cwnd_1(t)$ and $cwnd_2(t)$ satisfy Equation(2.1). 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 ($\tau_1 = \tau_2 = \tau$), Equation (2.2) after substituting Equation (2.1) becomes

$$W_e = 2\tau\mu$$

The Packet Loss line in the figure represents $cwnd_1 + cwnd_2 = W_l$ [packets], where W_l is the sum of W_e and the buffer size of the intermediate bottleneck switch;

$$W_l = W_e + B$$

Thus, packet loss occurs if the point is beyond the "Packet Loss Line". If the points are located between "Efficiency Line" and "Packet 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 packet 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 2.3 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 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 [msec]. We set μ = 20 [Mbps], $\tau_1 = \tau_2$ = 5 [msec], B = 10 [packets] and m = 1 [Kbytes] for parameters of the model shown in Figure 2.1. The other parameters are set as α = 2, β = 4 for TCP Vegas. Figure 2.4 shows the $cwnd_1$-$cwnd_2$ graph obtained from Figure 2.3.

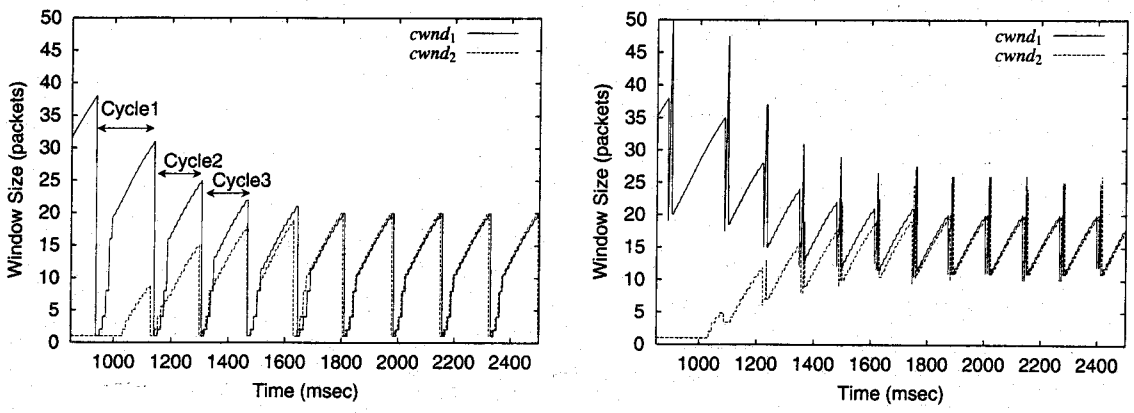
In Figures 2.5 and 2.6, we show the heterogeneous case where the propagation delays of the two connections are different. In simulation, we set μ = 20 [Mbps], B = 10 [packets], τ_1 = 4 [msec], τ_2 = 8 [msec], m =1 [Kbytes], and α =2, β = 4 for parameters of TCP Vegas. Connection 2 joins the network at time t=1500 [msec]. We will explain the effect of propagation delay on the congestion control mechanisms of TCP by using Figures 2.5 and 2.6 and our analytical results.

2.2.2 TCP Tahoe

In TCP Tahoe, the change of the window size is cyclic as shown in Figure 1.1(a) where the single connection utilizes the link. It is also true when two connections with identical propagation delays share the link (Figure 2.3(a)) since packets from two 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 packet simultaneously, and injects a new packet 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 packets 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 packets. It is true that we treat a special case for the network configuration, but the problem described above is inherent in TCP Tahoe.

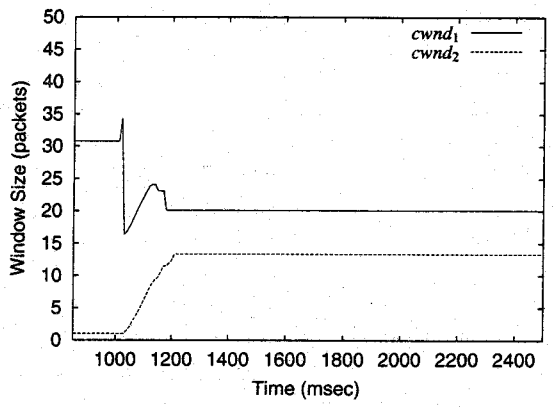
When propagation delays of two connections are different, on the other hand, the above discussions never be directly applicable. However, we can confirm that even if two connections have different propagation delays, the packet losses of both connections are likely to occur simultaneously in Figure 2.5(a). Therefore, in the analysis, we will assume that packet losses of the two connections take place simultaneously.

We introduce the following notations. Cycle i starts at the time when $(i-1)$ th packet



(a) TCP Tahoe

(b) TCP Reno



(c) TCP Vegas

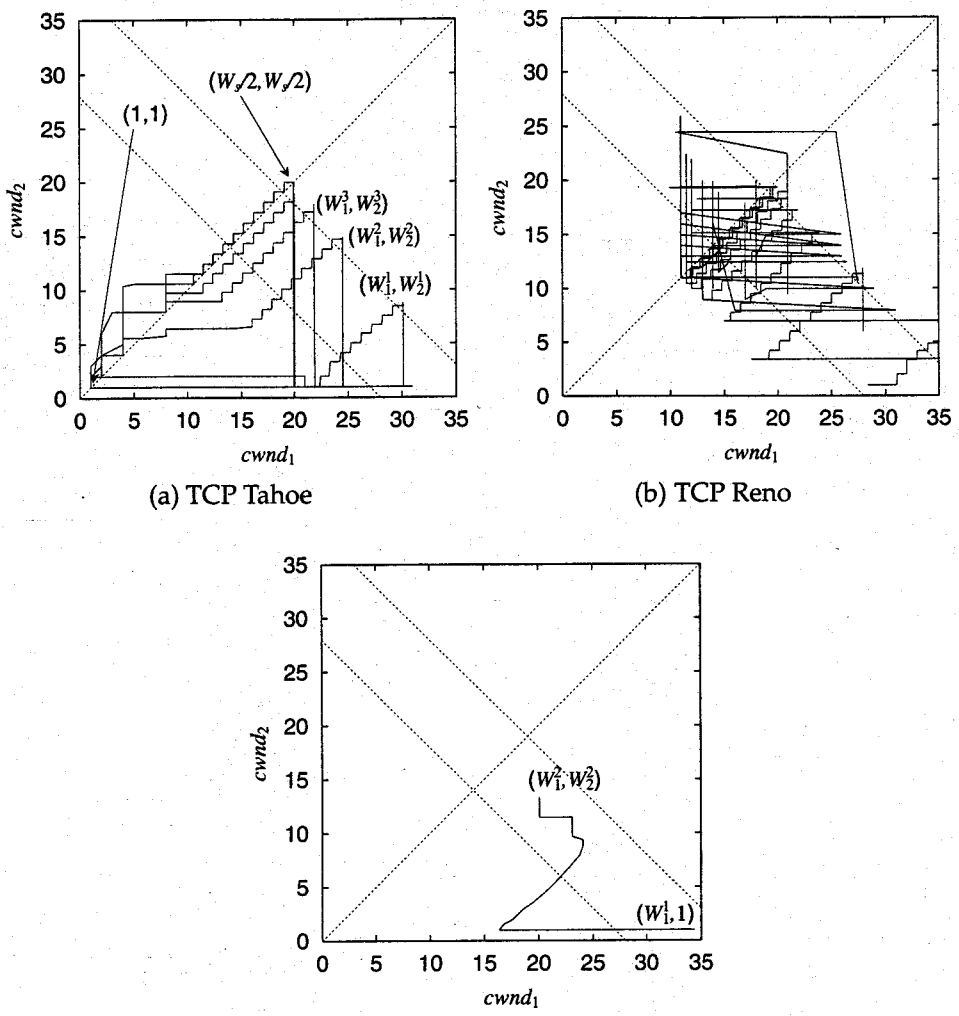
Figure 2.3: Simulation Results of Homogeneous Case

is lost, and terminates at i th packet loss. W_1^i [packets] and W_2^i [packets] are window sizes of Connections 1 and 2 when i th packet of two connections are lost. Similarly, $ssth_1^i$ [packets] and $ssth_2^i$ [packets] are defined as $ssthresh$ of cycle i for two connections, respectively. Let us assume that cycle i begins at time t = 0 [sec]. From Equation (1.2), we obtain;

$$ssth_j^i = \frac{W_j^{i-1}}{2}, \quad j = 1, 2 \tag{2.3}$$

When packet loss occurs, the window size is reset to one packet (since fast recovery 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 Equation (1.1));

$$cwnd_j(t) = \frac{(t -_{ssth} t_j^i)}{2\tau_j} + ssth_j^i, \quad j = 1, 2 \tag{2.4}$$



(a) TCP Tahoe

(b) TCP Reno

(c) TCP Vegas

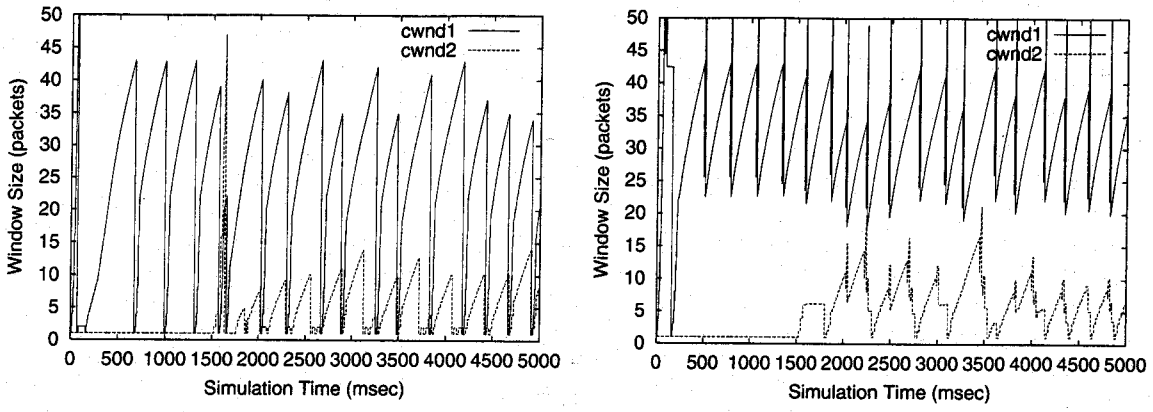
Figure 2.4: $cwnd_1$-$cwnd_2$ Graph of Homogeneous Case

where ${}_{ssth}t_j^i$ [sec] is the time when Slow Start phase terminates, that is, when $cwnd_j(t)$ reaches $ssth_j^i$. At the end of cycle i, i th packet loss takes place in both connections since the sum of the window sizes of both connections reaches W_l (defined in Equation (2.3)), i.e.,

$$W_l = cwnd_1(t_{loss}^i) + cwnd_2(t_{loss}^i) \tag{2.5}$$

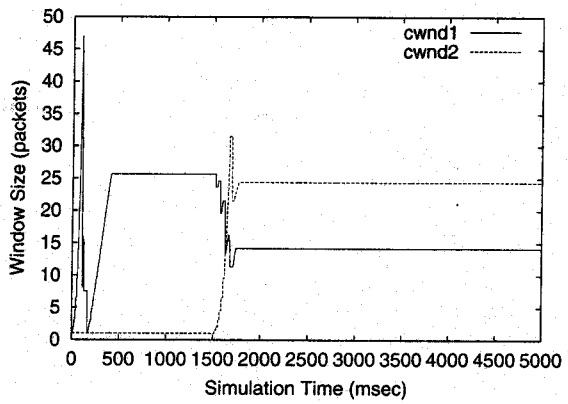
We can obtain t_{loss}^i [sec], the time when i th packet loss occurs, from Equations (2.3) and (2.4) as follows;

$$t_{loss}^i = \frac{\tau_1 \cdot \tau_2}{\tau_1 + \tau_2} W_l \tag{2.6}$$



(a) TCP Tahoe

(b) TCP Reno



(c) TCP Vegas

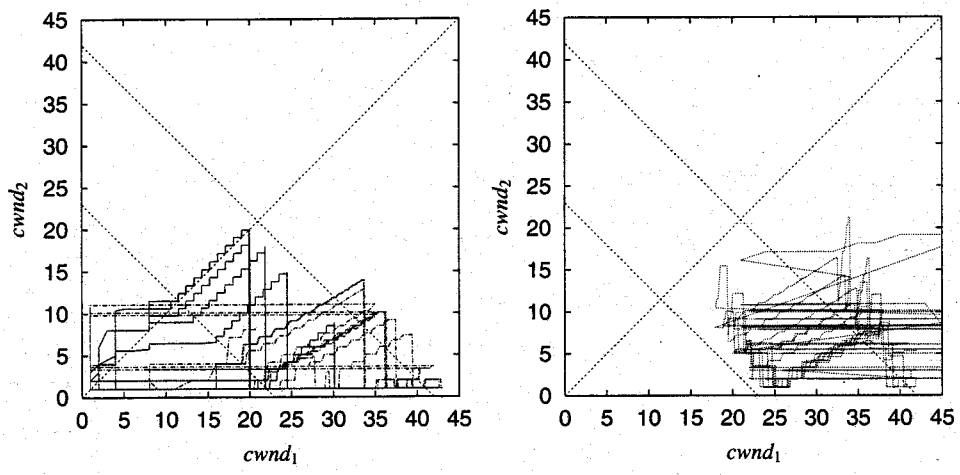
Figure 2.5: Simulation Results of Heterogeneous Case

Finally, W_j^i is obtained from Equations (2.3) through (2.6) as;

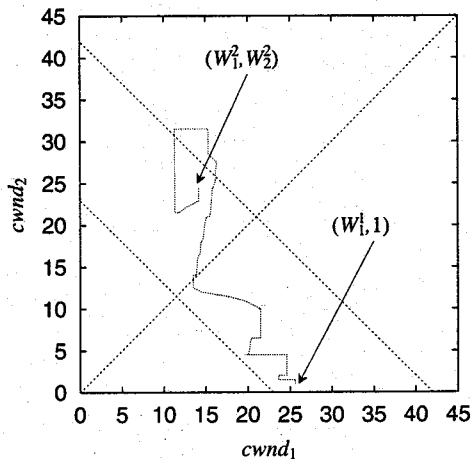
$$\begin{aligned} W_j^i &= \frac{W_j^i}{2} + \frac{1}{\tau_j}\frac{\tau_1 \cdot \tau_2}{2(\tau_1+\tau_2)}W_l \\ &= \frac{1}{\tau_j}\frac{\tau_1 \cdot \tau_2}{2(\tau_1+\tau_2)}W_l - \left(\frac{1}{2}\right)^{i-1}\left(\frac{1}{\tau_j}\frac{\tau_1 \cdot \tau_2}{2(\tau_1+\tau_2)}W_l - W_j^1\right) \end{aligned} \tag{2.7}$$

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 1.1(a), 2.3(a) and 2.5(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



(a) TCP Tahoe (b) TCP Reno



(c) TCP Vegas

Figure 2.6: $cwnd_1$-$cwnd_2$ Graph of Heterogeneous Case

connection with longer propagation delay suffers from the small window size. This can be observed in Figures 2.5(a) and 2.6(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 [packets] as the number of packets transmitted in cycle i of the connection j, and S_j [packets/sec] as throughput for the connection j. That is, the following relation holds;

$$S_j = \frac{A_j^i}{t_{loss}^i} \tag{2.8}$$

Thus, by utilizing Equations (2.3) through (2.7), we have

$$\begin{aligned} S_j &= \frac{A_j^i}{t_{loss}^i} \\ &= \frac{\int_0^{t_{loss}^i} cwnd_j(t)dt}{t_{loss}^i} \\ &= \frac{3W_j^i}{4\tau_j} \end{aligned}$$

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

$$S_j \to \frac{3}{\tau_j^2} \frac{\tau_1 \cdot \tau_2}{2(\tau_1 + \tau_2)} W_l \tag{2.9}$$

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

2.2.3 TCP Reno

As described in Section 1.2, 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 packet 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 Equation (2.7). 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 2.3(b) and 2.4(b) for the homogeneous case and Figures 2.5(b) and 2.6(b) for the heterogeneous case.

2.2.4 TCP Vegas

In TCP Vegas, it is noticeable that the window sizes of both connections remains constant at different values as shown in Figures 2.3(c) and 2.5(c). It can also be observed in the $cwnd_1$-$cwnd_2$ graph in Figures 2.4(c) and 2.6(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 [packets] (the window size of Connec-

tion 1 when Connection 2 is activated), and W_1^2 [packets], W_2^2 [packets] (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 [packets] and l_2 [packets] be the mean numbers of packets 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 Equation (1.4);

$$\frac{\alpha}{base\_rtt_1^1} < \frac{W_1^1}{base\_rtt_1^1} - \frac{W_1^1}{rtt_1} < \frac{\beta}{base\_rtt_1^1} \tag{2.10}$$

where $base\_rtt_1^1$ [sec] is $base\_rtt$ of Connection 1, being equal to the round trip time without queueing delays at the switch buffer. That is,

$$base\_rtt_1^1 = 2\tau + \frac{1}{\mu} \tag{2.11}$$

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

$$rtt_1 = 2\tau + \frac{l_1 + 1}{\mu} \tag{2.12}$$

From Equations (2.11) and (2.12), Equation (2.10) can be rewritten as;

$$\{2\tau\mu + (l_1 + 1)\}\frac{\alpha}{l_1} < W_1^1 < \{2\tau\mu + (l_1 + 1)\}\frac{\beta}{l_1} \tag{2.13}$$

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

$$W_1^1 = 2\tau\mu + l_1 \tag{2.14}$$

By substituting Equation (2.14) into Equation (2.13), we simply have

$$\alpha < l_1 < \beta. \tag{2.15}$$

Also, Equation (2.15) can be written by using Equation (2.14) as

$$2\tau\mu + (\alpha + 1) < W_1^1 < 2\tau\mu + (\beta + 1) \tag{2.16}$$

From the above equations, we observe that in steady state, the mean number of packets 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 packets into the network, the number of packets 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 Equation (1.4). Since all packets of both connections are served at the bottleneck link with the bandwidth of μ [packets/sec], the following equation is satisfied;

$$\frac{W_1^2}{rtt_1} + \frac{W_2^2}{rtt_2} = \mu \tag{2.17}$$

The window size of each connection changes according to Equation (1.4) as follows;

$$\frac{\alpha}{base\_rtt_1} < \frac{W_1^2}{base\_rtt_1} - \frac{W_1^2}{rtt_1} < \frac{\beta}{base\_rtt_1} \tag{2.18}$$

$$\frac{\alpha}{base\_rtt_2} < \frac{W_2^2}{base\_rtt_2} - \frac{W_2^2}{rtt_2} < \frac{\beta}{base\_rtt_2} \tag{2.19}$$

where rtt_1 [sec], rtt_2 [sec], $base\_rtt_1$ [sec], $base\_rtt_2$ [sec] are Round Trip Time and $base\_rtt$ of Connection 1 and Connection 2, respectively, and can be obtained as follows;

$$rtt_1 = 2\tau_1 + \frac{l_2}{\mu} \tag{2.20}$$

$$rtt_2 = 2\tau_2 + \frac{l_2}{\mu} \tag{2.21}$$

$$base\_rtt_1 = 2\tau_1 + \frac{1}{\mu} \tag{2.22}$$

$$base\_rtt_2 = 2\tau_2 + \frac{l_1}{\mu} \tag{2.23}$$

By substituting Equations (2.20) from (2.23) into (2.18) (2.19), and after some manipulation, we have

$$(2\tau_1\mu + l_2)\frac{\alpha}{l_2} < W_1^2 < (2\tau_1\mu + l_2)\frac{\beta}{l_2} \tag{2.24}$$

$$(2\tau_2\mu + l_2)\frac{\alpha}{l_2 - l_1} < W_2^2 < (2\tau_2\mu + l_2)\frac{\beta}{l_2 - l_1} \tag{2.25}$$

Namely, network status point ($cwnd_1(t)$, $cwnd_2(t)$) converges to the values satisfying Equations (2.24), (2.25), and (2.17). Furthermore, from the condition that network status point ($cwnd_1(t)$, $cwnd_2(t)$) that satisfying Equations (2.24), (2.25), and (2.17) exists, we

can determine the range of l_2, the numbers of packets queued in the switch buffer after Connection 2 joins the network. It is obtained by solving Equation (2.24) and (2.25) for l_2 as follows;

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

We can observe from Equations (2.24) and (2.25) 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 Equations (2.24) and (2.25) 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 (Equation (1.4)).

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 (Equations (2.22) and (2.23)) even in the homogeneous case with identical propagation delays. When Connection 2 joins the network, the switch buffer is occupied by several packets 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 Equation (1.4). This cannot be avoided in TCP Vegas if the number of packets 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 2.3(c) and 2.4(c) is one example, but by repeating the simulation experiments, we observe that the values of $\frac{W_1^2}{W_2^2}$ range from 1.03 to 1.58. On the other hand, Equations (2.24) and (2.25)) show that the upper and lower values of $\frac{W_1^2}{W_2^2}$ are 0.95 to 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 Equations (2.22) and (2.23). 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 Equations (2.22) and (2.23).

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 Section, we will explain our enhanced TCP Vegas, and show some analytic results to

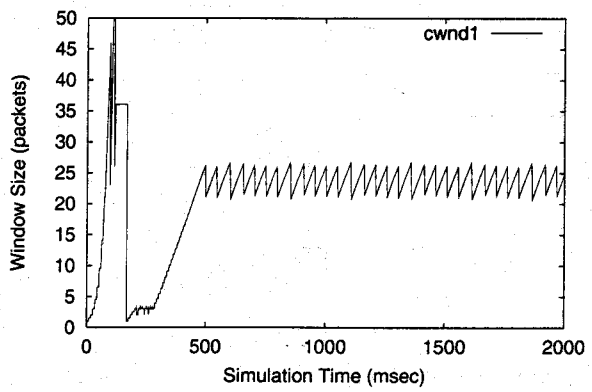


Figure 2.7: Evolution of Window Size of Enhanced TCP Vegas

confirm the effectiveness of our proposed method.

2.3 Enhanced TCP Vegas

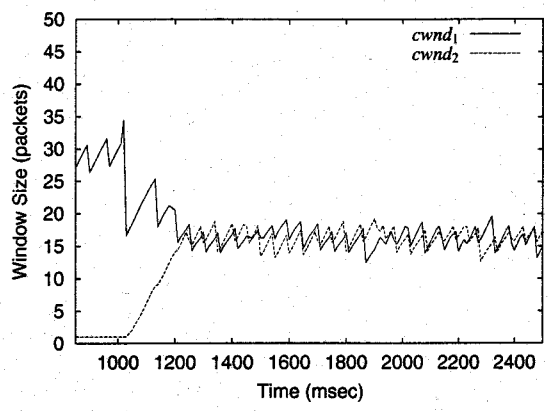
Equation (1.4) used in TCP Vegas indicates that if RTTs of the packets are stable, the window size remains unchanged. The range that the network is viewed as "stable" was derived in the previous Subsection. It is a fundamental problem of TCP Vegas, and our solution is to eliminate the condition of unchanging the window size. The following algorithm is used in our enhanced TCP Vegas to *prevent* the convergence of the window size;

$$\begin{aligned} cwnd(t+t_A) &= \begin{cases} cwnd(t)+1, & \text{if } diff < \frac{\delta}{base\_rtt} \\ cwnd(t)-1, & \text{if } \frac{\delta}{base\_rtt} \leq diff \end{cases} \\ diff &= \frac{cwnd(t)}{base\_rtt} - cwnd(t)rtt \end{aligned} \tag{2.26}$$

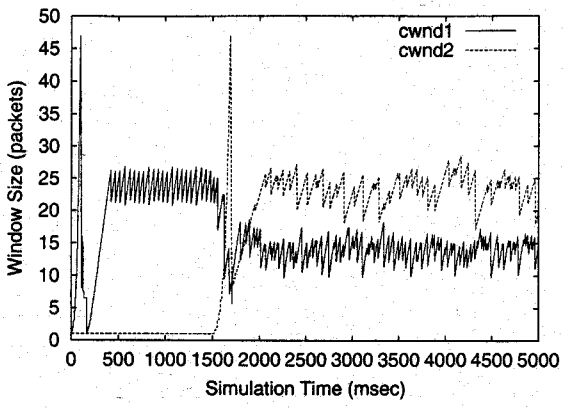
where δ is a some small constant value. The same algorithm can be obtained by setting α=β in TCP Vegas (Equation (1.4)), which clearly shows that the condition of unchanging the window size is eliminated. Figure 2.7 shows a typical example of our enhanced Vegas version. Here, we use δ=3. We can see from 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 Equation (2.26), we first have

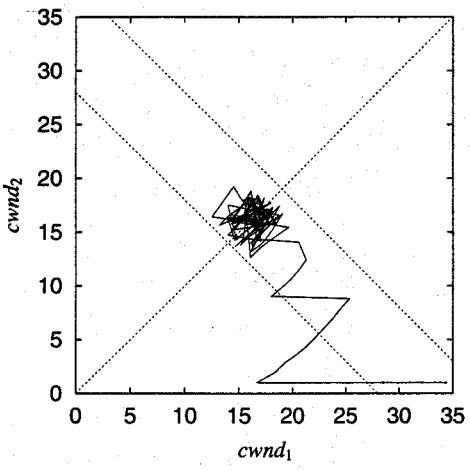
$$\frac{cwnd}{base\_rtt} - \frac{cwnd}{rtt} = \frac{\delta}{base\_rtt} \tag{2.27}$$



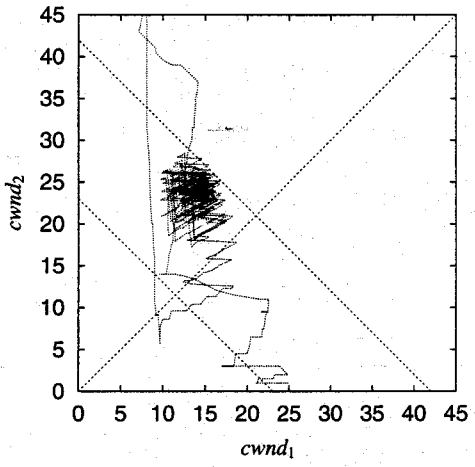
(a) Homogeneous Case: Window Size of Two Connections.



(b) Heterogeneous Case: Window Size of Two Connections.



(c) Homogeneous Case: $cwnd_1$-$cwnd_2$ Graph.



(d) Heterogeneous Case: $cwnd_1$-$cwnd_2$ Graph.

Figure 2.8: Simulation Results of Enhanced TCP Vegas.

Let W_1^2 [packets] and W_2^2 [packets] be the central points of oscillations of the window sizes of Connections 1 and 2, respectively. By applying Equation (2.27) to Connection 1 and Connection 2, the following equations can be obtained;

$$\frac{W_1^2}{base\_rtt_1} - \frac{W_2^2}{rtt_1} = \frac{\delta}{bnase\_rtt_1} \tag{2.28}$$

$$\frac{W_2^2}{base\_rtt_2} - \frac{W_2^2}{rtt_2} = \frac{\delta}{base\_rtt_2} \tag{2.29}$$

where rtt_1 [sec], rtt_2 [sec], $base\_rtt_1$ [sec] and $base\_rtt_2$ [sec] are determined from;

$$rtt_1 = 2\tau_1 + \frac{l_2}{\mu} \tag{2.30}$$

$$rtt_2 = 2\tau_2 + \frac{l_2}{\mu} \tag{2.31}$$

$$base\_rtt_1 = 2\tau_1 + \frac{1}{\mu} \tag{2.32}$$

$$base\_rtt_2 = 2\tau_2 + \frac{1}{\mu} \tag{2.33}$$

Note that Equation (2.33) is different from that of TCP Vegas (Equation (2.23)). It is because our enhanced method simply prevents the convergence of the window size as shown in Equation (2.26). Then the window sizes of both connections are changed dynamically. It also leads to the fluctuation of the number of packets at the switch buffer, and thus $base\_rtt_1$ and $base\_rtt_2$ become converged to the same value.

From Equation (2.17), Equations (2.28) and (2.29) can be solved as follows;

$$W_1^2 = (2\tau_1\mu + l_2)\frac{\delta}{l_2} \tag{2.34}$$

$$W_2^2 = (2\tau_2\mu + l_2)\frac{\delta}{l_2} \tag{2.35}$$

Furthermore, in similarly to TCP Vegas, we can obtain l_2 from Equations (2.17), (2.34) and (2.35);

$$l_2 = 2\delta \tag{2.36}$$

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 (Equations (2.24) and (2.25)) may be different because it has the *range* in the condition that the window size remains unchanged (Equation (1.4)). On the other hand, in enhanced TCP Vegas, it is avoided by oscillating the window size.

From Equations (2.34) and (2.35), 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 Subsection 2.2.4. 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 2.8(c) and 2.8(d)). The network status points $(cwnd_1(t), cwnd_2(t))$ oscillate around the 'Fairness Line', and the range of oscillation falls between 'Efficiency Line' and 'Packet 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.

2.4 Conclusion

In this Chapter, 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 and stability 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 multihop 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 [22, 36], researchers focused on isolation of *ill-behaved* flows emitting packets 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.

Chapter 3

Improvement of the Congestion Control Mechanism of TCP to Avoid Mis-retransmission

Some of new networking technologies make it possible for each connection to be provided constant bandwidth to accommodate new Internet services. One of there services is ER (Explicit Rate) mode within ATM ABR (Available Bit Rate) service class. In the ER mode, the intermediate switch explicitly specifies the cell emission rate of the source end systems dependent on the number of active connections [45, 46], and each connection is guaranteed a fixed amount of the bandwidth during which a number of active connections remains to be fixed. However, TCP is essentially designed for sharing the network bandwidth among multiple connections without bandwidth assignment. It assumes that the packet transfer delay fluctuates as a function of time as in Ethernet, and TCP performance under such bandwidth-guaranteed networks has not been investigated enough.

In this Chapter, we investigate TCP's problem in the packet retransmission mechanism of TCP. The problem occurs when a fixed amount of bandwidth is assigned to a TCP connection, or when an RTT (Round Trip Times) of a TCP connection changes frequently. In this case, TCP frequently performs unnecessary packet retransmission (we call *mis-retransmission* in this thesis), which causes serious performance degradation. In the analysis of this chapter, We focus on the change of RTT and RTO values, and make clear why mis-retransmission occurs. We also point out that mis-retransmission may occur even in the current Internet where no bandwidth guarantee takes place because of temporary oscillation of RTTs. We evaluate this performance degradation analytically, and propose an enhancement technique to the congestion control mechanism of TCP to avoid performance degradation caused by mis-retransmissions.

3.1 Timeout-based Retransmission Mechanism of TCP

In this Section, we briefly introduce the timeout-based retransmission mechanism of TCP now in broadly use [30, 76]. TCP employs a timeout based retransmission mechanism to react network congestion. More specifically, the TCP sender sets the RTO timer for each sending packet. Until the RTO timer expires, the sender may not receive a corresponding ACK packet from the receiver. At that time, the sender determines that the packet is lost within the network, and retransmits that packet. At the same time, the sender throttles the window size to one packet because the lost packet indicates the network congestion.

The value of RTO timer is calculated using RTTs of packets which have already been successfully transmitted. When the TCP sender receives ACK from the receiver, it determines the next value of the RTO timer (RTO_n [sec]) according to the RTT value of the n-th packet (RTT_n [sec]]). RTO_n is determined in the following way [30];

$$
\begin{aligned}
Err_n &= RTT_n - A_{n-1} && (3.1)\\
A_n &= A_{n-1} + g \cdot Err_n && (3.2)\\
D_n &= D_{n-1} + h \cdot (|Err_n| - D_{n-1}) && (3.3)\\
RTO_n &= A_n + k \cdot D_n && (3.4)
\end{aligned}
$$

where A_n represents the weighted average value of RTTs given as

$$A_n = (1-g) \cdot A_{n-1} + g \cdot RTT_n, \quad (3.5)$$

and D_n does the fluctuation part of RTTs as

$$D_n = (1-h) \cdot D_{n-1} + h \cdot |RTT_n - A_{n-1}|. \quad (3.6)$$

The coefficients, g, h and k, are generally set as $g = 0.125$, $h = 0.25$, $k = 4$ [30]. As shown in Equation (3.4), the value of RTO timer (we simply denote "RTO value" in the rest of this Chapter) is the sum of mean and fluctuation part of RTT, and $k = 4$ achieves the safe operation. However, above equations imply that no fluctuation of RTTs leads to a fixed value of A_n's (see Equation (3.2)), and therefore, D_n reaches zero (Equation (3.3)). Then, RTO_n also reaches a fixed value identical to A_n's, the mean of RTTs. This is a main problem that we will treat in the following Sections. That is, even if the RTT of some packet is increased slightly, it is recognized as the packet loss after the RTO value becomes very close to the RTT values.

TCP has another retransmission mechanism called fast retransmission, where the sender can detect a few numbers of packet losses within the window. If the packet loss

is detected, the packet is retransmitted without waiting the timeout. As shown in Section 3.2, our problem on mis-retransmission cannot be resolved by introducing the fast retransmission algorithm.

3.2 Mis-Retransmission of TCP Packets

In this Section, we will focus on the case where the ER mode of the ATM ABR service class is applied to TCP, and explain why and how mis-retransmission of TCP packet occurs in the network when a fixed amount of bandwidth is assigned to the TCP connection. Its analysis method is also shown.

3.2.1 Why Mis-retransmission Occurs?

As having been described in the previous Section, when a fixed amount of bandwidth is assigned to a TCP connection, the RTO value is converged. It can be verified by repeatedly applying Equations (3.1) through (3.4) under the condition that RTT_n is fixed at rtt. That is,

$$\lim_{i \to \infty} RTO_i = rtt. \tag{3.7}$$

This fact indicates that if the fluctuation of RTT is very small compared with the mean of RTT, the RTO value becomes close to RTT itself. After then, if RTT is suddenly increased due to decrease of the assigned bandwidth or increase of the machine load, RTT can easily become larger than the RTO value. It is because the calculation of RTO value (Equations (3.1) through (3.4)) cannot follow the sudden change of RTT. As a result, the TCP sender recognizes that the packet is lost within the network, and it retransmits the packet, which is just a mis-retransmission. Such a case is likely to happen in the ER mode of the ATM ABR service class. In the ER mode, the intermediate switch of the network explicitly specifies cell emission rate to each connection according to the number of active connections [45, 46]. If the network condition is stable, that is, if the number of active connections is fixed, each connection is assigned a fixed amount of the bandwidth. It means that when the ER mode is applied to TCP, the RTO value easily becomes close to RTT. When the new connection actively joins the network, the bandwidth assigned to the existing connections suddenly decrease in the ER mode. Then, RTT is increased, and becomes larger than the RTO value. It results in packet retransmissions. However, it is completely unnecessary in this case since the packet is not lost within the network, but is only delayed due to the decrease of the assigned bandwidth.

3.2.2 TCP Behavior after Mis-retransmission

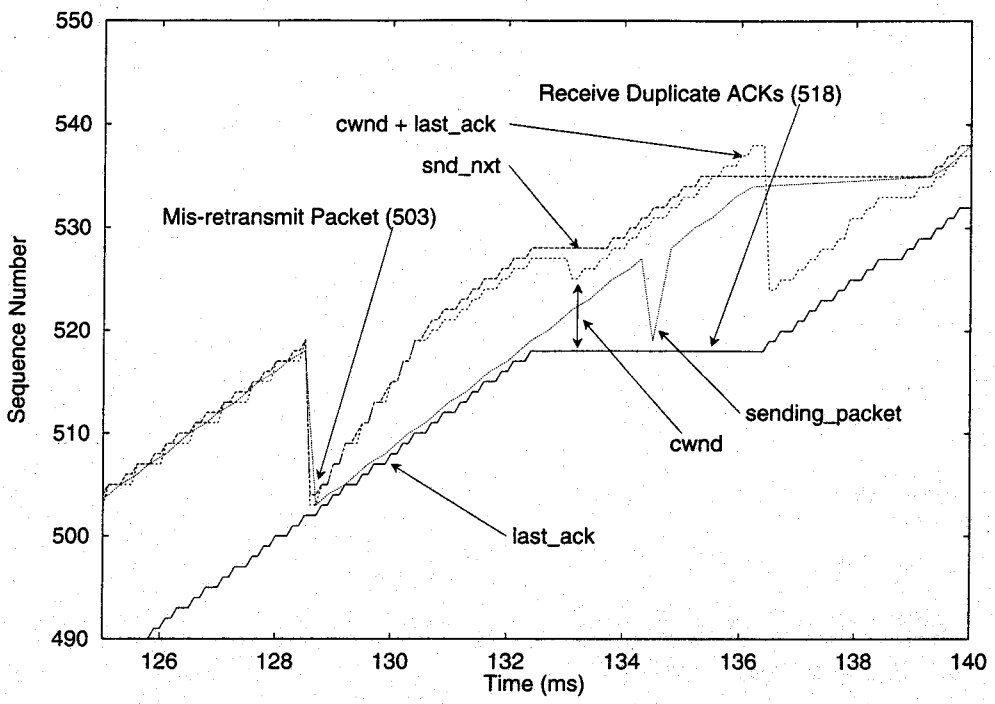
Simulation Experiment 1

As an illustrative example, we show a detailed behavior of the TCP connection using simulation experiments, and explain how TCP acts against the mis-retransmission. In the simulation experiment, we only consider the single connection using [150 Mbps] link. At time 0 [msec], the TCP sender starts transmission. In simulation, the propagation delay between source and destination terminals is set to 1 [msec], and the packet size of TCP is set to 4 [Kbyte]. The maximum window size of TCP is 64 [Kbyte]. We generated the mis-retransmission by intentionally setting the RTO value of the packet No.503 to a smaller value.

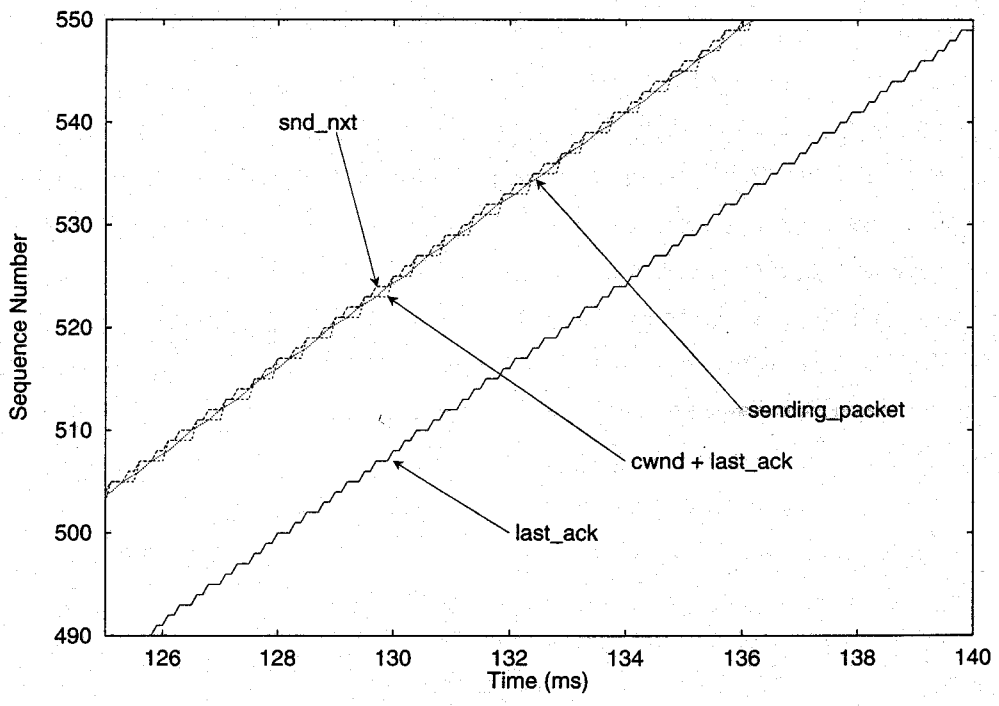
In Figure 3.1(a), the line labeled as "last_ack" represents the number of successfully acknowledged packets. The line with "snd_next" shows the sequence number of the packet that is expected to be transmitted next. The sequence number of the packet that has been sent by the TCP sender is shown by the line labeled as "sending_packet." The maximum sequence number that the TCP sender can transmit the packet is shown by the line "cwnd+last_ack," and therefore the difference between the line "last_ack" and the line "cwnd+last_ack" represents the congestion window size. By intentionally setting the RTO value of the packet No.503 to a smaller value, mis-retransmission of the packet No.503 occurs at time 128.5 [msec].

As shown in the figure, the window size (cwnd) is throttled down at time 128.5 [msec] due to mis-retransmission of the packet, that is, the mis-retransmission leads to the TCP's slow start phase [30]. After the mis-retransmission happens, the sender retransmits all the packets within the window from time 128.5 [msec] to 132.5 [msec], but all of those are needless to be retransmitted. Since these packets have been correctly received at the destination terminal, the ACKs corresponding to the retransmitted packets are for the same packet. In simulation, it is the packet No.518. It causes TCP's fast retransmission [30, 10] for the No.519 packet, and the window size is again throttled at 136.5 [msec]. After that, the TCP sender enters congestion avoidance phase [30], where the window size increases linearly as a function of time. That is, the mis-retransmission causes much degradation of the TCP throughput because of the needless retransmission process.

For only comparison purposes, we also conducted the simulation experiment where the mis-retransmission is avoided by setting the RTO value to be infinite. Since we assume that no cell loss occurs within the network, the infinite the RTO value works well in this simulation experiment. The result is shown in Figure 3.1(b). By comparing with this figure, the previous result in Figure 3.1(a) presents significant throughput degradation due to mis-retransmission of the packet. The additional delay incurred by the mis-retransmission at time 128.5 [msec] becomes about 6 [msec] in this case.



(a) TCP Behavior after Mis-retransmission



(b) TCP Behavior without Mis-retransmission

Figure 3.1: Detailed behavior of TCP against the mis-retransmission

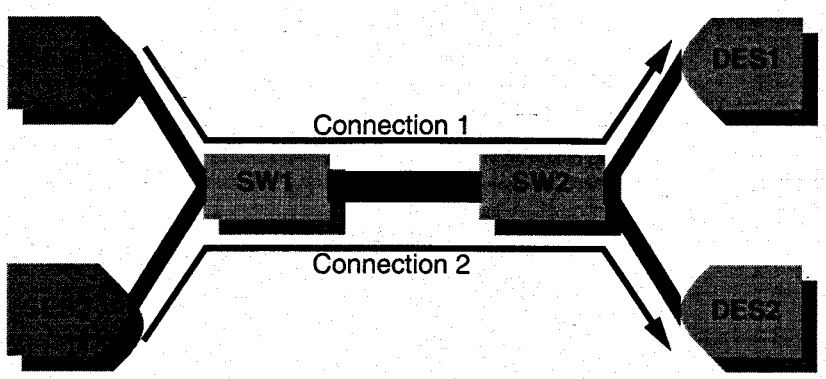
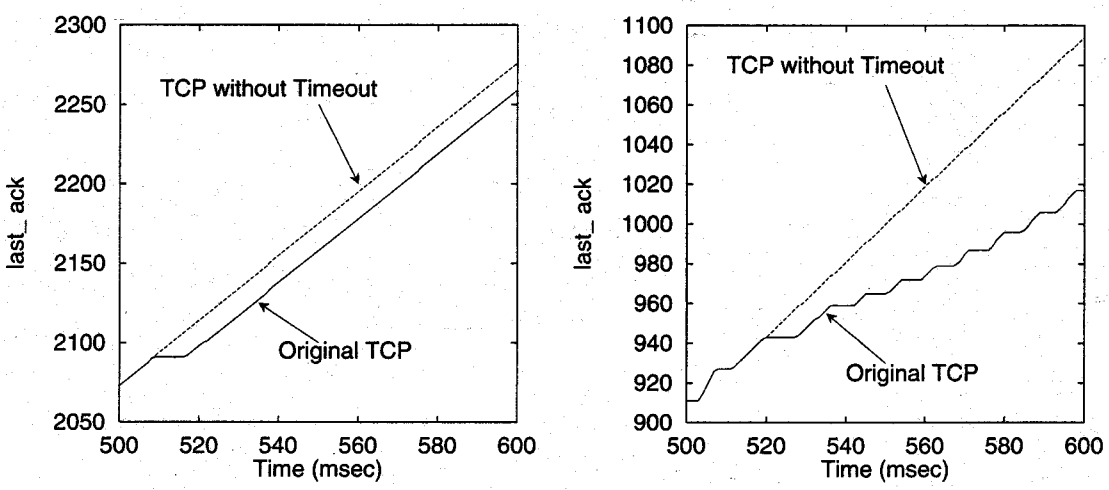


Figure 3.2: Simulation Model for Experiment 2



(a) $\tau = 0.01$ [msec]

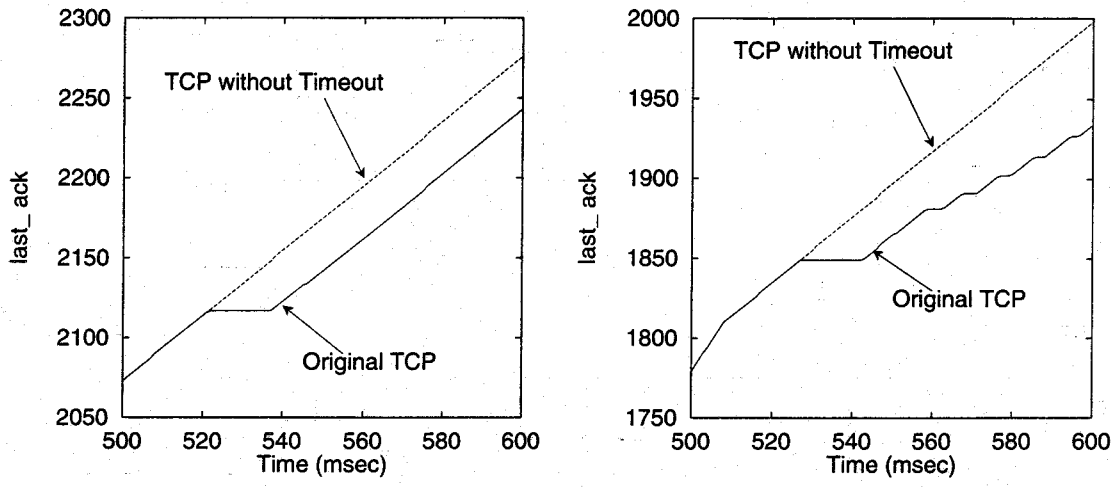
(b) $\tau = 4$ [msec]

Figure 3.3: Effect of Mis-retransmission: $W = 64$ [Kbyte]

Simulation Experiment 2

We next show other simulation results in which we use the network model depicted in Figure 3.2. The number of TCP connections is two, and the link speed is 150 Mbps. In the figure, SW stands for the intermediate ATM switch in which the ER mode is operated. In simulation, the connection 1 (SES1 → SW1 → SW2 → DES1) starts sending TCP packets, and at time 500 [msec] the connection 2 (SES2 → SW1 → SW2 → DES2) joins the network. When the bandwidth is assigned to each connection fairly by the ER mechanism, the bandwidth assigned to the connection 1 is suddenly decreased to a half after the connection 2 starts sending TCP packets. Then, the RTT value of connection 1 becomes larger than the RTO value and the mis-retransmission occurs.

Figure 3.3 shows the last_ack (the number of successfully acknowledged packets) of each connection as a function of time after the connection 2 starts the transmission.



(a) τ = 0.01 [msec] (b) τ = 4 [msec]

Figure 3.4: Effect of Mis-retransmission: W = 128 [Kbyte]

Here, the maximum window size of TCP is set to be 64 [Kbyte]. In Figure 3.3(a) where the propagation delay τ is set to be 0.01 msec, the solid line labeled as "Original TCP" shows how the mis-retransmission affects the throughput. After the simulation starts, the RTO value of connection 1 gradually becomes close to RTT. In this case, the RTT value is 3.86 msec, and the RTO value is 4.06 msec. The packet retransmission does not occur until time 500 msec (while not shown in the figure) since in our simulation the system is stable. At time 500 msec, the connection 2 is added. Then the RTT value of connection 1 is suddenly increased to 4.70 msec, but the RTO value is still 4.06 msec. The mis-retransmission then occurs at time 501 msec. As a result, last_ack does not increase around at time 520 msec. For comparison, the dotted line labeled as "TCP without Timeout" is also shown in the figure. It is the fictitious but ideal case that the mis-retransmission does not occur by setting the RTO value to be infinite.

Figure 3.3(b) compares two cases ("Original TCP" and "TCP without Timeout") by setting the propagation delay to be 4 msec. Throughput is further degraded due to repeatedly occurrences of mis-retransmissions when the propagation delay is set to be large. It is because when τ is large, the bandwidth-delay product of the connection becomes large. Then it takes more time until the maximum window size is fully utilized once the window size is throttled to be one packet. That is why the TCP sender cannot send some duration after the mis-retransmission.

It has been recognized that for large bandwidth-delay product networks, the window size should be enlarged to obtain high throughput. However, the larger window size further degrades the throughput due to mis-retransmission. It is because if the maximum window size W gets larger, the sender TCP should re-transmit more packets

when the time-out takes place. Figure 3.4 shows this case where the window size W is set to be 128 [Kbyte].

In the next Subsection, we will give the analysis to examine the degree of the performance degradation caused by mis-retransmissions.

3.2.3 Analysis

In what follows, we will present the analysis method to derive the additional delay caused by one mis-retransmission of the packet. We will use the following notations. The link speed is denoted as BW [Mbps]. The propagation delay between source and destination terminals is represented by τ [sec]. The maximum window size is W [Kbyte], and the packet size is m [Kbyte]. By assuming that each packet has a fixed length identical to m, the TCP sender can transmit W/m packets without acknowledgments. TCP's *current window size* (cwnd) at time t [sec] is represented by $cwnd(t)$ [packets]. We set $t = t_0$ at the time when the mis-retransmission occurs. We assume that RTTs of transmitted packets are initially settled down at rtt [sec].

Now we assume that $cwnd(t)$ reaches the maximum window size W before the mis-retransmission occurs. When the mis-retransmission occurs at time $t = t_0$, the TCP sender enters *slow start phase*, and the *slow start phase* ends when $cwnd(t)$ reaches $ssth$. It equals to $W/2$ in the current case. The TCP sender then enters the *congestion avoidance phase*. This phase terminates when the TCP sender begins to receive the duplicate ACKs at time $t = t_1$. Note that in the case of Figure 3.1(a), the mis-retransmission occurs at time $t = t_0 = 128.5$ [msec]. The TCP sender then enters the *slow start phase*. Its *congestion avoidance phase* starts and ends at time $t = 130.5$ [msec] and $t = t_1 = 132.5$ [msec], respectively.

The TCP sender receives the non-duplicate ACKs corresponding to all packets within the window until the mis-retransmission occurs, From time t_0 to t_1, W/m non-duplicate ACKs are received. Therefore, the window size $cwnd(t_1)$ at time t_1 can be calculated from Equation (1.1) as;

$$cwnd(t_1) = \sqrt{mW + \left(\frac{W}{2}\right)^2} \tag{3.8}$$

After that, the TCP sender again retransmits the packet for duplicate ACKs according to the fast retransmission algorithm. At the same time, the sender enters the fast recovery process. However, this process has no effect on our analysis because the window size after the fast retransmission process is not dependent on the process itself, but on the window size just before the fast retransmission process ($cwnd(t_1)$) according to the fast retransmission algorithm. Let us denote t_2 for the end of the fast retransmission process (136.5 [msec] in the case of Figure 3.1(a)). At time $t = t_2$, the fast retransmission

algorithm sets the window size to be a half of that before the fast retransmission, which is $cwnd(t_1)$. Therefore, the window size $cwnd(t_2)$ is updated as;

$$\begin{aligned} cwnd(t_2) &= \frac{cwnd(t_1)}{2} \\ &= \frac{1}{2}\sqrt{mW + \left(\frac{W}{2}\right)^2} \end{aligned} \tag{3.9}$$

For $t > t_2$, the window size $cwnd(t)$ increases since TCP gets into the Congestion Avoidance phase. If $cwnd(t_2)$ is enough large to fully utilize the link capacity, the additional delay D caused by the mis-retransmission is just equal to the initial value of the round trip time rtt. It corresponds to the time duration of fast retransmission process because it takes rtt to come back the ACKs corresponding to the fast retransmitted packets. (In Figure 3.1(a), it is from 132.5 [msec] to 136.5 [msec].) Therefore, we have a relation

$$D = rtt \tag{3.10}$$

On the contrary, if $cwnd(t_2)$ is too small to fully utilize the link capacity, the window size is not completely recovered before TCP enters the *congestion avoidance phase*. From Equation (1.1) and Figure 3.1(a), the time duration from the mis-retransmission occurs until $cwnd(t)$ reaches W is

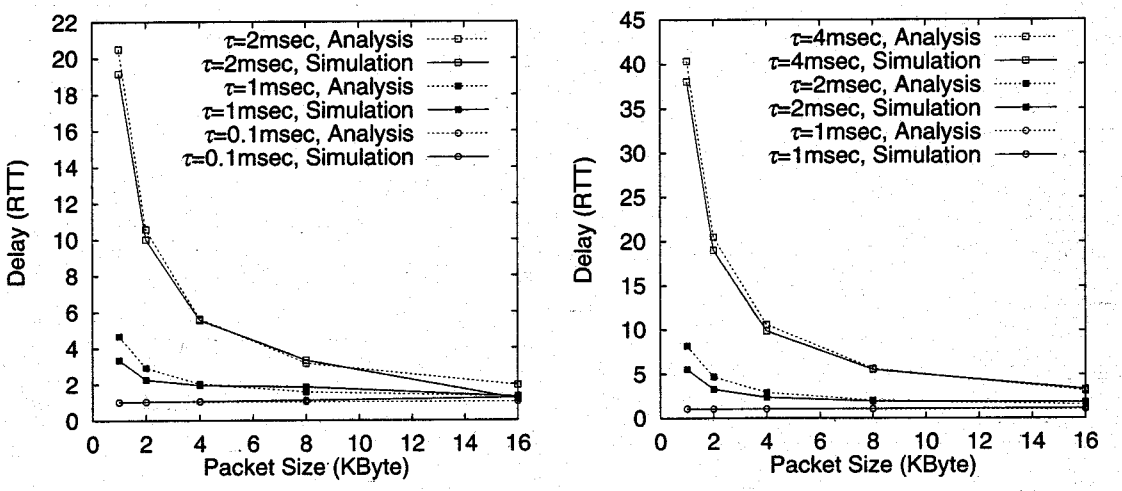
$$T = 2\,rtt + \frac{4W - \sqrt{4mW - W^2}}{4m} \tag{3.11}$$

The above Equation (3.11) can be obtained as follows. The first term ($2\,rtt$) corresponds to the time duration from the mis-retransmission occurs (t = 0 [sec]) until the fast retransmission process finishes at time $t = t_2$. (136.5 - 132.5 = 4.0 [msec] in the case of Figure 3.1(a)). The second term is the time duration from the fast retransmission process finishes at time $t = t_2$ until $cwnd(t)$ reaches W during the *congestion avoidance phase*. It is obtained from Equation (1.1).

The window size $cwnd(t)$ for the duration $t_2 \leq t \leq t_0 + T$ is calculated from Equation (1.1) as;

$$cwnd(t) = cwnd(t_2) + \frac{m^2}{rtt}t \tag{3.12}$$

Then, the total amount of data that the TCP sender can transmit during the time duration T can be calculated by integrating $cwnd(t)$ from time $t = t_0$ to $t = t_0 + T$. By defining



(a) W = 64 [Kbyte] (b) W = 128 [Kbyte]

Figure 3.5: Additional Delay D vs. Packet Size m

it by A, we have

$$\begin{aligned} A &= W + \int_{t_2}^{t_0+T} cwnd(t)dt \\ &= \frac{W(15W + 60m)}{32m} \end{aligned} \tag{3.13}$$

If the mis-retransmission had not occurred, the TCP sender could transmit the data at rate $R = W/rtt$ during the time duration $A/R = rtt \cdot (A/W)$. Therefore, the additional delay, D, incurred by the mis-retransmission is finally obtained as;

$$\begin{aligned} D &= T - rtt \cdot \frac{A}{W} \\ &= \frac{17W + 4m - 8\sqrt{4mW + W^2}}{32m} rtt \end{aligned} \tag{3.14}$$

3.2.4 Numerical Results and Discussions

Based on the analysis presented in the previous Subsection, we compare the additional delay D as a function of the packet size m in Figure 3.5. Three different values of the propagation delays are used; τ = 0.1, 1, and 2 [msec]. In Figures 3.5(a) and (b), the maximum window sizes W are set at 64 [Kbyte] and 128 [Kbyte], respectively. In the figures, the additional delay D is presented in unit of rtt. The values of rtt [sec] is dependent

on the bandwidth-delay product, and is given as;

$$rtt = \begin{cases} 2\tau + \frac{m}{BW}, & \text{if } W \leq 2\tau\, BW \\ \frac{W}{BW}, & \text{if } W > 2\tau\, BW. \end{cases} \tag{3.15}$$

In the figures, the simulation results are also presented in order to show accuracy of our analysis. The simulation model is just same as Simulation Experiment 1 used in Subsection 3.2.2, i.e., the single TCP connection uses 150 [Mbps] link. The simulation results were obtained by measuring the additional delay caused by intentionally generating mis-retransmission. From the figures, we can observe that the additional delay D becomes smaller as the packet size m is larger. It is because it takes more time for the window size to increase such that the link bandwidth can be fully utilized. This can be explained by Equation (3.14) which shows that the value of the packet size m significantly affects the additional delay D.

By comparing Figure 3.5(a) with Figure 3.5(b), we can observe that the larger W makes the additional delay D larger. As explained in Subsection 3.2.2, the TCP sender retransmits all packets within the window. Therefore, when W becomes larger, the TCP sender should retransmit more packets at the occurrence of the mis-retransmission. However, those are needless to be retransmitted. It is also shown in Equation (3.14) where the value of W affects the result.

From the above results, it is clear that the performance degradation caused by the mis-retransmission becomes remarkable when *bandwidth-delay product* gets larger. In the next Section, we will modify TCP to limit the performance degradation.

3.3 TCP Enhancement against Mis-retransmission

In this Section, we will propose a new mechanism of TCP to limit the performance degradation at a minimum caused by mis-retransmission.

3.3.1 A Proposed Mechanism

Figure 3.6 displays another view of the detailed behavior of TCP when mis-retransmission occurs. At time t_1, the packet transmitted at time t_0 is mis-retransmitted since the actual RTT ($realRTT$) of that packet is larger than the RTO value. The sender therefore decides that the packet has been lost. The ACK packet corresponding to the original packet is received at time t_2, and the packet corresponding to the mis-retransmitted packet is received at time t_3. Since the TCP sender cannot determine whether the ACK packet is responded for the original packet or for the retransmitted one, it decides that the time

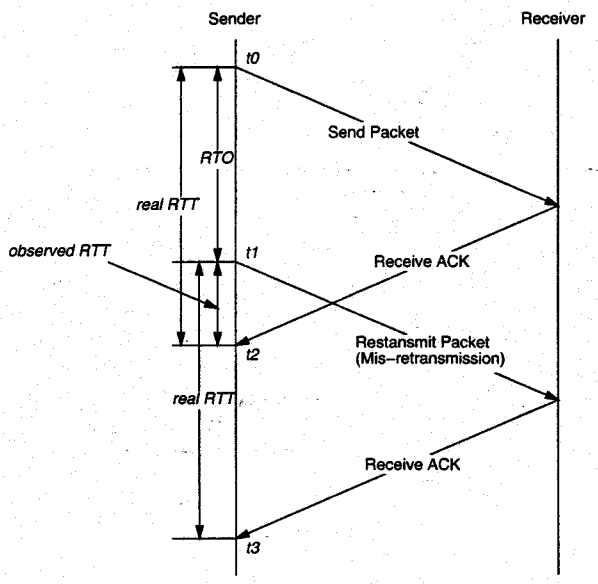


Figure 3.6: Another View of the Detailed Behavior of TCP after Mis-retransmission

duration $t_2 - t_1$ is the RTT ($observedRTT$) of the retransmitted packet. When RTTs of the TCP connection are stable, the value of $observedRTT$ is much smaller than the value of $realRTT$. That is, if the retransmission is mis-retransmission, the RTT of the mis-retransmitted packet becomes smaller than that of packets before the occurrence of the mis-retransmission.

Therefore, the sender in our enhanced TCP treats a retransmission as a mis-retransmission if,

$$(\text{RTT of retransmitted packet}) < k \cdot (\text{RTT before retransmission has occured}). \quad (3.16)$$

where k is a threshold between 0 and 1. When such a mis-retransmission is recognized by the sender, the value $ssth$ is restored to the one just before the occurrence of mis-retransmission. Then the TCP sender can open the window quickly. Furthermore, the retransmission process is not performed because it is not necessary at all, and the sender begins to transmit new packets.

In our proposed mechanism, mis-retransmissions are detected by comparing the RTT of the retransmitted packet with that before retransmission has occurred. Therefore, if the fluctuation of RTTs is large, the proposed method sometimes fails to detect the mis-retransmission. Figure 3.7 shows this case where the mis-retransmission cannot be detected. In this case, $realRTT$ of the packet suddenly becomes large so that $observedRTT$ also becomes too large to satisfy Equation (3.16). Next, we illustrate Figure 3.8 where the correct retransmission is detected as a mis-retransmission. In this case, the packet is truly lost, but the RTT of the retransmitted packet ($observedRTT$) is too

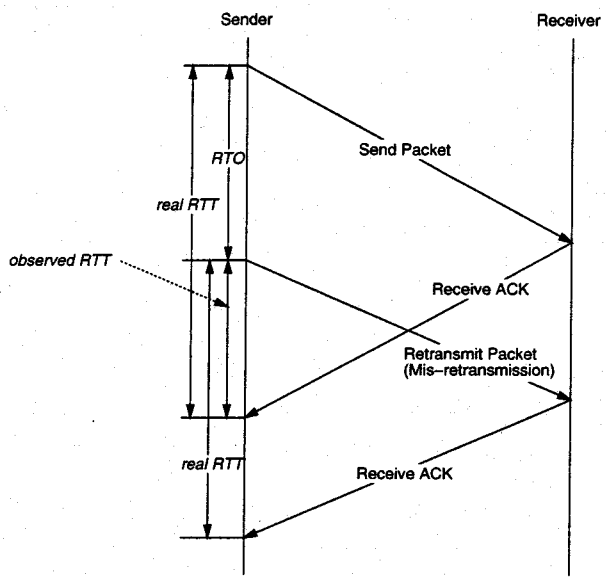


Figure 3.7: Case where Mis-retransmission is not Detected

small. Then Equation (3.16) becomes satisfied and the sender determines the retransmission is mis-retransmission. Therefore, an appropriate choice of the threshold k is a key issue to our TCP. If k is close to 0, it becomes difficult to detect mis-retransmissions as can be seen in Equation (3.16). On the other hand, if the large value of k is selected, the sender TCP may often detect the mis-retransmission even when the retransmission is actually required. In the below, we show the modified code in which we set $k = 0.5$. Its appropriateness will be validated in the next Subsection.

The modified TCP source code requires six new variables and additional 13 lines at the sender side, and no changes are necessary at the receiver side. The following code is to be added to the original TCP code of 4.4BSD-Lite distribution [76].

- `tcp_var.h`, line 105

```
/* variable for mis-retransmission detection */
tcp_seq to_snt;                 /* snd_nxt    */
tcp_seq to_seq;                 /* last_ack   */
u_long  to_cwnd;                /* cwnd       */
short   to_rtt;                 /* RTT        */
short   to_time;                /* time       */
int     to_ph;                  /* detection phase? */
```

- `tcp_timer.c`, line 206

```
/* record the variables at retransmission */
tp->to_cwnd = tp->snd_cwnd;
```

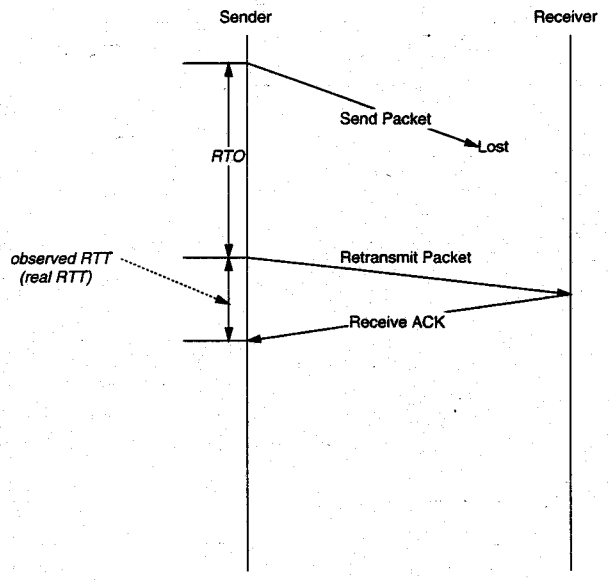


Figure 3.8: Case where Correct Retransmission is Recognized as Mis-retransmission

```
tp->to_snt  = tp->snd_nxt;
tp->to_seq  = tp->rcv_nxt;        /* last_ack */
tp->to_rtt  = tp->t_rtt;
tp->to_time = tcp_now;            /* time     */
tp->to_ph   = 1;
```

- `tcp_input.c`, line 407

```
/* mis-retransmission detection */
if ( tp->to_ph == 1 ) {
  tp->to_ph = 0;
  if ( (tcp_now - tp->to_now) < (1/2)*tp->to_rtt ) {
    tp->snd_ssthresh = tp->to_cwnd;
    tp->snd_nxt      = tp->to_snt;
  }
}
```

3.3.2 Simulation Results of the Proposed Method

In this Subsection, we evaluate the effectiveness of our proposed method described in Subsection 3.3.1 by a simulation technique. We consider that TCP is applied to the ER-based ABR network.

The first simulation model is identical to the one used in Simulation Experiment 1 presented in Subsection 3.2.1. That is, the single TCP connection uses the link with

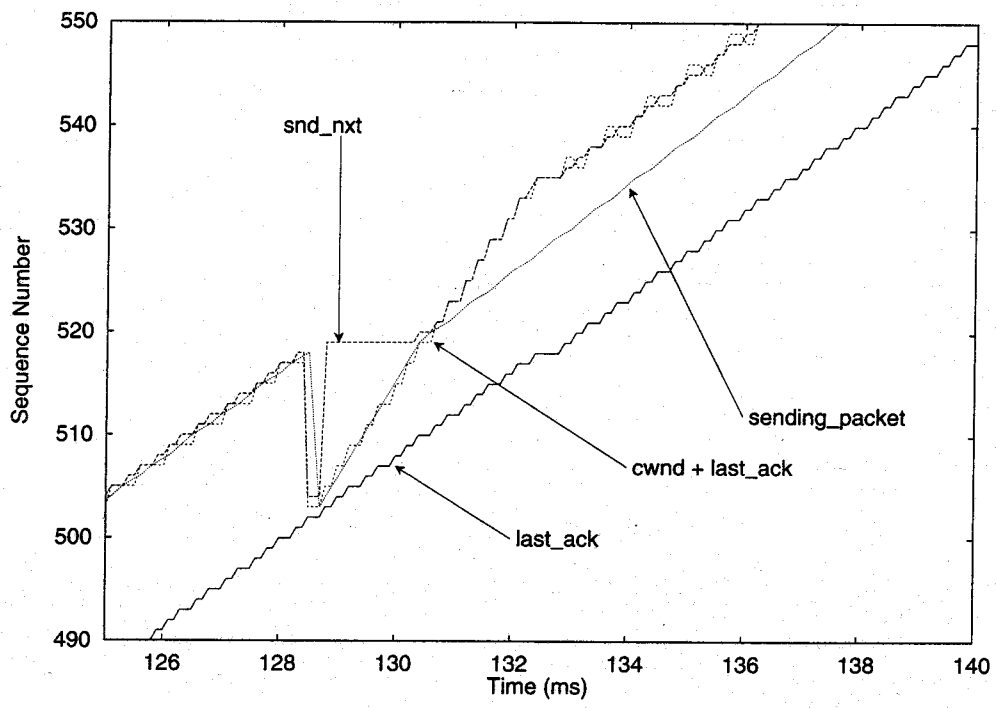
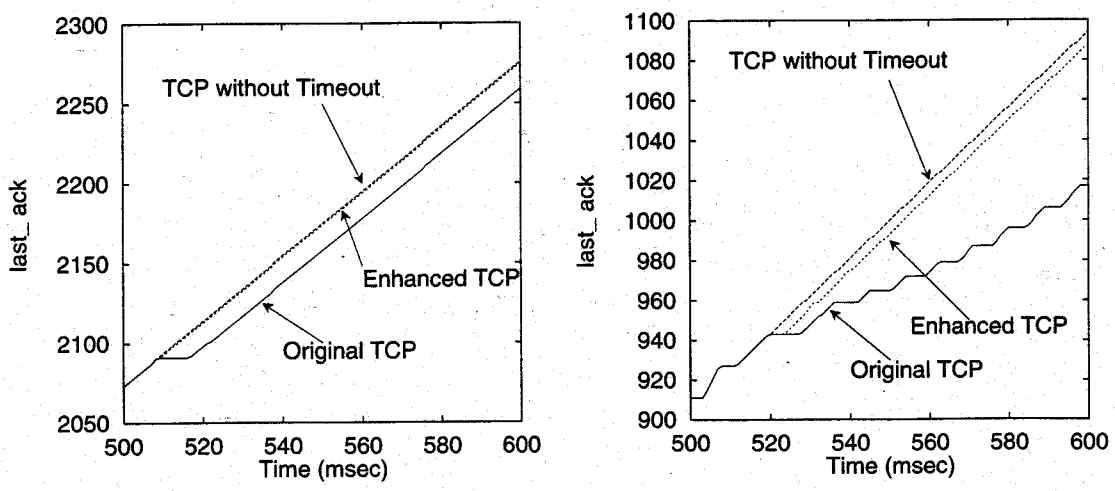


Figure 3.9: Enhanced TCP Behavior after Mis-retransmission

150 [Mbps] capacity. Figure 3.9 corresponds to Figures 3.1(a) (original TCP) and 3.1(b) (TCP with an infinite value of RTO to intentionally avoid the mis-retransmissions). We can see from this figure that although our enhanced TCP also mis-retransmits the packet at time 128.5 [msec], it quickly detects the mis-retransmission and the window size is recovered fast. As a result, the throughput is not much degraded when compared with the original TCP (Figure 3.1(a)).

We next use the simulation model depicted in Figure 3.2. As in Simulation Experiment 2, two connections share the link with 150 [Mbps] capacity. The connection 1 starts sending TCP packets at time 0 [msec], and at time 500 [msec] the connection 2 joins the network. The result is depicted in Figure 3.10(a) which shows `last_ack`'s of three versions of TCP; the original TCP, TCP with an infinite value of RTO, and our enhanced TCP. Here, the maximum window size W is set to be 64 [Kbyte] and the propagation delay τ is 0.01 [msec]. The figure clearly shows that the performance degradation of TCP after the mis-retransmission is very limited in our enhanced mechanism. However, when the propagation delay gets large, throughput degradation of our enhanced TCP becomes noticeable. It is shown in Figure 3.10(b) where the propagation delay is τ = 4 [msec]. It is because our enhanced mechanism cannot avoid the mis-retransmission itself so that the throughput degradation is not completely avoided. As described previously, since our enhanced mechanism needs the ACK packet corresponding to the original packet, it takes more time to detect mis-retransmission for the larger propagation



(a) τ = 0.01 [msec] (b) τ = 4 [msec]

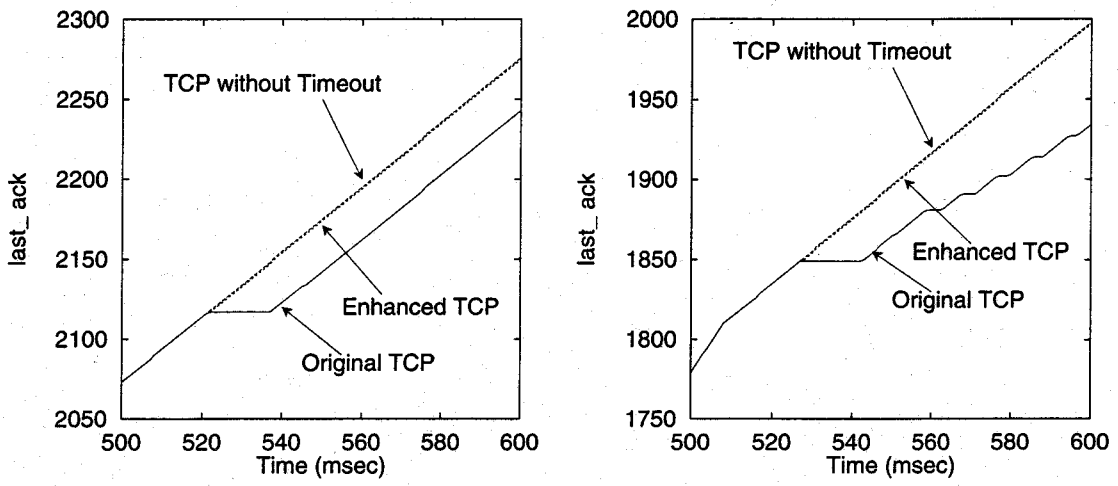
Figure 3.10: `last_ack` of Three Methods after Mis-retransmission: W = 64 [Kbyte]

delay. The cases of W = 128 [Kbyte] with 0.01 and 4 [msec] propagation delays are presented in Figure3.11. The same tendencies can also be observed in these figures. In the current simulation experiment, the connection 2 joins the network once. Of course, in the actual system, the addition/deletion of active connections occurs more frequently. In such a case, the performance degradation of the original TCP would become unacceptable while it is limited in our proposed TCP.

3.3.3 Robustness to Existing Network

In the previous Subsection, we have conducted the simulation experiments for TCP to be applied to the new data communication service; the ER-based ABR service class of ATM networks. We have shown that our proposed method works well in such an environment that RTTs do not fluctuate. On the other hand, in the current network system, RTTs vary dependent on the time. It is just a reason that the estimation method of RTT was introduced in TCP as having been described in Section 2. Accordingly, we need to confirm that our proposed method does not give an ill effect on the current system.

For this purpose, we modified the TCP code as presented in Subsection 4.1, and applied it to the Ethernet. In the experiment, file transfer was executed using `ftp` between two workstations connected to the Ethernet. Then, we measured the file transfer time. Figure 3.12 shows the file transfer times as a function of the file size. In the figure, the cases of original TCP and our enhanced TCP are compared. As shown in the figure, we can confirm that the performance is not degraded by applying our proposed mechanism to the existing network. That is, selection of the threshold value of k = 0.5 is suit-



(a) $\tau = 0.01$ [msec] (b) $\tau = 4$ [msec]

Figure 3.11: `last_ack` of Three Methods after Mis-retransmission: $W = 128$ [Kbyte]

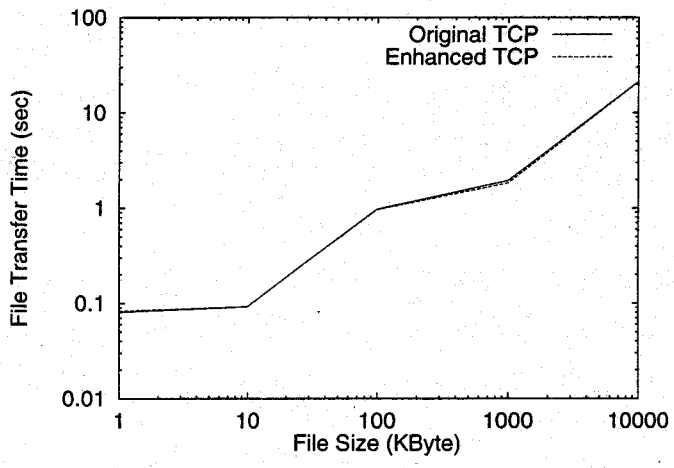


Figure 3.12: File Transfer Time vs. File Size

able to the Ethernet.

3.4 Conclusion

In this Chapter, we have focused on one serious problem of TCP's packet retransmission strategy. It arises when TCP is applied to the network where a fixed bandwidth can be assigned to each connection. An example is the ATM ABR service class where the ER mode mechanism is implemented. Through the simulation results, we have explained why mis-retransmission of the packet occurs, and have derived the analytic method to examine the degree of performance degradation caused by mis-retransmission. We

have also proposed an enhanced mechanism to improve the congestion control mechanism of TCP to detect mis-retransmission and to avoid performance degradation. Then, its effectiveness has been shown using the simulation experiments.

We have verified that our enhanced version of TCP does not give an ill effect on the existing Ethernet, but the experiments are limited. We need further experiments using real networks such as FDDI and, of course, the ER mode of the ATM ABR service when it is available.

Chapter 4

Performance Evaluation and Parameter Tuning of TCP over ATM Networks

In this Chapter, we investigate performance of TCP over ATM networks for data transmission. The main reason in this Chapter is to investigate the interaction of the congestion control mechanism of TCP, and ATM's rate control algorithms. For supporting TCP, we consider two service classes available at the ATM layer: UBR and ABR service classes (which we will refer as *TCP over UBR* and *TCP over ABR*). It has been shown in [77] that TCP over UBR cannot provide an effective resource usage. A main reason is that if at least one cell of multiple cells segmented from the upper-layer PDU (Protocol Data Unit: TCP/IP data packet in this case) is lost, the entire packet is treated to be lost since the ATM or AAL layers do not provide the cell retransmission function for error recovery. For supporting UBR, which has no congestion control mechanism, we consider EPD (Early Packet Discarding) [77] mechanism is considered (we will refer as *TCP over UBR with EPD* or simply *TCP over EPD*). EPD discards whole incoming cells constructing a packet when congestion occurs at the ATM switch to fully utilize outgoing link at the switch.

On the other hand, the ABR service class can provide more generic data transfer service at the expense of its control cost. A defect is to require several control parameters of the congestion control algorithm, and as has been shown in [47], an appropriate choice of control parameters is a key issue for the rate-based congestion control to work effectively [57, 61]. In this chapter, we find the appropriate parameter setting of ABR service class for supporting TCP, and show its effectiveness through some simulation experiments. To do that, we use two types of network model: singlehop network model and multihop network model for confirming the robustness of TCP over UBR/ABR network.

4.1 Congestion Control Methods for TCP over ATM Networks

In this Section, we present congestion control methods we will investigate in our simulation experiments. Definitions of control parameters and values used in our simulation are summarized in Table 4.1.

Plain UBR (Unspecified Bit Rate) Service Class protect[55]

No congestion control mechanism is provided at both of the switch and end systems. When the cell buffer at the switch becomes full, all incoming cells are discarded.

EPD (Early Packet Discard) Mechanism[77]

In the case of EPD, no end-to-end congestion control is employed at the ATM layer. Instead, EPD introduces a simple mechanism at the switch to reduce the packet loss. When at least one cell of the packet is lost, it would be impossible to reassemble the packet at the destination end system. Therefore, to avoid the waste of bandwidth by transferring incomplete packets, EPD discards all cells of the newly arriving packet when the queue length at the switch buffer exceeds some threshold value. By this mechanism, the packet, a part of which has already been accepted at the buffer, can be conveyed safely. For implementing this mechanism, EPD requires a VC table at each port. On arrival of the first cell of the packet, the corresponding entry of the VC table for that connection is marked if the queue length is longer than the threshold, or cleared otherwise. All subsequent cells of the packet are discarded if the VC table is marked. When the end-of-packet cell (EOP) arrives, the entry is cleared. Following [77], we set the threshold at half the buffer size in simulation.

EPD/A Mechanism [58]

In addition to the EPD mechanism, EPD/A is equipped with a per-VC accounting method, by which the number of cell in the switch buffer per connection is counted to keep the fair service among connections. That is, even if the queue length at the switch buffer, Q, exceeds the threshold of EPD, the arriving packet of connection i is accepted if the number of cells of connection i in the switch buffer, Q_i, satisfies the following equation;

$$Q_i < \frac{Q}{N_{VC}}$$

where N_{VC} is the number of active connections. By this mechanism, it is expected to alleviate the unfairness service among connections found in EPD.

Rate-based Congestion Control for ABR (Available Bit Rate) Service Class [55]

The ABR service class provides the rate-based congestion control. Each source end system periodically sends the forward RM (Resource Management) cell to the corresponding destination end system for every N_{RM} data cells sent. RM cells are used to notify congestion status of the network to the source via the destination. When the switch detects its congestion, it informs destination end systems of the congestion occurrence by marking the EFCI (Explicit Forward Congestion Indication) bit in the header of each forward RM cell. When the forward RM cell arrives at the destination end system, it is then sent back to the source as a backward RM cell to notify the congestion. In our simulation, the congestion occurrence and relief are recognized by the threshold value of the queue length at the buffer of Switch 1. This kind of the switch is a most basic one for the rate-based congestion control and sometimes called an EFCI switch or a binary switch. See [78] about another type of the switch.

Each source end system changes its cell emission rate, called ACR (Allowed Cell Rate), according to the congestion status notified by the backward RM cells. Only when it receives EFCI bit cleared backward RM cell, each source end system increases its ACR linearly as

$$ACR \leftarrow \min(ACR + AIR \times N_{RM}, PCR) \tag{4.1}$$

where PCR (Peak Cell Rate) is a maximum allowable rate and AIR (Additive Increase Rate) is a control parameter to determine the slope of rate increase. ACR is decreased when at least one of the following conditions is met.

(a) The source end system receives the EFCI bit marked backward RM cell. In this case, ACR is updated as

$$ACR \leftarrow \max(ACR - \frac{ACR \times N_{RM}}{RDF}, MCR) \tag{4.2}$$

where MCR is a minimum of ACR, i.e., a guaranteed cell rate for the connection. RDF (Rate Decrease Factor) is also a control parameter to determine the degree of rate decrease.

(b) Before receiving a backward RM cell, at least the number X_{RM} of forward RM cells have been sent since the receipt of the last backward RM cell. In this case, ACR is updated as

$$ACR \leftarrow \max(ACR - ACR \times XDF, MCR) \tag{4.3}$$

where XDF is the X_{RM} Decrease Factor.

(c) Before sending the forward RM cell, the following conditions are satisfied.

$$ACR > ICR$$

and

$$T > TOF \times N_{RM}$$

where ICR is the Initial Cell Rate, T is the time elapsed since the last forward RM cell transmission, and TOF is the Time Out Factor. In this case, ACR is decreased as

$$ACR \leftarrow \max(ACR - ACR \times T \times TDF, \quad ICR) \tag{4.4}$$

where TDF is the Timeout Decrease Factor.

At the connection setup, each source negotiates with the network to determine the control parameters ICR, X_{RM} and P-Vector. P-Vector is a set of control parameters predefined in [55], and among several control parameters included in P-Vector, our concern is $AIRF$ and $RDFF$ which control cell flow.

4.2 Singlehop Network Case

In this Section, we evaluate the performance of TCP over UBR, TCP over UBR with EPD (*TCP over EPD*), and TCP over ABR in terms of throughput and fairness among connections, using simple singlehop network model.

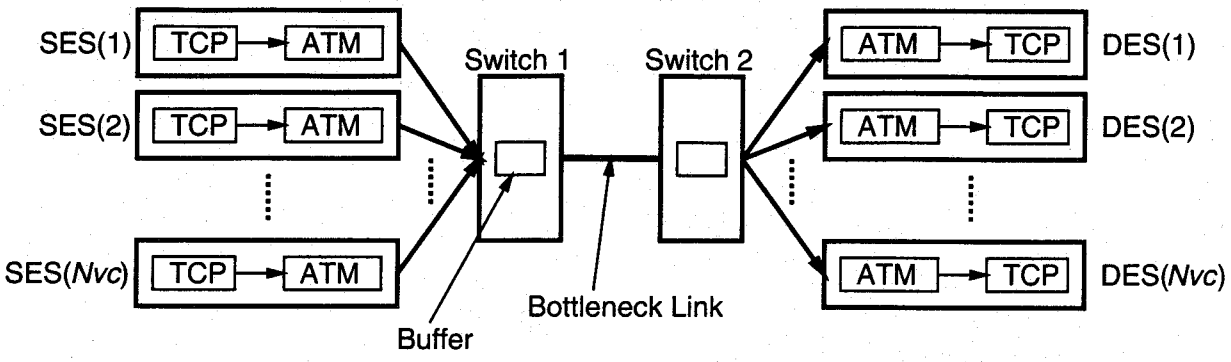


Figure 4.1: Singlehop Network Model

4.2.1 Network Model

Our simulation model for singlehop network is depicted in Figure 4.1. The model consists of homogeneous N_{VC} sources end systems (SES), corresponding N_{VC} destination end systems (DES), and a single bottleneck link shared by the number N_{VC} of active connections. At the output buffer of the left-hand side switch(Switch 1) the queue length is observed for congestion control in the cases of TCP over EPD and TCP over ABR. The i th source end system sends cells to the corresponding i th destination end system. Each connection is terminated in the TCP layer at both of source and destination end systems. We assume that each source always has TCP data packets to transmit. The propagation delays between source and destination end systems are set to be identical and denoted by τ [sec]. Note that the switch location is irrelevant in our simulation since congestion indication is always propagated via the destination in both of TCP and rate-based congestion control method adopted here. See below.

We model both Transport and ATM layers at each of source/destination end systems. The TCP layer is assumed to always have packets to transmit. The TCP packet is passed to the ATM layer, and segmented to cells to be transmitted to the network. An actual cell emission rate may be limited by the TCP window size, or by the rate of ABR. As soon as the destination end system receives cells, those cells are reassembled into the packet, and then passed to the TCP layer. Corrupted packets are discarded at the receiver side if at least one cell of the packet is lost at SW1. In simulation, a TCP packet size is fixed at 4,352 [Byte] (approximately 90 ATM cells). We further assume that IP is irrelevant and it only determines the size of the packet as in [77]. AAL5 is used for the AAL sublayer. In addition, processing time of TCP is neglected. This assumption may be unrealistic since the processing overhead affects performance considerably in some cases. Nevertheless, we assume zero processing times since it heavily depends on the processing power of machines, and it only introduces uncertainty for investigat-

ing our objective, i.e., potential capabilities of the above-mentioned congestion control mechanisms for supporting high speed data transfer.

4.2.2 Performance Comparisons

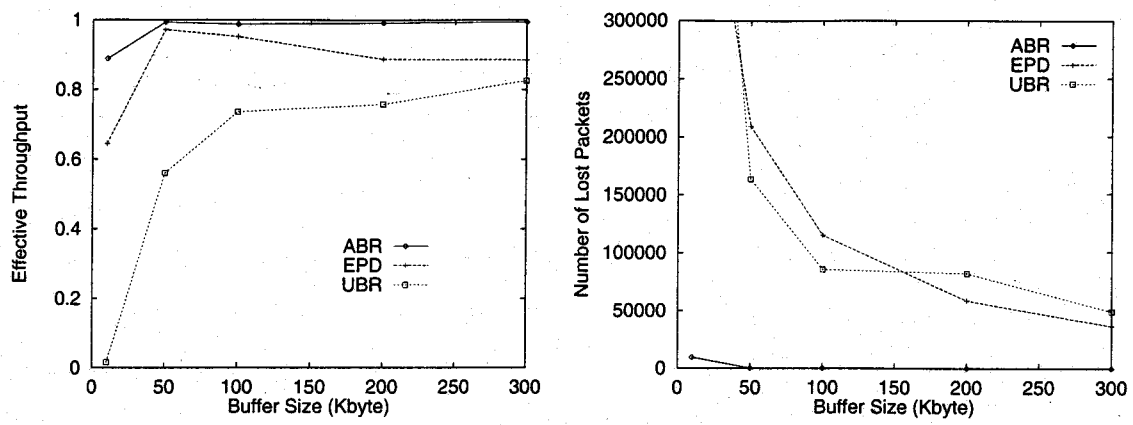
In this subsection, we provide comparative results for TCP over UBR, TCP over EPD and TCP over ABR in terms of the throughput and the fairness among connections. Table 4.1 summarizes the control and simulation parameters used in this subsection. The run time of each simulation experiment is 50 [sec], which is reasonably long to obtain steady state statistics. The start time of connections are staggered by 1 [msec]: i th connection starts its packet transmission at $(i-1)$ [msec] to avoid the traffic phase effect [79]. In this Subsection, control parameter values for the ABR service are fixed (see Table 4.1). Parameter tuning for improving the performance will be presented in the next Subsection.

Throughput Comparison

In what follows, we compare the performance of three methods mainly in terms of the effective throughput, which is defined as the total numbers of successfully transmitted packets (i.e., the sum of latest sequence number of the received acknowledgments at sources) in cells normalized by the link capacity.

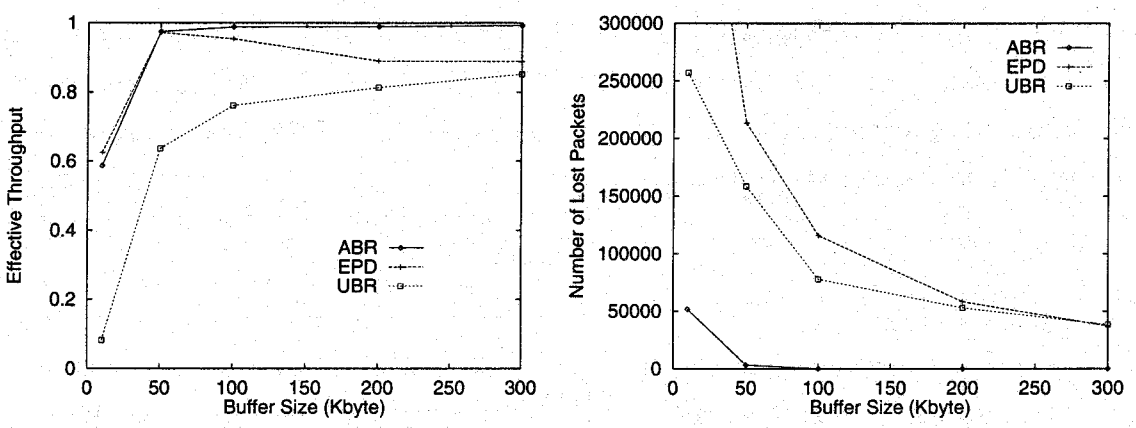
The first result in Figure 4.2 shows the effective throughput and the number of lost packets during the simulation run dependent on the buffer size. The propagation delay between sources and destinations, τ, is set at 0.01 [msec], approximately 2 [Km]. The number of connections, N_{VC}, is set to 10. We note that the similar result can be found in [77] for TCP over UBR and TCP over EPD cases. From the figure, we can see the effectiveness of TCP over ABR independently on the buffer size. Another observation is related to EPD. When the buffer size becomes large, the effective throughput of EPD is slightly decreased. It is because cells (and hence packets) are not likely to be lost at the large buffer. It leads to larger response times of acknowledgments, and henceforth larger round trip time estimation at the TCP layer [2]. Since the time out for detecting the packet loss is based on the round trip times of packets, it results in slow reaction against the packet losses once it happens.

Similar results can be observed when the propagation delay is changed to ten times larger than the previous case, i.e., 0.1 [msec] as shown in Figure 4.3. Other parameters are fixed in obtaining this figure. In ABR, the number of lost packets is increased, but the throughput degradation is not observed in the figure (except the small buffer size case) because the increase of packet losses can still be compensated by the packet re-



(a) Effective Throughput (b) Number of Lost Packets

Figure 4.2: Comparisons of Three Methods as a Function of Buffer Size: $\tau = 0.01$ [msec], $N_{VC} = 10$

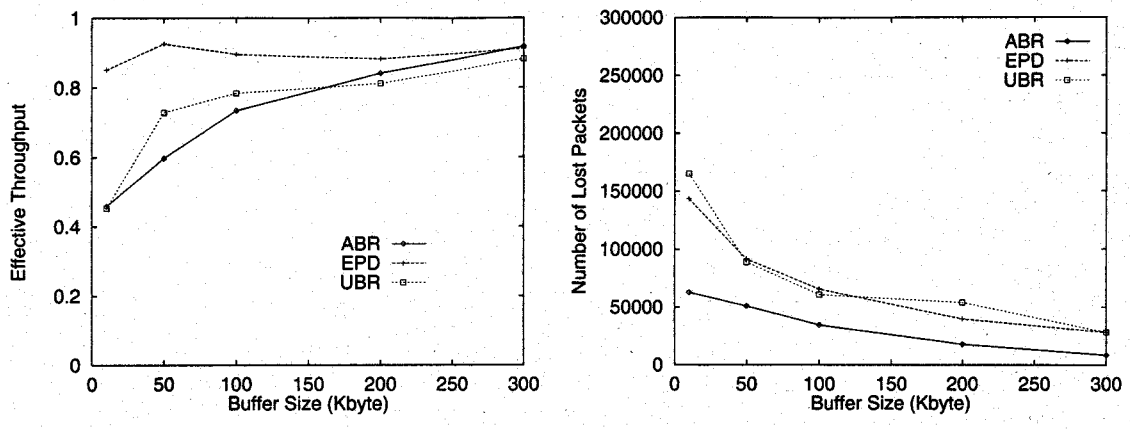


(a) Effective Throughput (b) Number of Lost Packets

Figure 4.3: Comparisons of Three Methods as a Function of Buffer Size: $\tau = 0.1$ [msec], $N_{VC} = 10$

transmissions at the TCP layer.

When the propagation delay becomes much larger, however, the order of the performance gain in three methods is reversed. The case of 1.0 [msec] propagation delay is shown in Figure 4.4. The throughput of EPD and UBR cases are slightly decreased, but throughput degradation of ABR is drastically except the case of the large buffer size. In UBR and EPD, the congestion management relies on the TCP layer. Thus, the effect of the propagation delay is not so large if our concern is the throughput. However, we should note here that the bandwidth-delay product [80] is not an issue in this parameter settings; the number of connections is 10 and the advertised window size is 64 [Kbyte].



(a) Effective Throughput (b) Number of Lost Packets

Figure 4.4: Comparisons of Three Methods as a Function of Buffer Size: τ = 1.0 [msec], N_{VC} = 10

A rough estimation shows that the bandwidth-delay product in the current case is

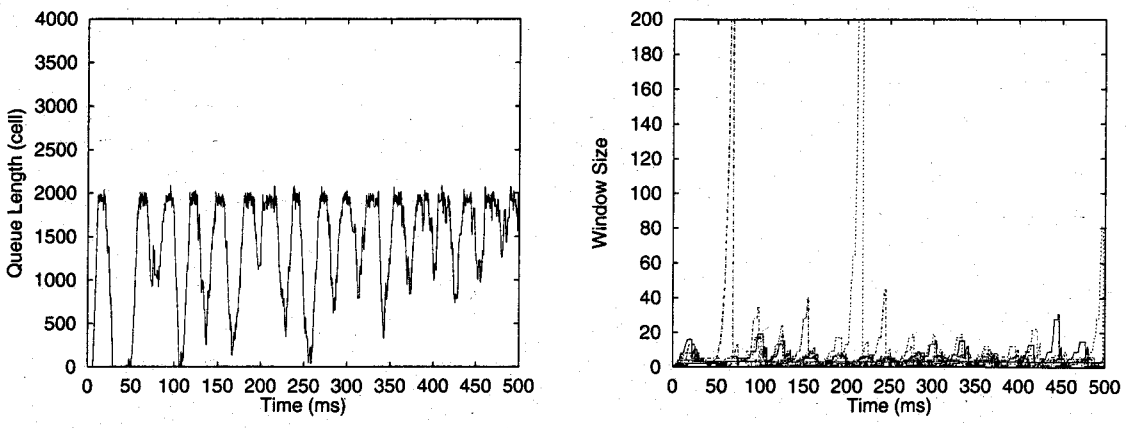
$$\begin{aligned} & \text{bandwidth} - \text{delay product} \\ = \ & \text{Link bandwidth} \times \text{two-way propagation delay} \\ = \ & 150 \text{ [Mbps]} \times (2 \times 1.0 \text{ [msec]}) \\ \approx \ & 300 \text{ [Kbit]} \end{aligned}$$

while in the current parameter setting, we have

$$\begin{aligned} \text{window size} \ \times \ & \text{the number of connections} \\ = \ & (64 \text{ [Kbyte]} \times 8) \times 10 \\ \approx \ & 5 \text{ [Mbit]} \end{aligned}$$

It is apparent, however, that the smaller number of connections and/or the larger propagation delay immediately would lead to the performance degradation even in UBR and ABR since the bandwidth-delay product becomes relatively large in such a case. In that case, we need to increase the window size to overcome the large bandwidth-delay product.

The longer propagation delay degrades performance of ABR, which can be observed by comparing Figure 4.2 through 4.4 although ABR achieves the smallest number of lost packets among three methods in all cases. In this Chapter, we use the binary switch for the rate-based congestion control. The long propagation delay causes control delay on congestion occurrence. Then, the rate decrease at the source end systems is also

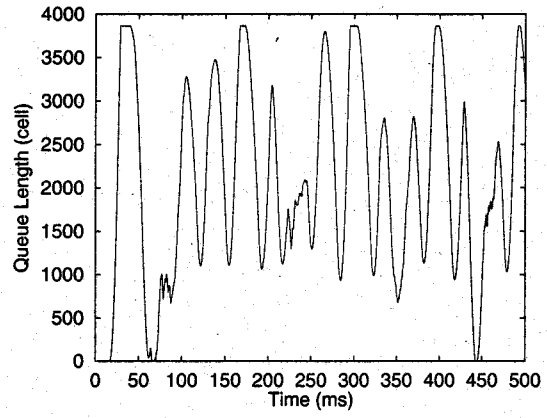


(a) Queue Length at the Buffer (b) TCP Window Size

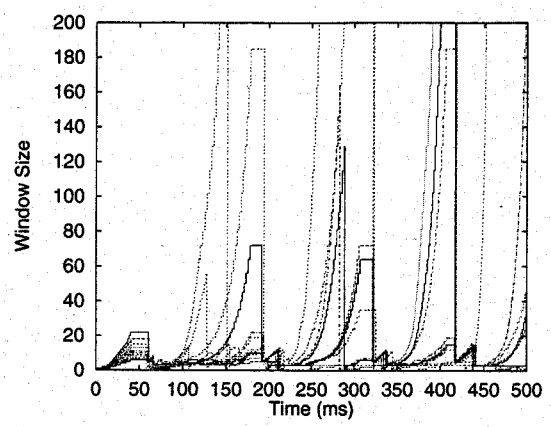
Figure 4.5: Time Dependent Behavior of TCP over EPD: $\tau = 1.0$ [msec], $N_{VC} = 10$, Buffer Size = 200 [Kbyte]

delayed. It introduces a long congestion period, which is one of main problems of the rate-based congestion control algorithm [81, 82]. When the packets are corrupted due to cell losses during the long congestion period, the time out at the TCP layer tends to occur frequently. Then, the TCP window size is decreased to one packet according to its congestion control strategy. If the cell losses take place continuously for some time, window sizes of all TCP connections are decreased. It is just the ABR case with inappropriate control parameter settings. In the case of EPD, on the other hand, such a case can be avoided. Once a packet is accepted at the buffer, the whole packet consisting of multiple cells can be transfered onto the output link. This mechanism can assure the randomness of packet dropping to some extent, and the packet loss is likely to be detected by the fast retransmit policy, which avoids the throughput degradation [36]. As a typical example, we show time-dependent behaviors of queue length at the switch buffer and the TCP window size during first 500 [msec] of simulation run for EPD and ABR in Figures 4.5 and 4.6, respectively. Figure 4.6 clearly demonstrates the "synchronization" of the TCP window size among connections in the case of ABR. It is noted, however, that the appropriate parameter setting makes it possible to avoid the synchronization and to improve performance of ABR, which will be discussed in the next Subsection. We further notice that while EPD can provide better throughput in this case, its randomness is not completely achieved; good throughput is obtained by a limited number of connections. A fairness aspect of both EPD and ABR will be presented in the next Subsection.

Last, we illustrate the effect of the number of connections on the effective throughput in Figure 4.7. The propagation delays, τ, are set to 0.01 [msec] (Figure 4.7(a)) and 1.0 [msec] (Figure 4.7(b)), and the buffer size is 300 [Kbyte]. When the propagation de-



(a) Queue Length at the Buffer



(b) TCP Window Size

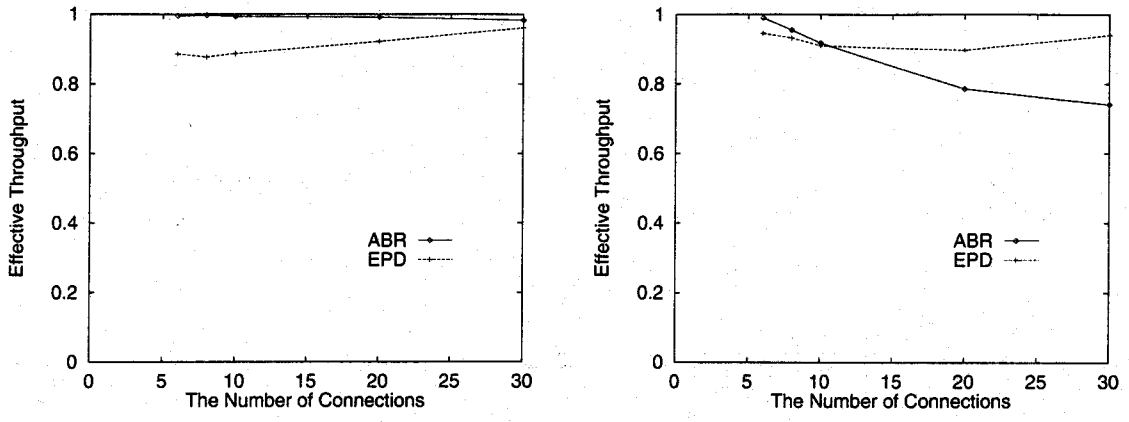
Figure 4.6: Time Dependent Behavior of TCP over ABR: τ = 1.0 [msec], N_{VC} = 10, Buffer Size = 200 [Kbyte]

lay is small (0.01 [msec]), the high throughput can be kept even with the larger number of connections in both of ABR and EPD. On the other hand, when the propagation delay becomes large, the throughput is degraded by the larger number of connections in ABR. It is another problem of ABR that the maximum queue length is increased almost linearly by the number of connections [81]. Then, cell losses and resulting packet losses are increased in the case of ABR. The parameter tuning of ABR in this case will be also discussed in the next Subsection.

Fairness Comparison

Since the fairness aspect in throughput has already been examined in [58], we limit our presentation to the results of different parameter sets from those in [58]. Note that those results will be used in the next Section for comparison purposes.

In Figure 4.8, we present the number of successfully transmitted packets for each connection as a function of time in ABR and EPD. The propagation delay and the buffer size are set to 0.01 [msec] and 300 [Kbyte], respectively. From this figure, we can see an excellent fairness property of ABR. An exception is observed during very early time of simulation since the connection with the larger number experiences the larger round trip time at the start up time. Then the increase of the TCP window size is delayed. On the other hand, in the case of EPD, the fairness cannot be provided; a close glance exhibits that only a limited number of connections transmit packets at time. See, for example, around 300 [msec] in Figure 4.8(a). One connection transmits about 40 packets during 10 [msec] while other connections does only a few packets. This phenomenon can be explained as follows. In EPD, when the queue length exceeds the threshold, the



(a) Effective Throughput for τ = 0.01 [msec] (b) Effective Throughput for τ = 1.0 [msec]

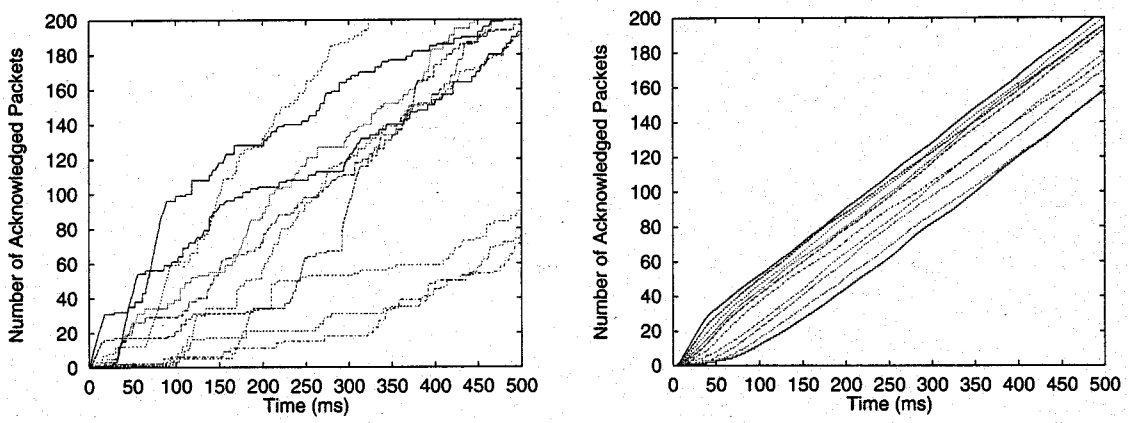
Figure 4.7: Comparisons of EPD and ABR Methods as a Function of the Number of Connections N_{VC}: Buffer Size = 300 [Kbyte].

newly arriving packets are blocked as long as the cells belonging to the same accepted packet arrive. After the EOP cell arrives, another newly arriving packet can be accepted. The problem is that the new packet (actually the first cell of the packet) is likely to come from the same connection. That is, the link tends to be possessed by the limited connections for some time.

The next figure shows the case of the larger propagation delay, 1.0 [msec] (Figure 4.9). While the degree of fairness among connections is not very different from the previous case in EPD, it is degraded in ABR. This indicates that the large propagation delay heavily affects the behavior of ABR in fairness as well as in throughput. The main reason is that the packet losses take place in this case, which leads to the unfairness treatment among connections even in the case of TCP over ABR. In Figure 4.9(b), we can find that all the curves become flat at around 230 [msec]. This is because, during 230 [msec], almost all connections' packets are lost during some time because of buffer overflow. Then, connections wait TCP timer expiration. After the TCP time out, connections begin to send packets at almost same time.

The hogging tendency in EPD becomes stronger by the smaller buffer size as shown in Figure 4.10. In this figure, the buffer size is changed to 10 Kbyte while the propagation delay is set to be again small (0.01 [msec]). As has been shown in Figure 4.4, the total throughput of EPD is larger than the one of ABR. However, it was achieved by the large throughput of limited connections. On the other hand, the difference of the throughput among connections is still fairly small in the case of ABR when compared with the case of EPD. We note here that the fairness among connections in ABR can be further improved, which will be presented in the next section.

Last, we provide another view of fairness. We examine the transfer time of the burst



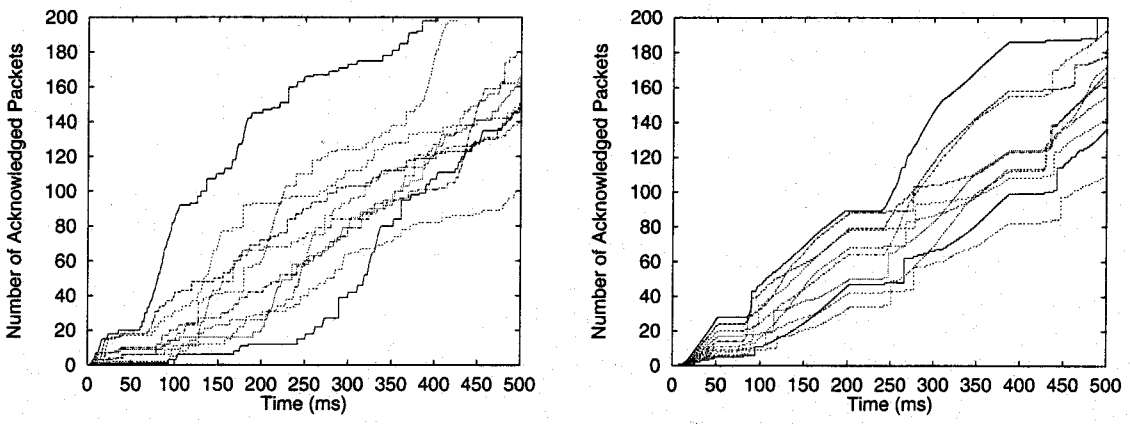
(a) EPD Case (b) ABR Case

Figure 4.8: Number of Successfully Transmitted Packets as a Function of Time: τ = 0.01 [msec], N_{VC} = 10, Buffer Size = 300 [Kbyte]

which consists of multiple packets. For this purpose, we add a new connection under the condition that the network is in steady state. By this experiment, we intend to demonstrate that the burst transmission delay is much affected by the start time of the new connection in the case of EPD while not in ABR. As typical examples (not exceptional cases), we present two cases where the new connection starts its packet transmission at 175 [msec] and 250 [msec]. Figures 4.11 and 4.12 show the cases of EPD and ABR, respectively. In the figures, we plot the number of successfully transmitted packets dependent on time. In ABR (Figure 4.12), we can observe that the burst delay is almost in proportional to the burst size in ABR. Exceptional are early packets, which is due to the slow start of the window size in TCP. On the other hand, EPD cannot provide such a property: it sometimes achieves very fast transmission (see first thirty packets in Figure 4.11(a)), or no transmission (Figure 4.11(b)).

4.2.3 Parameter Tuning for TCP over ABR

In the previous Subsection, we have shown that the ABR service can provide an effective use of the bandwidth compared with EPD and UBR except the cases of (1) the small buffer size, (2) the large number of connections and (3) the large propagation delay. As will be shown in the below, performance degradation of ABR is mostly caused by inappropriate setting of control parameters, but selection of appropriate parameters is not an easy task when we use a simulation technique through try and error. Therefore, we utilize the analytical result obtained in [47]. In [47], two conditions are imposed to achieve full link utilization and no cell losses at the buffer by assuming that all connections behave identically.



(a) EPD Case (b) ABR Case

Figure 4.9: Number of Successfully Transmitted Packets as a Function of Time: τ = 1.0 [msec], N_{VC} = 10, Buffer Size = 300 [Kbyte]

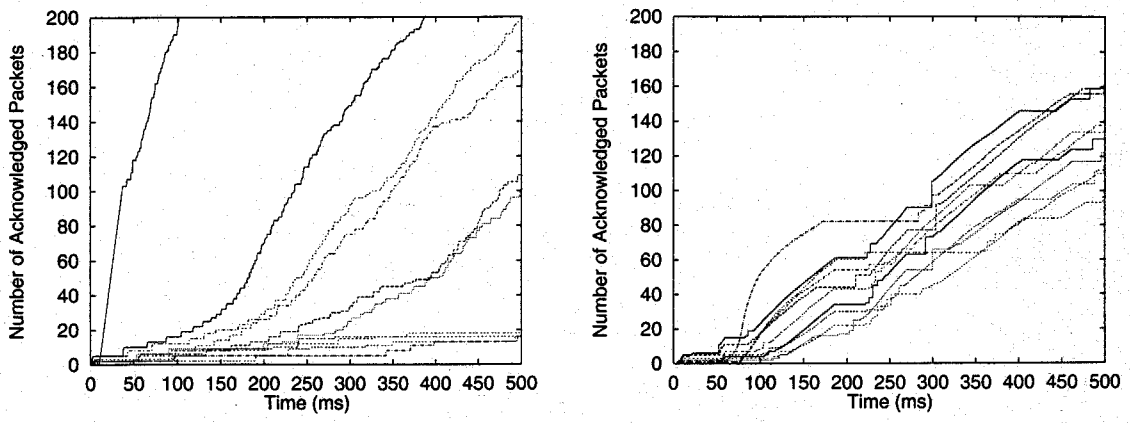
Condition 1: For the maximum queue length (denoted by Q_{max}) to be less than the buffer size, we should have

$$\begin{aligned} BL \geq Q_{max} \\ = \; & Q + RDF\sqrt{\frac{2AIR\,N_{VC}\,Q}{BW}} \\ & - N_{VC}\,RDF\,\log\left(1+\frac{2AIR\,Q}{BW}\right) \\ & + \tau\sqrt{2AIR\,BW\,Q} + AIR\,N_{VC}\,RDF\,\tau \\ & + \frac{AIR\,BW\,\tau^2}{2}. \end{aligned} \tag{4.5}$$

where Q represents a threshold value at the cell buffer. From the above equation, we can see that both of RDF and AIR should be small, but the reduce rate RDF has a larger effect than the increase rate AIR on the maximum queue length.

Condition 2: The minimum queue length should not reach zero for avoiding "under utilization" of the link. Since the required condition is more complicated, we omit the equation. The results in [47] show that RDF and AIR should be large to satisfy this condition.

There may exist many combinations of parameters that satisfy the above two conditions. In that case, we choose larger $AIRF$ and smaller $RDFF$, which leads to the more rapid rate increase and decrease. It means that it can achieve the good transient behavior.



(a) EPD Case (b) ABR Case

Figure 4.10: Number of Successfully Transmitted Packets as a Function of Time: τ = 0.01 [msec], N_{VC} = 10, Buffer Size = 10 [Kbyte]

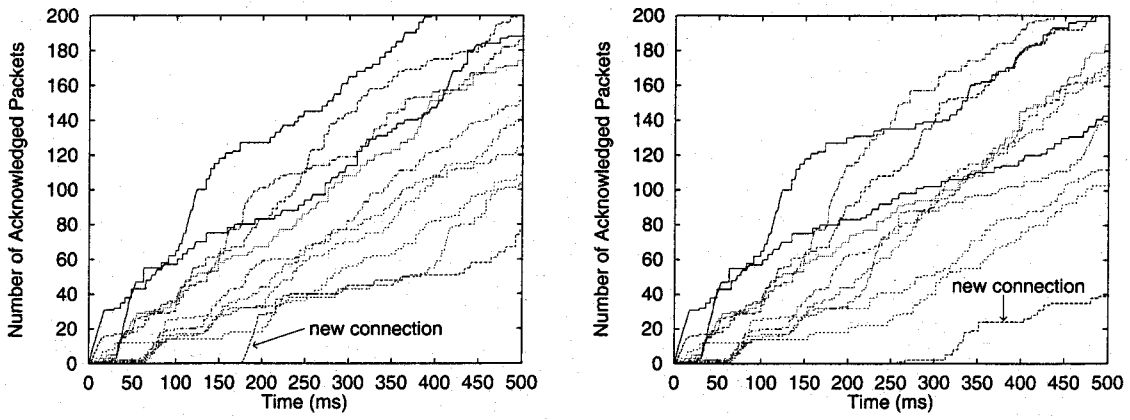
We have another possibility that there exist no control parameter set satisfying the above two conditions since the above two conditions imply opposite directions regarding *RDF* and *AIR*. Such a case tend to occur when the propagation delay is large as will be shown in the below.

Parameter Tuning for the Effective Throughput

In what follows, our investigation is devoted to improve the effective throughput. We then show that as a side effect, the fairness among connections can also be improved. We examine the values of increase and decrease rates of *ACR* under the condition that the propagation delay can be estimated a priori, which is implemented by the ABR service [55]. We have another control parameter Q, the threshold value of the queue length for congestion indication. While it is not determined in the negotiation process, we can determine it in the design phase of the ABR service based on two conditions presented in the above. However, our main concern in this Chapter is to tune the ABR control parameters. Therefore, in the below, we will seek two parameters, *AIRF* and *RDFF*, based on the above two conditions for given Q (a half of the buffer size) and other system parameters.

We begin with the small buffer size case for the small propagation delay. For given conditions that N_{VC} = 10, the Buffer Size = 10 [Kbyte], and the propagation delay τ = 0.01 [msec], we change parameters as

$$\begin{aligned} AIRF &: 1/64 \rightarrow 1/256 \\ RDFF &: 16 \rightarrow 2 \end{aligned}$$



(a) Start Time is 175 [msec].

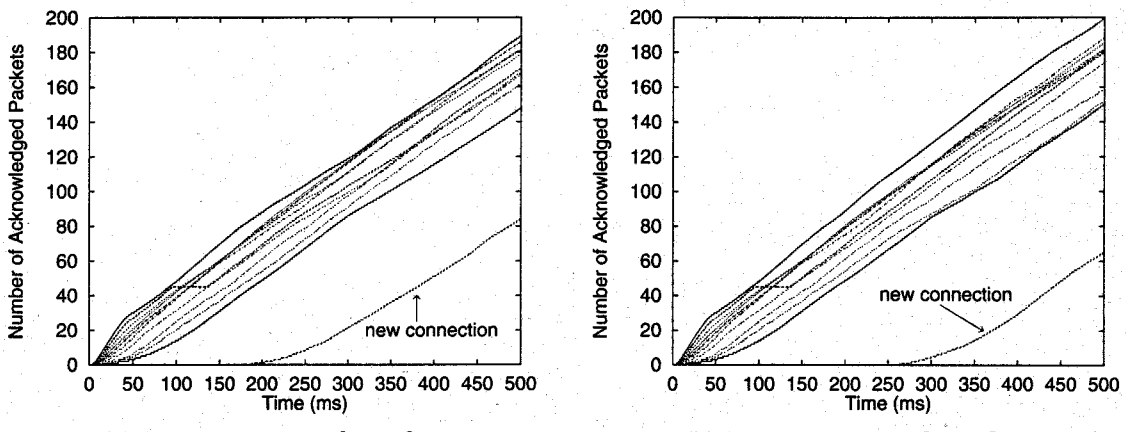
(b) Start Time is 250 [msec].

Figure 4.11: Number of Successfully Transmitted Packets as a Function of Time in EPD: τ = 0.01 [msec], N_{VC} = 10, Buffer Size = 300 [Kbyte]

where $AIRF = (AIR \times N_{RM}/PCR)$ controls the increase rate of ACR, and a smaller value of $AIRF$ leads to the moderate rate increase. On the other hand, $RDFF$ (= RDF/N_{RM}) controls the decrease rate of ACR, and a quick rate decrease is accomplished by a smaller value of $RDFF$.

Using the above parameters, we have confirmed that the throughput can be increased to 0.964 from 0.888 (see Figure 4.2(a)) as a result of limiting the number of lost packets (decreased to 51 from 9,598). A more notable result obtained in this experiment is that the fairness among connections is much improved. Compare Figure 4.13 with Figure 4.10(a) for EPD case and Figure 4.10(b) for ABR case without parameter tuning. We should note here that even with tuned parameters, we still have some lost packets. The difference is mainly due to the simplification of the analysis adopted in [47]. In [47], it is assumed that all connections behave identically, and that the rate changes is accounted in the unit of bits, that is, the fluid flow approximation is applied in the analysis. Therefore, we may need some margin when the results in [47] is applied in real situations.

The large propagation delay is next considered. We first use 300 [Kbyte] as large buffer size, and the propagation delay is set to 1.0 [msec]. In this case, we have the following parameters for satisfying two Conditions 1 and 2: $AIRF$ = 1/128 and $RDFF$ = 2. $AIRF$ is set to be large compared with the previous case. The throughput is increased from 0.9178 (Figure 4.4(a)) to 0.9588. The reason is that the corresponding number of lost packets is decreased to 0 from 8,464 (Figure 4.4(b)). Furthermore, the fairness, which has not been achieved in the case without tuning, can be improved. In Figure 4.14, we show the time dependent behavior of the queue length at the switch buffer and the number of successfully transmitted packets. These figures correspond to Figures 4.6(a) and 4.9(b) without parameter tuning.



(a) Start Time is 175 [msec].

(b) Start Time is 250 [msec].

Figure 4.12: Number of Successfully Transmitted Packets as a Function of Time in ABR: τ = 0.01 [msec], N_{VC} = 10, Buffer Size = 300 [Kbyte]

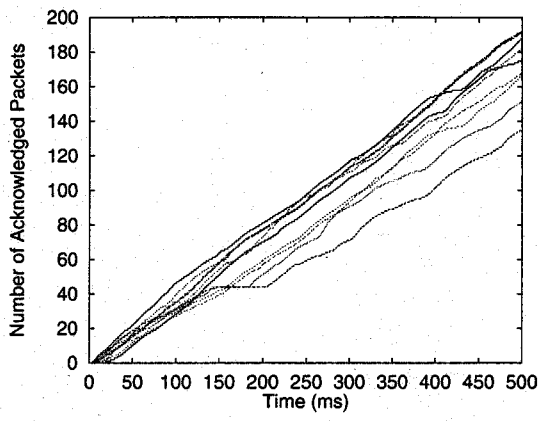
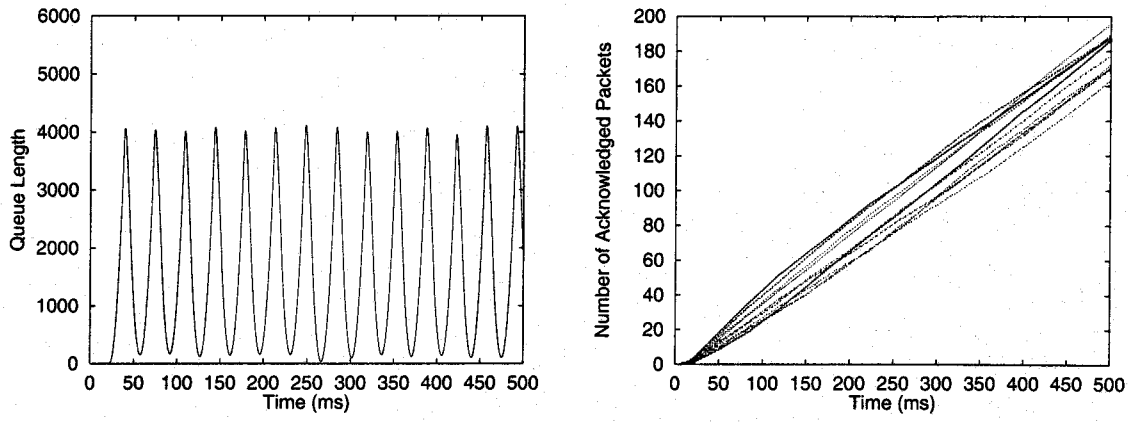


Figure 4.13: Number of Successfully Transmitted Packets as a Function of Time in ABR with Parameter Tuning: τ = 0.01 [msec], N_{VC} = 10, Buffer Size = 10 [Kbyte]

One problem is that we could not find the parameters which satisfy both conditions in the case of the small buffer size (10 [Kbyte]). Allowing "under utilization", but inhibiting the cell loss leads to parameters $AIRF = 1/256$ and $RDFF = 2$. The result was that while the number of lost packets is decreased to 1,163 from 62,953, the throughput was unexpectedly decreased to 0.435 from 0.458. (EPD achieved the throughput of 0.853 in Figure 4.4(a).) One favorable point is that it can achieve the fairness among connections as shown in Figure 4.15. In the figure, the case of EPD is also shown for comparison purpose. We next change the parameters so that the condition of no under utilization is fulfilled, but the cell loss is allowed. By this settings, we expect the ability of TCP packet retransmissions to improve the performance. If we use parameters $AIRF = 1/256$ and $RDFF = 4$, the throughput is improved to 0.542 at the expense of the fairness degree as shown in Figure 4.16.



(a) Queue Length at Switch Buffer

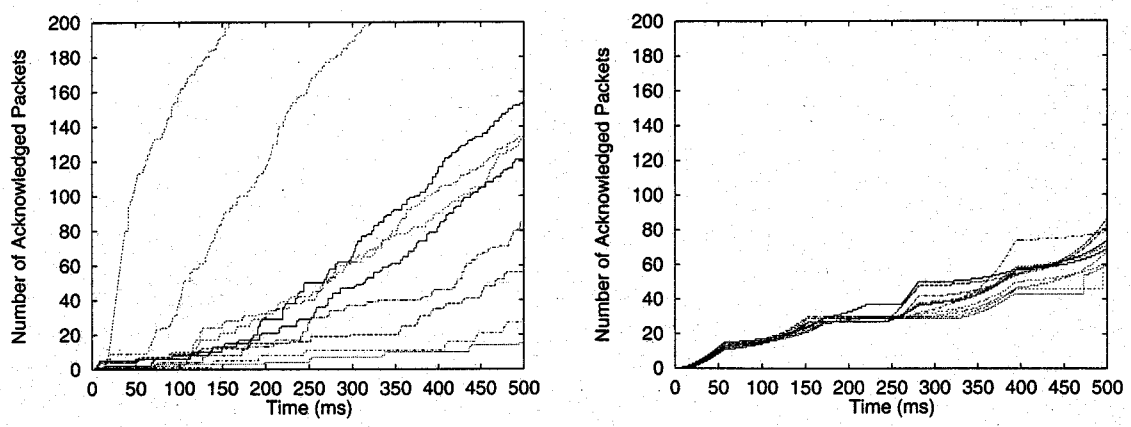
(b) The Number of Successfully Transmitted Packets

Figure 4.14: Time Dependent Behavior of TCP over ABR with Parameter Tuning: τ = 1.0 [msec], N_{VC} = 10, Buffer Size = 300 [Kbyte]

Last, in Figure 4.17, we show the comparison of three cases: EPD, ABR without tuning, ABR with tuning. The first two cases are drawn from Figure 4.4(a). In obtaining the last case, we change the control parameters so that high throughput can be obtained for given buffer size. Table 4.2 summarizes the parameter sets of ABR after tuning. For actual implementation, 10 [Kbyte] buffer seems to be rather small. However, we can see from the figure that EPD can achieve the high throughput at the expense of fairness among connections even with the small buffer size. On the other hand, while ABR cannot provide the better performance with the small buffer size, higher throughput can be attained by carefully choosing control parameters.

The last experiment in this Subsection is parameter tuning dependent on the number of connections N_{VC}. As has been shown in Figure 4.7(b), the throughput of ABR is degraded by the larger number of connections when we use fixed values of control parameters. However, it can be avoided. For example, when N_{VC} = 30, the parameters should be $AIRF = 1/128$ and $RDFF = 4$ for the case of 1.0 [msec] propagation delay and 300 [Kbyte] buffer size. That is, the total increase rate is reduced. Then, the throughput is dramatically improved from 0.740 (Figure 4.7(b)) to 0.995, which outperforms even the EPD case (0.931). The reason is that the number of lost packets is decreased from 34,862 to only 59 during simulation run. As an example behavior, we compare the queue length dependent on time in Figure 4.18. Recall that 300 [Kbyte] buffer corresponds to approximately 5,800 [cells]. By comparing Figures 4.18(a) and (b), we can observe that the cell loss and under utilization can be successfully avoided.

We last note that in the above results, the sensitivity on parameter selections has not been discussed. However, it is not a serious problem in our case since TCP can recover



(a) EPD Case (b) ABR Case

Figure 4.15: Number of Successfully Transmitted Packets as a Function of Time in EPD and ABR: τ = 1.0 [msec], N_{VC} = 10, Buffer Size = 10 [Kbyte], $AIRF$ = 1/256, and $RDFF$ = 2

the small number of cell losses and performance degradation can be sustained.

Case where a New Connection is Added

We next treat the case where the new connection is added to the network. The selection of the initial transmission rate, ICR, is important in the rate-based congestion control [83] since if ICR is set too large,the queue length at the buffer grows suddenly, which may result in cell losses. According to [47], we need to fulfill the following conditions.

$$ICR = \min\left(PCR, \frac{\max(BL - Q_{max}, 0)}{\tau + Q_{max}/BW}\right), \tag{4.6}$$

and

$$X_{RM} = \frac{ICR}{N_{RM}}\left(\tau + \frac{Q_{max}}{BW}\right), \tag{4.7}$$

where Q_{max} is given by the right-hand side of Equation (4.5). In the case of TCP over ABR, however, we have the slow start mechanism of TCP [2].Therefore, the selection of ICR does not contribute performance improvement of the new connection due to the small window size of TCP, Which limits the initial packet transmission rate.Then, the smaller value of ICR only degrade the performance. To see this, ICR for the new connection was varied and the transient behavior of the system was observed.

As an extreme example, the case of $ICR = PCR$ is plotted in Figure 4.19 to present ACR, the queue length, and the number of successfully transmitted packets dependent

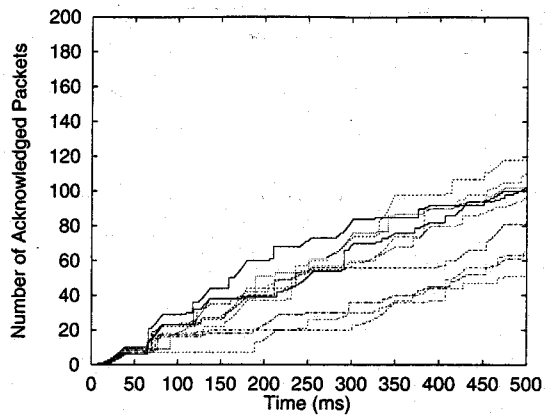


Figure 4.16: Number of Successfully Transmitted Packets as a Function of Time in ABR with Parameter Tuning: τ = 1.0 [msec], N_{VC} = 10, Buffer Size = 10 Kbyte, $AIRF$ = 1/256 and $RDFF$ = 4

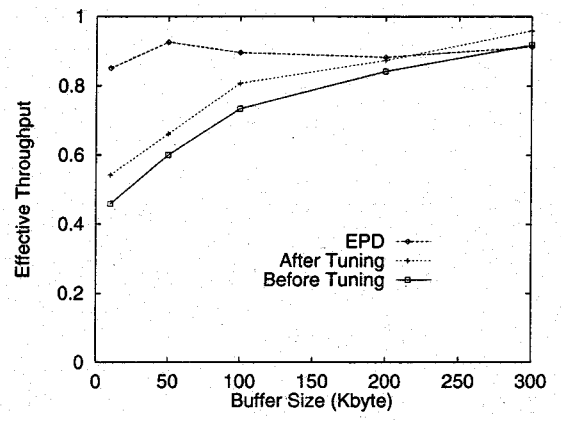
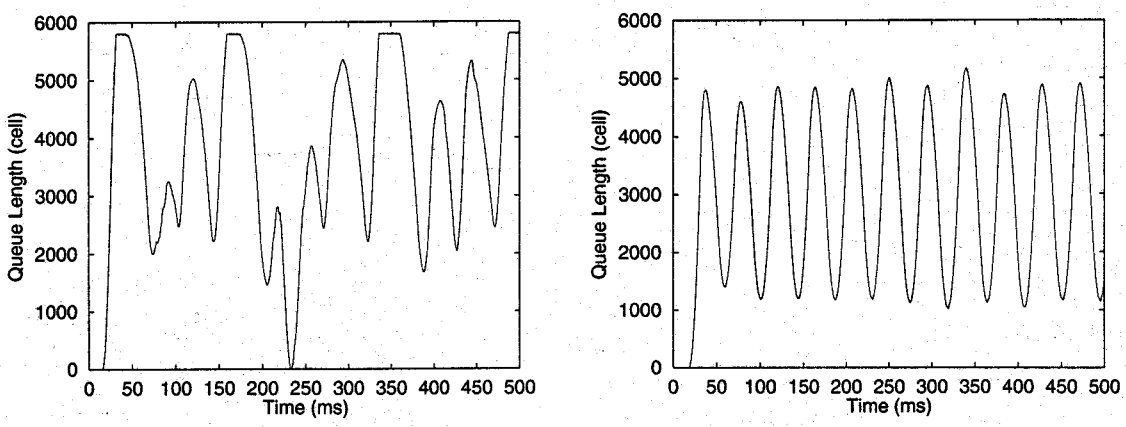


Figure 4.17: Comparisons of EPD, ABR with and without Parameter Tuning as a Function of Buffer Size: τ = 1.0 [msec], N_{VC} = 10

on time. In obtaining the figure, we used the following parameters: the propagation delay τ = 0.01 [msec], the number of connections N_{VC} = 10, and the buffer size = 300 [Kbyte]. The new connection begins its packet transmission at 250 [msec]. The figure shows that even when the new connection starts its transmission with ICR, a stable operation can be accomplished. Further, we can observe that the slopes of the number of successfully transmitted packets are almost same among the connections including the new added connection, which indicates that the fairness can be offered even to the new connection.

For comparison, time dependent behaviors of the new connection with several values of ICR are shown in Figure 4.20. In the figure, the behaviors of other existing connections are omitted. From the figure, we observe that the larger ICR is preferable for the new connection. When we consider a short-length burst, however, the delays are



(a) Without Parameter Tuning (b) With Parameter Tuning

Figure 4.18: Time Dependent Behavior of Queue Length with and without Parameter Tuning: τ = 1.0 [msec], N_{VC} = 30, Buffer Size = 300 [Kbyte].

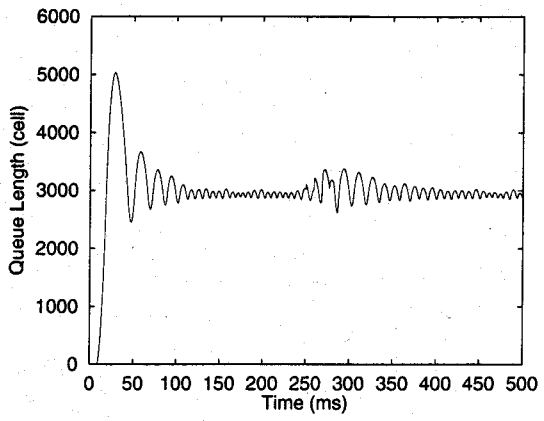
not different due to the slow start of TCP. In the figure, the short-length burst corresponds to less than twenty packets. Such a property is not expected since the larger ICR should lead to the smaller burst delays for the performance objective. Fortunately, we can choose ICR so that the cell loss does not occur at the switch buffer [47]. It means that we can expect the high speed transmission even for short burst by utilizing a more aggressive window increase algorithm instead of the slow start mechanism in TCP. Such a investigation should be a future research topic.

4.3 Multihop Network Case

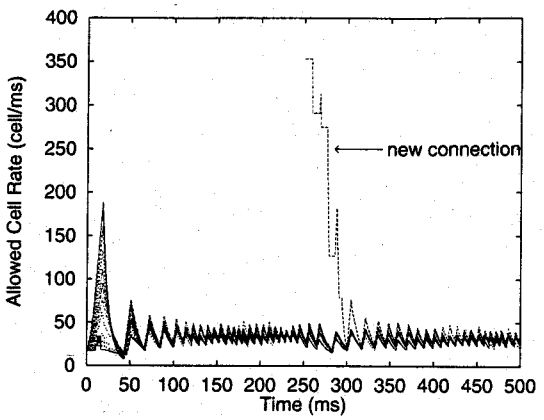
In this Section, we show the simulation results of the case of multihop network model, to investigate the robustness of TCP over UBR with EPD (TCP over UBR), TCP over EPD/A and TCP over ABR networks. We also consider the TCP over ABR with an EPD enhancement where an EPD mechanism is incorporated into ABR.

4.3.1 Network Model

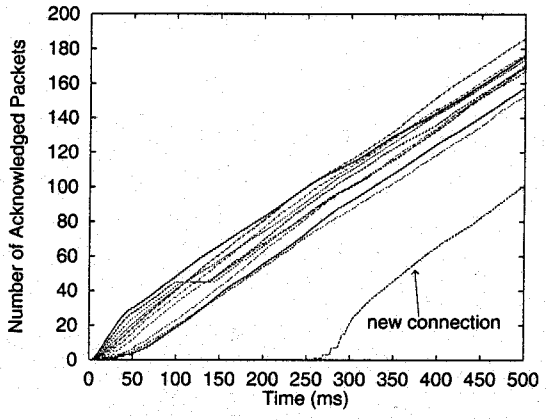
The simulation model for multihop network is depicted in Fig. 4.21. The model consists of source end systems (SES), destination end systems (DES), six switches (SW1~SW6) and links. The connection of VC i is established from SES i to DES i. Every connection passes through the bottleneck switch, SW5, where ten connections share the output link of the switch, SW5. Using this model, we study the throughput and fairness among connections, traversing different numbers of links. We assume the output-buffered switch,



(a) Queue Length at Switch Buffer



(b) ACR



(c) The Number of Successfully Transmitted Packets

Figure 4.19: Time Dependent Behavior with $ICR = PCR$ for the New Connection: $\tau = 0.01$ [msec], $N_{VC} = 10$, Buffer Size = 300 [Kbyte]

and the bandwidth of each link is assumed to be 156 [Mbps] (325.2 [cell/msec]). The propagation delay from each of switches to SES/DES is fixed at 0.005 [msec], corresponding to approximately 1 [Km] long. By setting the propagation delays between switches to be identical, we denote τ as the propagation delay between farthest switches (SW1 and SW6). In simulation, τ is set to be 0.01 [msec] (approximately 2 [Km] long) or 1.0 [msec] (approximately 200 [Km]) by considering LAN and WAN environments, respectively.

4.3.2 Performance Comparisons

In this Subsection, we provide comparative results for TCP over EPD(/A) and TCP over ABR in terms of the throughput and the fairness among connections. Unless otherwise stated, the control and simulation parameters follow that of the singlehop network case

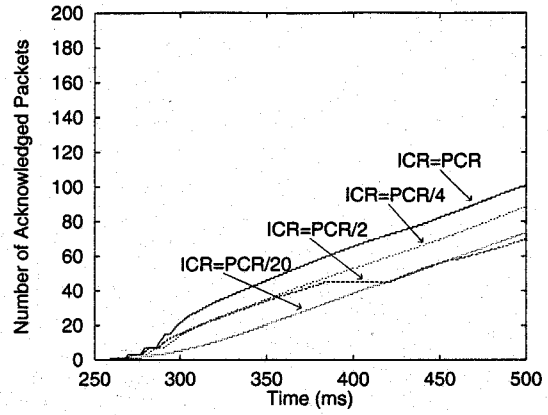


Figure 4.20: Number of Successfully Transmitted Packets as a Function of Time in ABR with Various ICR's: τ = 1.0 [msec], N_{VC} = 10, buffer size = 300 [Kbyte]

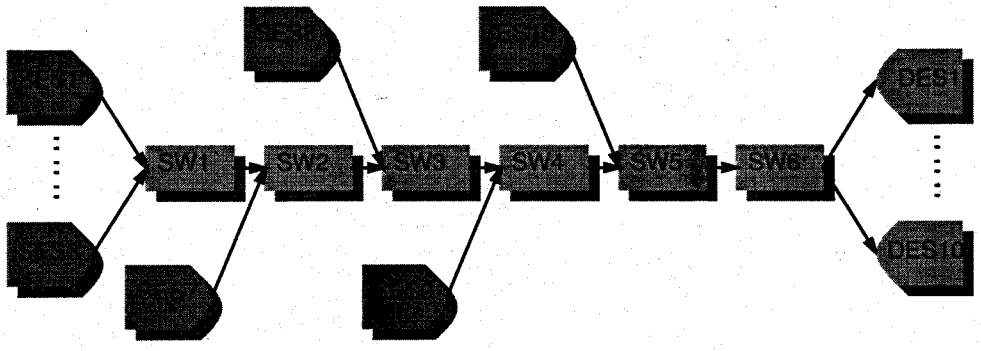
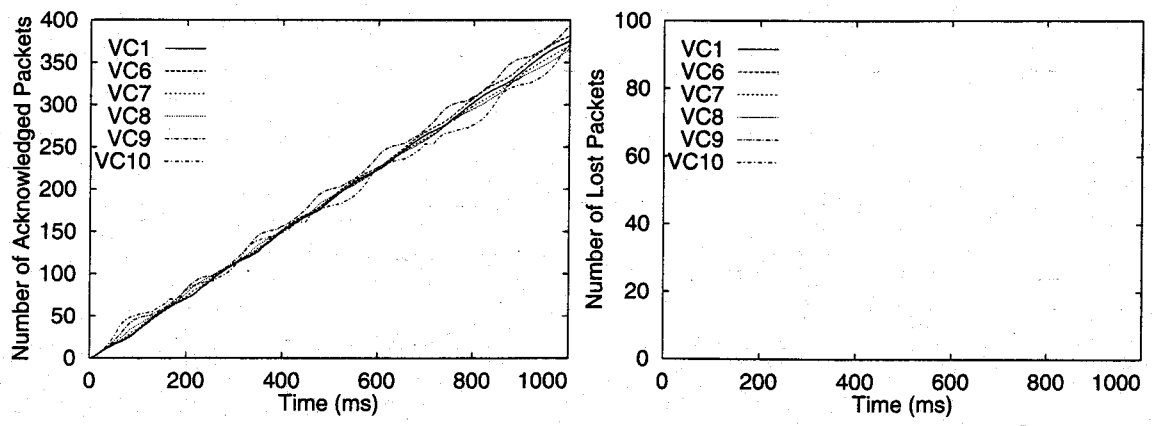


Figure 4.21: Multihop Network Model

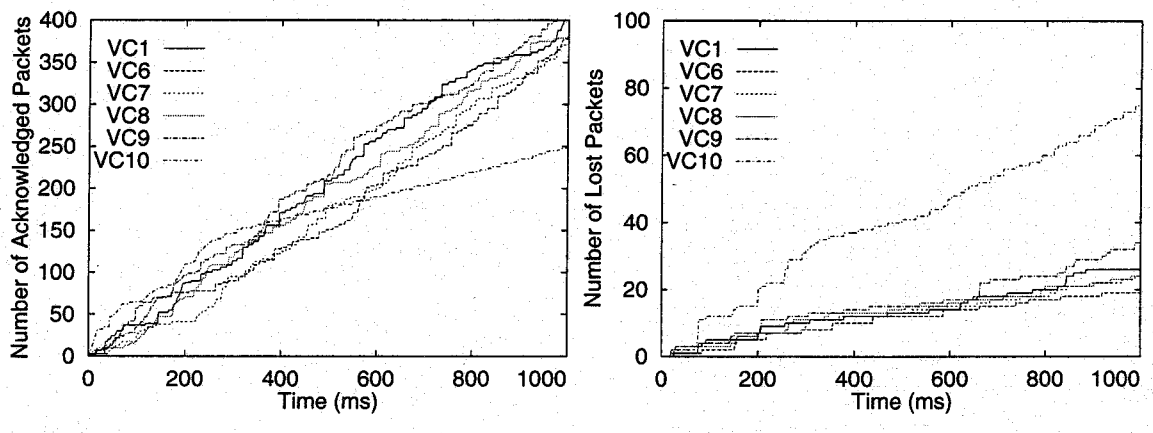
(summarized in Table 4.1). Figure 4.22 shows the case of TCP over ABR, plotting the number of successfully transmitted packets and the number of lost packets as a function of time. In the figure, results of connections VC1, VC6, VC7, VC8, VC9 and VC10, are plotted (results of VC2-VC5 are omitted because those behave like VC1 or VC6). We set RIF to 1/256 and RDF to 1/32 in this case based on the analytic result in Subsection 4.2.3 where the single bottleneck link is considered. The propagation delay, τ, is set to be 0.01 [msec]. The buffer size of switches is 300 [Kbyte], approximately 5,700 [cells]. From this figure, we can observe an excellent fairness property of TCP over ABR. In the figure, the behavior of first 1 sec during a simulation run is plotted to illustrate the transient behavior and the fairness among connections. The total throughput of all connections is 0.967. Noting that the overhead due to RM cells (one RM cell for 31 data cells) is introduced in the rate-based congestion control, the link utilization of output link at SW5 is almost full.

The next figure, Fig. 4.23, shows the case of TCP over EPD. The total throughput



(a) Successfully Transmitted Packets (b) Lost Packets

Figure 4.22: TCP over ABR: τ = 0.01 [msec], Buffer Size = 300 [Kbyte]

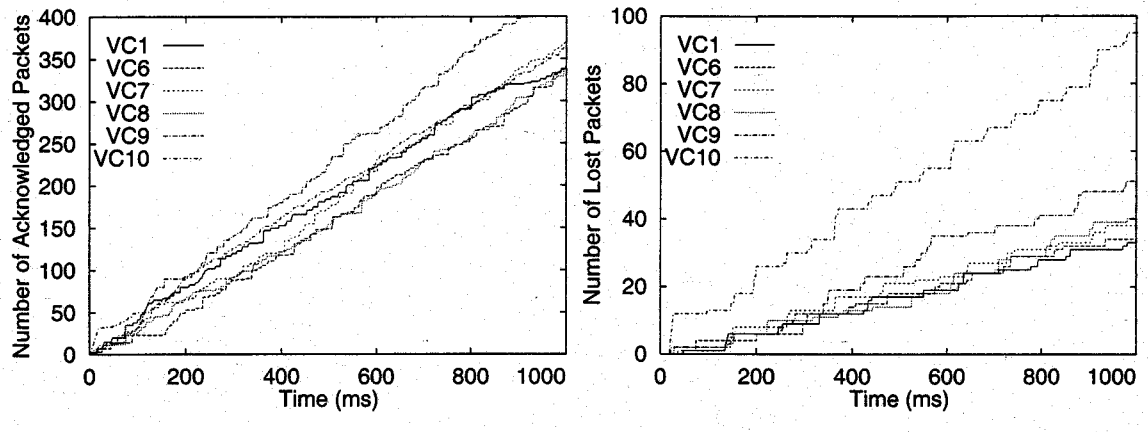


(a) Successfully Transmitted Packets (b) Lost Packets

Figure 4.23: TCP over EPD: τ = 0.01 [msec], Buffer Size = 300 [Kbyte]

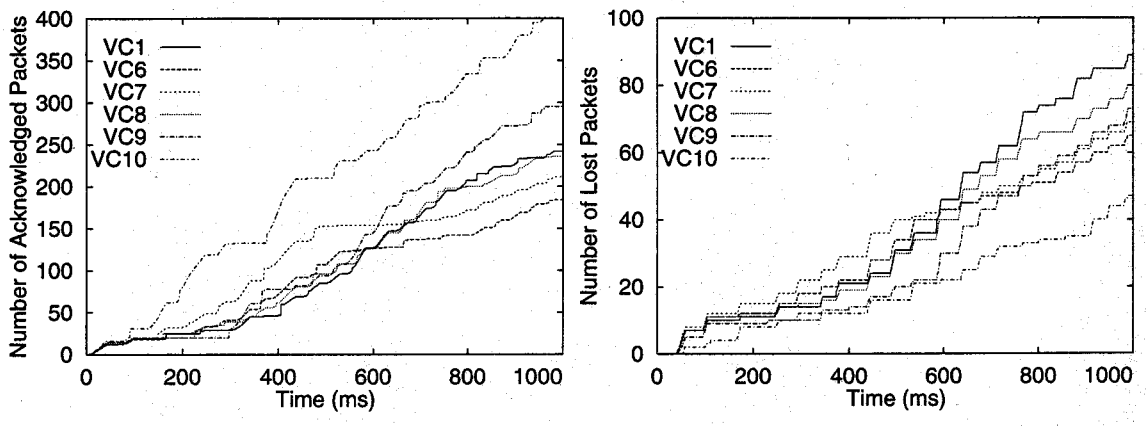
becomes 0.934, slightly smaller than that of TCP over ABR (0.967) because of lost packets and resulting packet retransmissions. Furthermore, it shows considerable unfairness among connections. VC9 and VC10, the connections with small propagation delay, lose more packets than VC1 with larger propagation delays. It is because for the smaller propagation delay of connections, the source can detect a packet loss faster, and retransmit packets by fast retransmission [10]. Then the source can send more packets, which results in an increase of lost packets, and the obtained throughput for each connection is much different with each other.

Figure 4.24 shows that the fairness of TCP over EPD can be slightly improved among connections by providing a per-VC accounting mechanism with EPD (TCP over EPD/A). A notable point is that a misbehaved connection (VC10) observed in Fig. 4.23(a) disappears. However, the throughput is decreased to 0.898 from 0.93. Further, the fairness



(a) Successfully Transmitted Packets (b) Lost Packets

Figure 4.24: TCP over EPD/A: τ = 0.01 [msec], Buffer Size =300 [Kbyte]

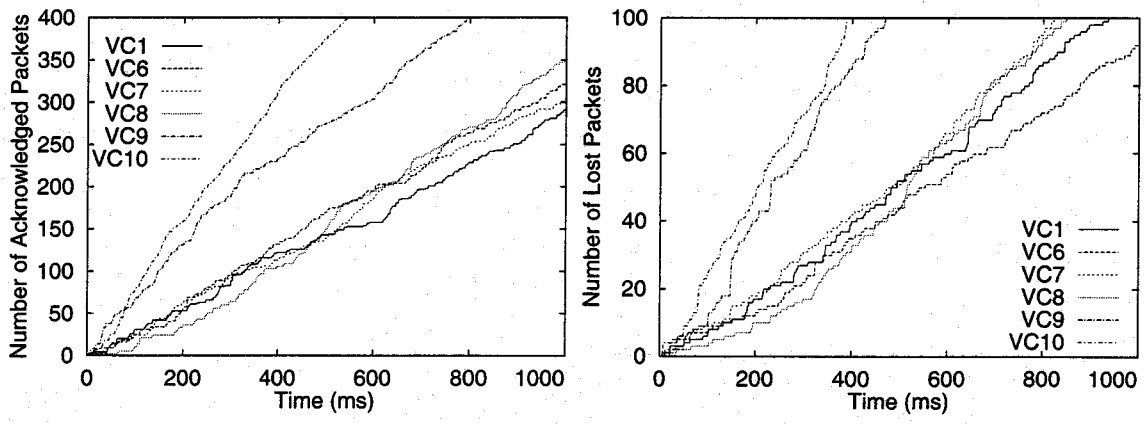


(a) Successfully Transmitted Packets (b) Lost Packets

Figure 4.25: TCP over ABR: τ = 1.0 [msec], Buffer Size = 50 [Kbyte]

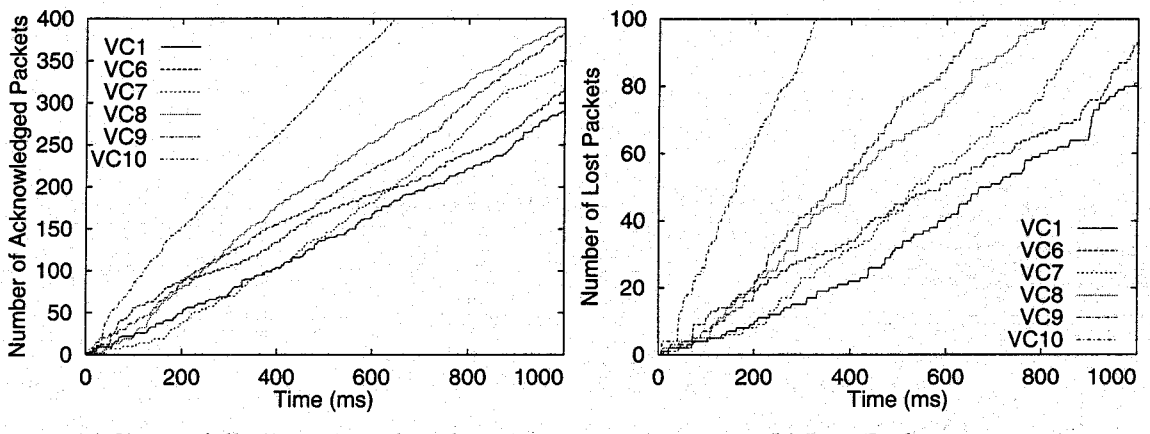
degree is still inferior to that of TCP over ABR (Fig. 4.22(a)). When comparing with TCP over EPD (Fig. 4.23(a)), the number of lost packets of the connection with the small propagation delays (e.g., VC10) increases. It is because the packets from such a connection is more frequently discarded at the switch since the per-VC accounting mechanism tries to perform a fair sharing of the link at the switch.

We next consider the long propagation delay and the small buffer size, i.e., τ is changed from 0.01 [msec] to 1.0 [msec] and buffer size is changed from 300 [Kbyte] to 50 [Kbyte]. Figures 4.25, 4.26 and 4.27 show the case of TCP over ABR, TCP over EPD, TCP over EPD/A, respectively. In this experiment, the total throughput of each case becomes 0.636, 0.891 and 0.903. Note that, the throughput of TCP over ABR is drastically degraded when cell loss takes place at the intermediate switch (Fig. 4.25(b)). As pointed out in Subsection 4.2.3, the drawback of the rate-based congestion control is that two



(a) Successfully Transmitted Packets (b) Lost Packets

Figure 4.26: TCP over EPD: τ = 1.0 [msec], Buffer Size = 50 [Kbyte]

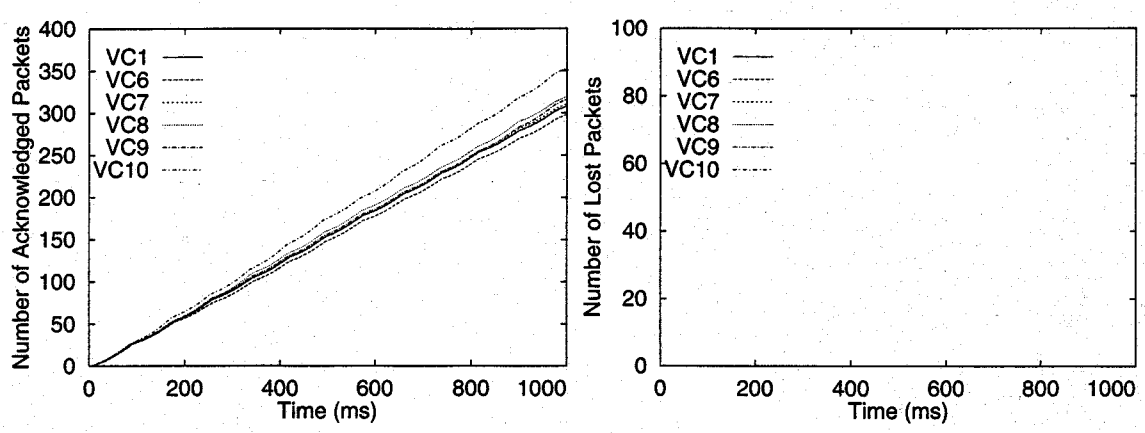


(a) Successfully Transmitted Packets (b) Lost Packets

Figure 4.27: TCP over EPD/A: τ = 1.0 [msec], Buffer Size = 50 [Kbyte]

conditions, no cell loss occurrence and no under-utilization, cannot be fulfilled at the same time in some cases as in the current one. Our current parameter setting is to allow the small cell loss. Then, the throughput is degraded as well as the fairness is also lost. It is possible that cell loss is avoided by tuning control parameters of ABR. For example, Fig. 4.28 shows the case where *RIF* is changed to 1/1024 from 1/256 and *RDF* to 1/8 from 1/32. We observed no cell loss as expected, and the total throughput is improved to 0.814, but still lower than that of TCP over EPD (0.891) because of under-utilization at SW5. On the other hand, in TCP over EPD, the throughput is still high while the degree of unfairness is much larger than that of TCP over ABR. As shown in Fig. 4.27, an introduction of the per-VC accounting does not help improve the fairness among connections in this case.

In summary, TCP over ABR can provide high throughput and fairness unless the



(a) Successfully Transmitted Packets (b) Lost Packets

Figure 4.28: TCP over ABR: τ = 1.0 [msec], Buffer Size = 50 [Kbyte], RIF = 1/1024, RDF = 1/8

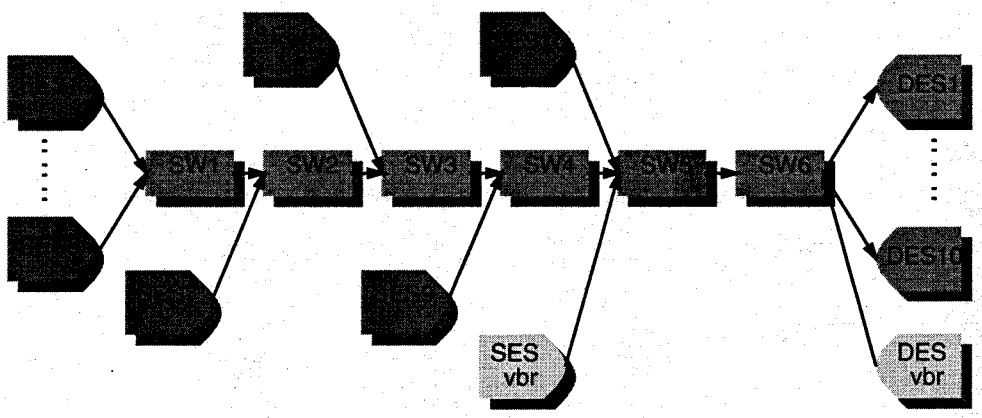


Figure 4.29: Network Model with VBR Connections

propagation delay is large and the switch buffer is small. On the other hand, TCP over EPD can offer high throughput even in such conditions at the compensation of the fairness. The fairness can be improved by providing the per-VC accounting mechanism to EPD (TCP over EPD/A), but its effect is limited. In the case of TCP over ABR, it is possible to obtain higher throughput at the expense of the fairness as in TCP over EPD(/A). That is, the rate-based congestion control algorithm has a freedom to choose control parameters, which affects a balance of the throughput and the fairness. In the following Subsection, we will seek another solution to obtain good throughput and fairness in TCP over ABR.

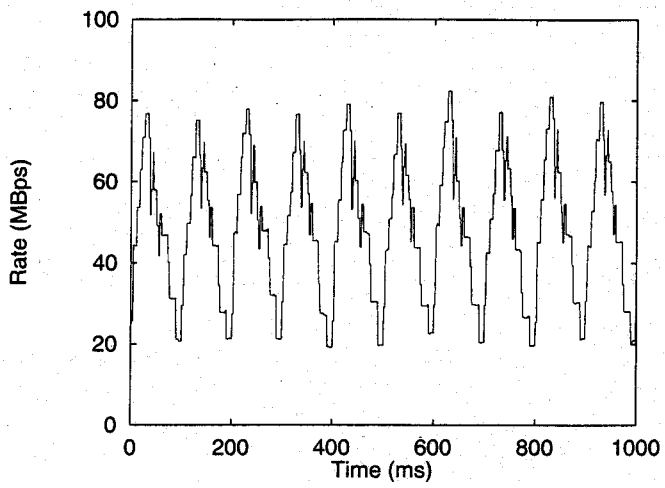
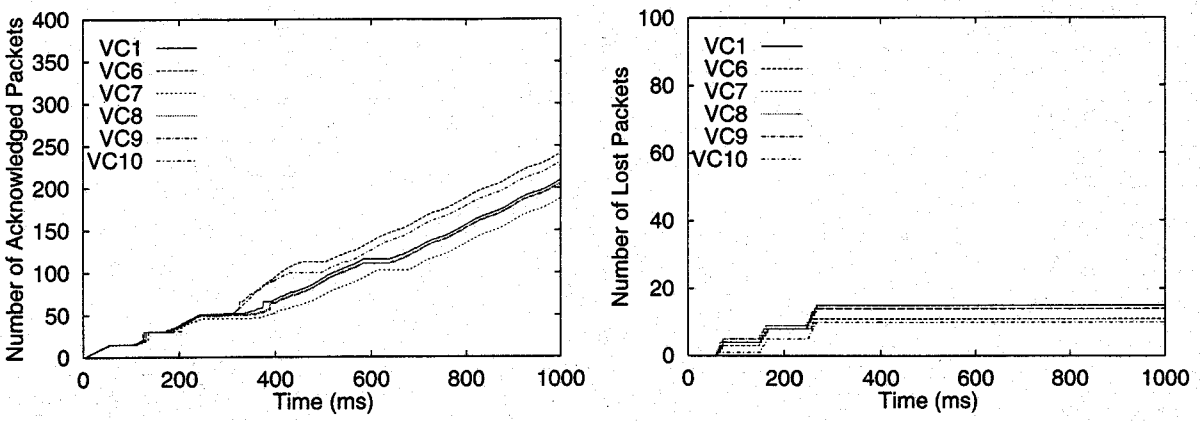


Figure 4.30: Aggregation Rate of VBR Traffic as a Function of Time

4.3.3 Effect of VBR Traffic

In this section, we investigate the effect of VBR traffic on the performance of TCP traffic which utilizes the ABR service class. As an example of the VBR traffic, we have used an MPEG-1 encoded video stream of 30 [frame/sec], 352 × 240 [pixels] with 4.5 [Mbps] average rate and 14.84 [Mbps] maximum rate. In the simulation experiments in this Subsection, ten identical VBR sources (SESvbr 1 to SESvbr 10) are added at SW5 with different starting points (Fig. 4.29). The aggregate cell generation rate of VBR traffic as a function of time is shown at Fig. 4.30. In the following simulation experiments, it is assumed that VBR traffic is given higher priority than ABR/UBR traffic, i.e., VBR traffic cells are transmitted prior to ABR/UBR cells at the switch buffer if VBR cells exist. In other words, the bandwidth available to the ABR/UBR service class is varied dependent on time, and we want to examine the robustness of the rate-based congestion control method in such an environment.

When there does not exist the VBR traffic, TCP over ABR showed an excellent performance as illustrated in Fig. 4.22 if we set the small propagation delay (0.01 [msec]) and large switch buffer (300 [Kbyte]). When we use the same values for RIF and RDF as in previous section ($RIF = 1/256$, $RDF = 1/32$), the total throughput of TCP over ABR traffic is slightly degraded with VBR traffic. The total throughput of TCP traffic becomes 0.569. Since the mean cell generation rate of VBR traffic is 45 Mbps in total (about 0.29 in utilization), the available bandwidth to the ABR service class is not fully utilized even if we take account of RM cells overhead in the ABR service class ($1/32 \approx 0.03$). The fairness is also lost in this case as shown in Fig. 4.31. On the other hand, TCP over EPD shows an excellent performance. The obtained throughput is 0.614, larger than that of TCP over ABR. Further, by comparing Figs. 4.32 and 4.23 (cases with and



(a) Successfully Transmitted Packets

(b) Lost Packets

Figure 4.31: TCP over ABR with VBR Traffic: τ = 0.01 [msec], Buffer Size = 300 [Kbyte], $RIF = 1/256$, $RDF = 1/32$

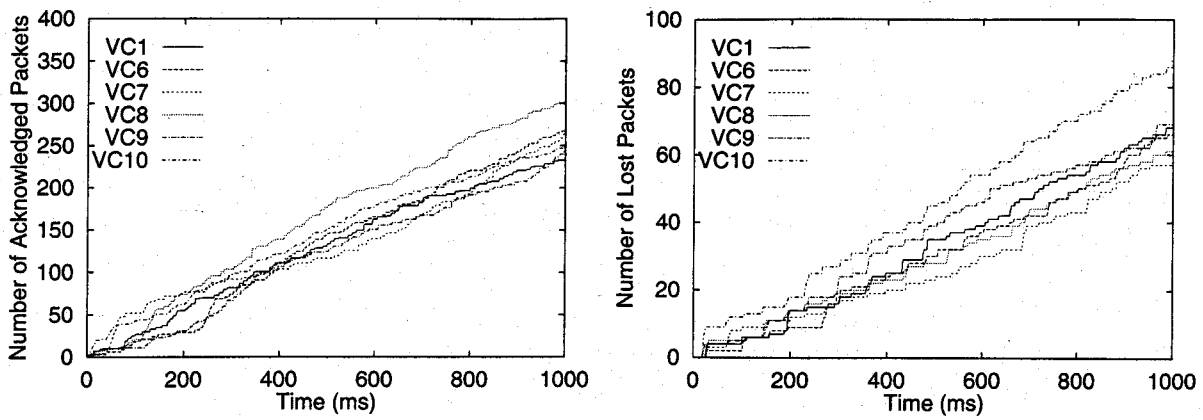
without VBR traffic), we can see that the existence of VBR traffic does not affect the fairness among connections so much.

We next examine the parameter tuning for TCP over ABR. Figure 4.33 shows the case where RIF is changed to $1/512$ and RDF to $1/32$. We choose these values from Condition 1 and Condition 2 of Subsection 3.3 by setting BW to 100 [Mbps] (247.2 [cells/msec]) because the average rate of VBR traffic is 45 [Mbps] (106.0 [cells/msec]). The total throughput is improved to 0.630 because no cell loss occurs by appropriate control parameters. Then, the good fairness property can be achieved as shown in Fig. 4.33(a). That is, even if VBR traffic exists in the network, the performance of TCP over ABR can be kept high by appropriately choosing the control parameters of the rate-based congestion control. However, call admission control (CAC) for the CBR/VBR service classes should reflect the existence of the ABR service class. That is, the call acceptance for the CBR/VBR service class should be limited to keep a some amount of the bandwidth available to the ABR service class.

4.4 Conclusion

In this Chapter, we have investigated performance of TCP over ATM networks for TCP data transfer. We have compared TCP over UBR, TCP over EPD and TCP over ABR with some simulation experiments, and have obtained the following results about their nature of throughput and fairness among connections:

Singlehop Network Case:



(a) Successfully Transmitted Packets

(b) Lost Packets

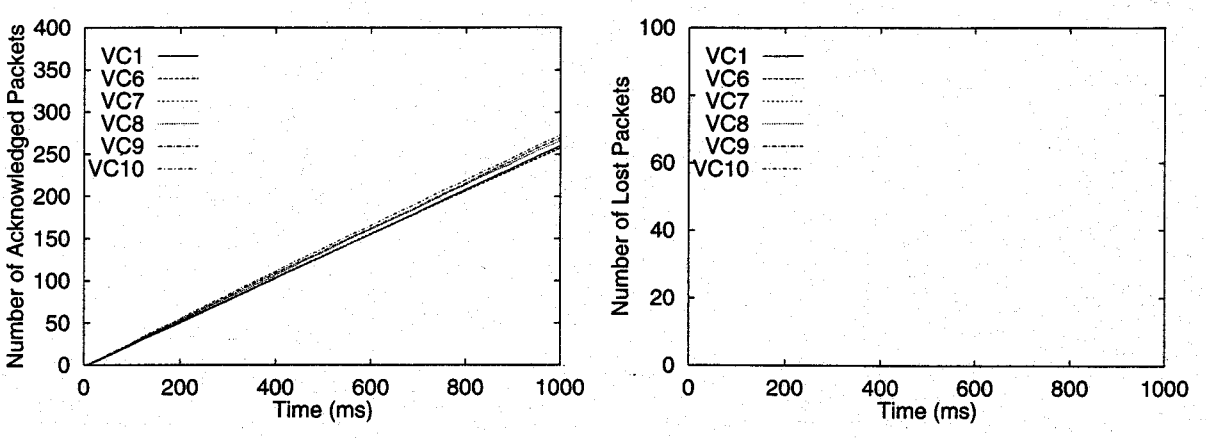
Figure 4.32: TCP over UBR with EPD with VBR Traffic: τ = 0.01 [msec], Buffer Size = 300 [Kbyte]

TCP over ABR can outperform the other two methods (TCP over UBR and TCP over UBR with EPD) if the control parameters of rate-based congestion control algorithm are chosen carefully. While the duality problem of two congestion control mechanisms of TCP and ATM was observed in some parameter settings, we have also shown that it can be avoided when we can choose control parameters of ABR service class appropriately.

Multihop Network Case:

Even in multihop network case, TCP over ABR can achieve higher throughput than TCP over UBR and TCP over UBR with EPD. In the WAN environment, however, where the propagation delay is over 1 ms, an EPD enhancement is preferred since a *pure* ABR mechanism cannot avoid cell loss unless the switch buffer is sufficiently large. We can decrease cell loss and improve the performance while we need a careful setting of the threshold values.

If VBR traffic exists in the network, the control parameters of ABR is carefully tuned according to the volume of the VBR traffic to avoid cell loss. However, a simple calculation of mean VBR traffic rate is sufficient in order to take account of the effect of the VBR traffic. Of course, the larger buffer is preferred to avoid the cell loss since the bandwidth available to the ABR service class cell is certainly reduced due to the VBR traffic.



(a) Successfully Transmitted Packets

(b) Lost Packets

Figure 4.33: TCP over ABR with VBR Traffic: $\tau = 0.01$ [msec], Buffer Size = 300 [Kbyte], $RIF = 1/512$, $RDF = 1/32$

We have left several research topics; when the propagation delay is large and the buffer size is relatively small, we have no means to attain high performance in ABR. One solution was to provide the EPD mechanism to the ABR service as having been demonstrated in this paper. Another approach may be to re-consider the congestion control mechanism of TCP to be suitably applied to the ABR service class. One example seems to be the Slow Start algorithm.

Table 4.1: Control and Simulation Parameters for Singlehop/Multihop Network Case

TCP Specific Parameters	
Packet size	4,352 [Byte] (Approx. 90 [cells])
Receiver's advertised window size	64 [Kbyte]
UBR Service Class Specific Parameters	
PCR (Peak Cell Rate) 353.2 [cells/msec]	
EPD Specific Parameters	
Threshold at switch buffer	half the buffer size
ABR Service Class Specific Parameters	
N_{RM}	32
PCR (Peak Cell Rate)	353.2 [cells/msec]
MCR (Minimum Cell Rate)	$PCR/1000$
ACR (Allowed Cell Rate)	variable
XDF (X_{RM} Decrease Factor)	1/2
$AIRF$ (= $AIR\, N_{RM}/PCR$)	1/64
AIR (Additive Increase Rate)	(see above)
$RDFF$ (= RDF/N_{RM})	16
RDF (Rate Decrease Factor)	(see above)
ICR (Initial Cell Rate)	$PCR/20$
X_{RM}	32
XDF (X_{RM} Decrease Factor)	1/2
TOF (Time Out Factor)	2
TDF (Timeout Decrease Factor)	1/8
Threshold at switch buffer	half the buffer size
Network Specific Parameters	
Link bandwidth, BW	353.2 [cells/msec]
Distance between source and destination, τ	0.01, 0.1, 1.0 [msec]
The number of active connections, N_{VC}	10
Buffer size:	variable (10 [Kbyte] $\sim$ 300 [Kbyte])
Simulation Related Parameters	
Simulation Runtime	50 [sec]

Table 4.2: Parameter Set of ABR after Tuning as a Function of Buffer Size: τ = 1.0 [msec], N_{VC} = 10

Buffer Size at Switch	AIRF	RDFF
10Kbyte	1/256	4
50Kbyte	1/256	4
100Kbyte	1/128	4
200Kbyte	1/256	2
300Kbyte	1/256	2

Chapter 5

Performance Evaluation of HTTP/TCP on Asymmetric Networks

In this Chapter, we extensively investigate the performance of HTTP/TCP on asymmetric networks such as ADSL (Asymmetric Digital Subscriber Line) or Cable modem networks. Since these network services provides *asymmetric* bandwidth for upstream link (from the client to the server) and downstream link (from the server to the client), the throughput of TCP may degrades because it was not designed for asymmetric networks. Our analytical approach is similar to the one adopted in [14], but in addition to TCP Tahoe, we also consider TCP Vegas, which adjusts the sending window size by observing the round trip times of the connection. We investigate the degree of TCP performance degradation under asymmetric networks and the applicability of TCP Vegas to asymmetric networks.

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 [70] 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 all of results, we discuss which combination of HTTP and TCP protocols are appropriate in asymmetric networks.

5.1 Model Definitions

In this Section, we first describe our asymmetric network model in Subsection 5.1.1. Then, HTTP/1.0 and HTTP/1.1 protocols are briefly summarized in Subsection 5.1.2.

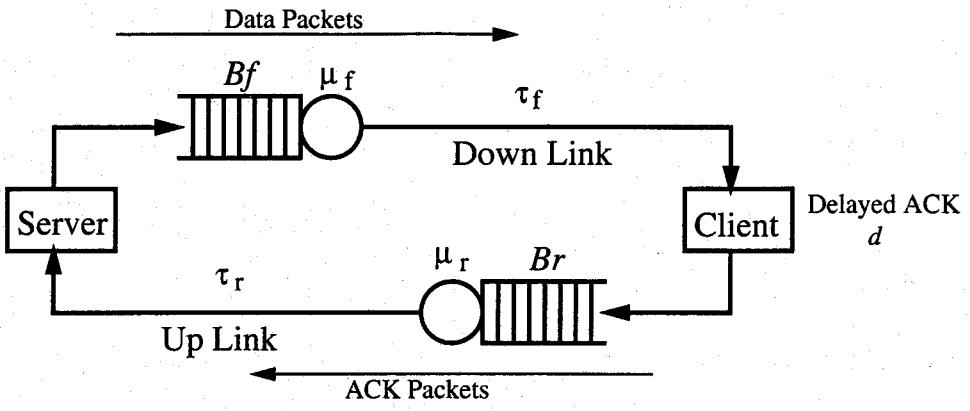


Figure 5.1: Network Model

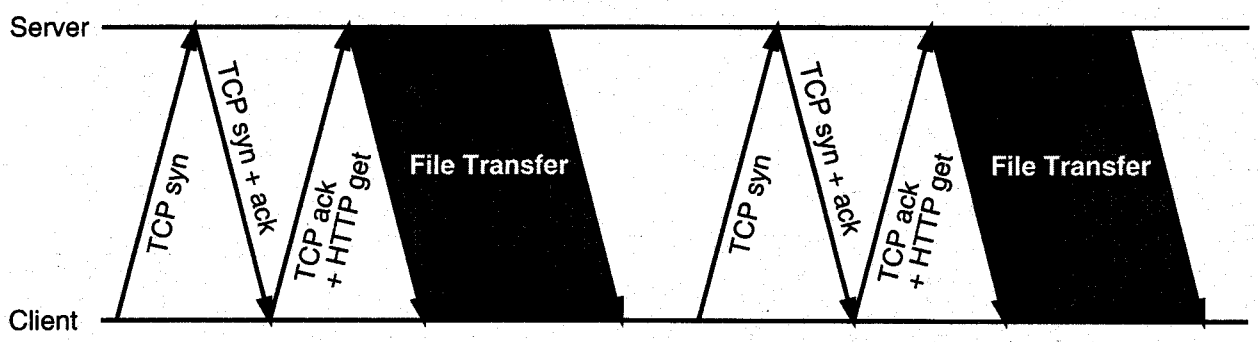
5.1.1 Network Model

The network model which we will use in the analysis and simulation is depicted in Figure 5.1. The model consists of a server, a client, and two links; downlink from the server to the client and upstream link from the client to the server. The upstream and the downstream links have asymmetric bandwidth, denoted as μ_f [data packets/sec] and μ_r [ACK packets/sec], respectively. Note that μ_f and μ_r are represented by units of data/ACK packet, respectively. The buffer sizes are denoted as B_f [data packets] and B_r [ACK packets], and the propagation delays between the server and client are τ_f [sec] and τ_r [sec].

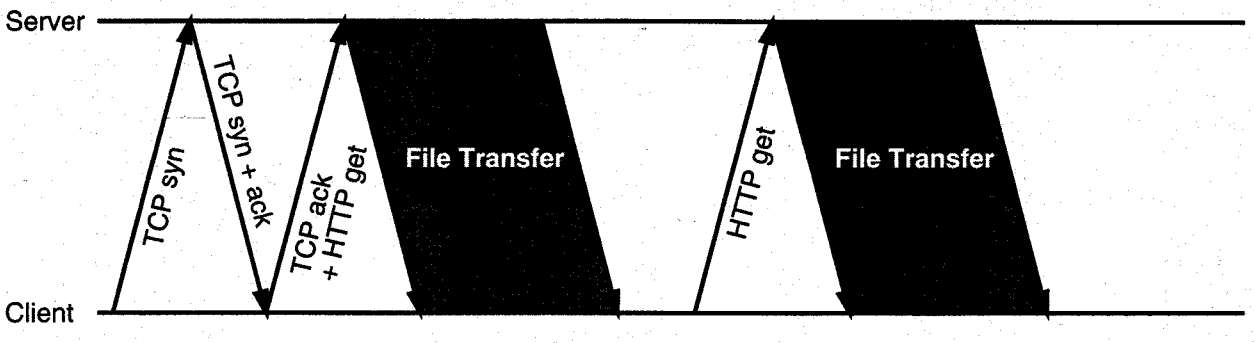
Following [14], we introduce an *asymmetry factor* k defined as μ_f/μ_r to represent the degree of network asymmetry. For instance, if the bandwidth of downstream and upstream links are 16 [Mbps] and 160 [Kbps], respectively, and the data packet size and ACK packet size are 1 [Kbyte] and 40 [byte], we then have $\mu_f = 2000$ [data packets/sec], $\mu_r = 500$ [ACK packets/sec]. Then, the asymmetry factor k becomes $\mu_f/\mu_r = 4$. As the asymmetry factor k becomes large, the upstream link with smaller bandwidth cannot serve all ACK packets generated by the client, and some of ACK packets are lost at the buffer of upstream link. It causes the performance degradation of TCP, which will be discussed in detail in Section 5.3.

5.1.2 HTTP (Hyper Text Transfer Protocol)

Figure 5.2(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



(a) HTTP/1.0 Transfer



(b) HTTP/1.1 Transfer

Figure 5.2: Web File Transfer by Two Versions of HTTP

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 Figure 5.2(b).

5.2 Analysis

In this section, we present the analysis of Web document transfer delay in asymmetric networks. As shown in Figure 5.2, 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.

5.2.1 Connection Setup Phase

As has been described in Subsection 5.1.2, it takes one and a half of RTT 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 (Figure 5.2(b)).

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

$$T_{setup} = \begin{cases} \frac{3}{2}rtt, & \text{(all transfers in HTTP/1.0 and the first transfers in HTTP/1.1)} \\ \frac{1}{2}rtt, & \text{(the second and later transfers in HTTP/1.1)} \end{cases} \tag{5.1}$$

where rtt is the round trip time of the connection.

5.2.2 Document Transfer Phase

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

TCP Tahoe Version

As shown in Figure 1.1(a), in Tahoe version, *cwnd(t)* has cycles. We assume that one cycle starts on time t = 0 [sec]. 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 shown in Equations (1.1). Furthermore, when packet loss is detected, the value of $ssth$ at time t is updated by Equation (1.2). From Equation (1.1), we obtain

$$\frac{d\,cwnd(t)}{d\,ack(t)} = \begin{cases} m, & \text{if } cwnd(t) < ssth; \\ \dfrac{m^2}{cwnd}, & \text{if } cwnd(t) > ssth; \end{cases} \tag{5.2}$$

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

$$\frac{d\,ack(t)}{dt} = \begin{cases} \dfrac{cwnd(t)}{rtt}, & \text{if } \frac{cwnd(t)}{rtt} \le \mu_r; \\ \mu_r, & \text{if } \frac{cwnd(t)}{rtt} > \mu_r; \end{cases} \tag{5.3}$$

where rtt [sec] is RTT for the case where the downstream link buffer is empty, i.e.,

$$rtt = \frac{1}{\mu_f} + \tau_f + \frac{1}{\mu_r} + \tau_r \tag{5.4}$$

Finally, $cwnd(t)$ can be obtained from Equations (5.2) and (5.3) by utilizing the following relationship;

$$\frac{d\,cwnd(t)}{dt} = \frac{d\,cwnd(t)}{d\,ack(t)} \frac{d\,ack(t)}{dt} \tag{5.5}$$

The instantaneous throughput at time t, $\rho(t)$ [packets/sec], is then obtained by Equation (5.5) as;

$$\rho(t) = \begin{cases} \dfrac{cwnd(t)}{rtt}, & \text{if } \frac{cwnd(t)}{rtt} \leq \mu_f; \\ \mu_f, & \text{if } \frac{cwnd(t)}{rtt} > \mu_f; \end{cases} \tag{5.6}$$

One cycle of TCP Tahoe terminates when packet loss occurs. Let W_{max} [packets] denote the window size at time when packet loss occurs. To obtain W_{max}, we need to consider how the packet 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 upstream and downstream links are also included.

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

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

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

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

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

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

In this case, packet loss occurs when the burst size exceeds the buffer size. Once the upstream link is fully utilized, the server receives ACK packets with rate of μ_r. The corresponding window size, W_r, is equal to the sum of the upstream link 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 upstream link and downstream link). Further, we introduce b_i to represent the number of packets 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 packet is lost at that time. After that, the length of the burst is incremented in turn as the server receives an ACK packet and the window size is increased by one packet. 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 packet loss occurs if the size of some burst exceeds the buffer size of the downstream link B_f, and one cycle terminates.

In [14], 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,

$$W_{max} = \sum_i b_i \text{ when } \max_i(b_i) = B_f + 1 \tag{5.7}$$

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;

$$\overline{\rho} = \frac{1}{T}\int_0^T \rho(t)\,dt \tag{5.8}$$

TCP Vegas Version

In TCP Vegas, evolution of the window size does not have any cycle. See Figure 1.1(c). 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 downstream link can be fully utilized. In what follows, we calculate W_{max}.

The convergence of TCP Vegas is achieved by the following algorithm. TCP Vegas observes RTT of each packet, and controls the window size according to RTTs. From Equation 1.4 in Section 1.2, the window size is kept unchanged if the following condition is satisfied [28, 29, 84];

$$\frac{m\,\alpha}{base\_rtt} < \frac{cwnd}{base\_rtt} - \frac{cwnd}{rtt} < \frac{m\,\beta}{base\_rtt} \tag{5.9}$$

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

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

$$cwnd = \mu_f \cdot rtt$$

By utilizing the above relations, Equation (5.9) 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 downstream link is fully utilized without packet 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 packets continuously as is the case of TCP Tahoe, TCP Vegas cannot also avoid packet losses. Therefore, the window size has cycles like that of TCP Tahoe (Figure 1.1(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 Subsection 1.2.3.

5.2.3 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 5.2.1, and the actual document transfer time, where is obtained from $\rho(t)$ in Section 5.2.2 by solving the following equation;

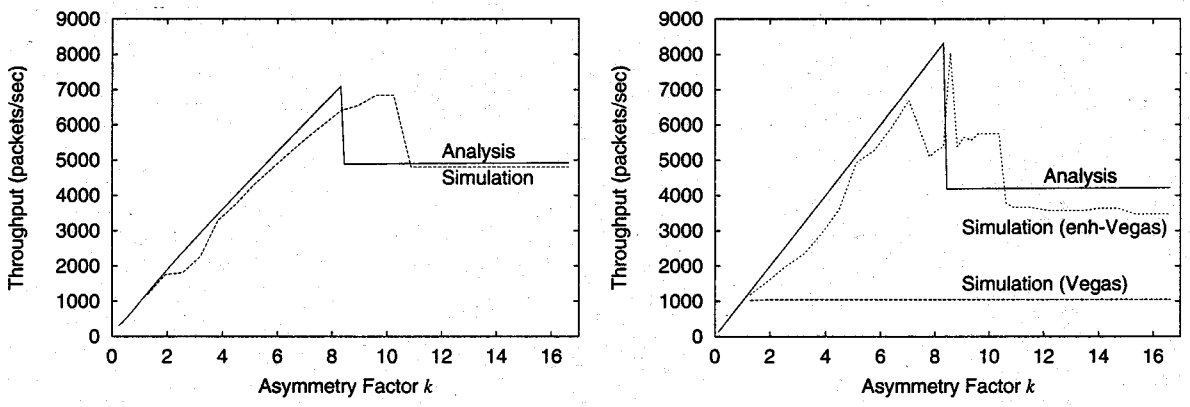
$$\int_0^{T_{transfer}} \rho(t)\, dt = S \tag{5.10}$$

where S [packets] is the size of Web document. Finally, T_{total} [sec] can be obtained from Equations (5.1) and (5.10) as;

$$T_{total} = T_{setup} + T_{transfer} \tag{5.11}$$

Table 5.1: Parameter Set Used in Numerical Examples

Variable		Default Value
TCP data packet size	m	1 [Kbyte]
TCP ACK packet size	m_{ack}	40 [byte]
Downstream link buffer size	B_f	8 [packets]
Downstream link bandwidth	μ_f	4000 [packets/sec]
Downstream link propagation delay	τ_f	1 [msec]
Upstream link buffer size	B_r	5 [packets]
Upstream link bandwidth	μ_r	1000 [packets/sec]
Upstream link propagation delay	τ_r	1 [msec]



(a) TCP Tahoe

(b) TCP Vegas

Figure 5.3: Throughput vs. Asymmetry Factor k

5.3 Numerical Examples and Discussion

In this Section, we show some numerical examples based on our analysis presented in Section 5.2, 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 5.1 are used in this section unless otherwise stated.

5.3.1 Comparisons of TCP Mean Throughputs

In this subsection, we only consider TCP level performance. Figure 5.3 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 downstream link bandwidth μ_f. In the figure, we also plot simulation results to confirm the accuracy of our analysis presented in the previous section.

Figure 5.3(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 decreased and kept constant when k is large. These two regions correspond to Cases 1 and 2 presented in Subsection 5.2.2, respectively. That is, if k is smaller than the downstream link buffer size B_f, packet loss occurs only when the window size exceeds the sum of bandwidth-delay product of the connection and the upstream and downstream link buffer sizes. In other words, the throughput increases in proportion to the downstream link bandwidth, μ_f. If k is larger than B_f, on the other hand, packet loss takes place when the size of the burst generated by the server exceeds the downstream link 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 Figure 5.3(b), it is noticeable that the simulation result of TCP Vegas shows quite low throughput (the line labeled as "Simulation (Vegas)"). This can be explained as follows; as described in Subsection 1.2.3, 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 packets passing through the upstream link is first delayed at the upstream link buffer in asymmetric networks. As a result, the window size is converged to the capacity of the upstream link, 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 upstream link buffer. The line labeled by "Simulation (enh-Vegas)" in Figure 5.3(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.

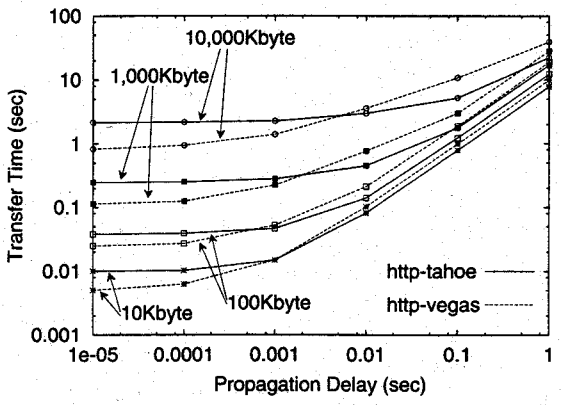


Figure 5.4: Document Transfer Time vs. Propagation Delay

It is clear by comparing Figures 5.3(a) and 5.3(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 Figure 1.1(c), and no packet loss occurs. Therefore, TCP Vegas can achieve higher throughput than TCP Tahoe. When k becomes larger than B_f, however, packet loss occurs even in TCP Vegas and the change of the window size has cycles like TCP Tahoe (Figure 1.1(c)). It is because even TCP Vegas cannot avoid packet losses caused by the bursty generation of data packets 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 1.2.3. Then, the throughput of TCP Vegas becomes lower than even that of TCP Tahoe.

5.3.2 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 (labeled "http-tahoe" in Figures 5.4-5.7), 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 5.4 shows 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

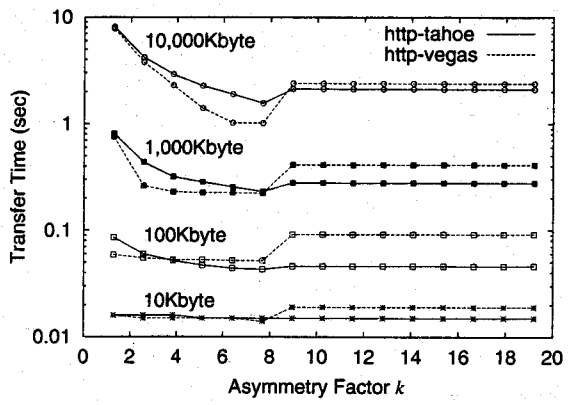
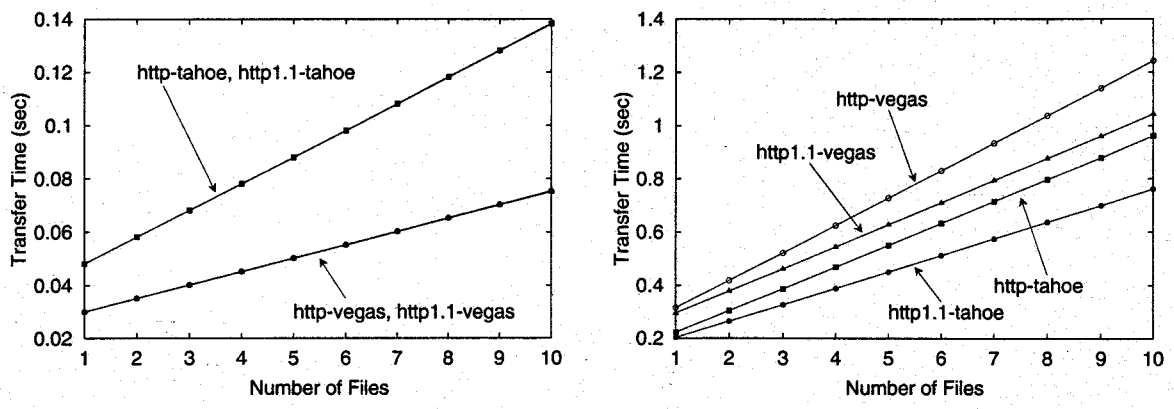


Figure 5.5: Document Transfer Time vs. Asymmetric Factor k

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 packets. 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 Figure 5.5 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 5.3.1 (see Figure 5.3). 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 downstream link 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 5.6 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. Fig-



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

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

Figure 5.6: Throughput vs. Asymmetry Factor k

Table 5.2: 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

ures 5.6(a) and 5.6(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 (Figure 5.6(a)). Even if the propagation delay becomes large, the effect of HTTP/1.1 is limited, and the selection of TCP becomes more important (Figure 5.6(b)).

We last consider another case for the document size distribution. The probability distribution of the document that we tested is shown in Table 5.2, which we brought from [85]. Figure 5.7 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, HTTP/1.1 does not give

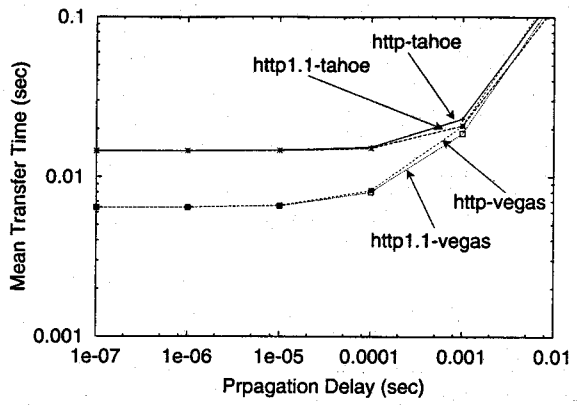


Figure 5.7: Mean Document Transfer Time vs. Propagation Delay

performance improvement in asymmetric networks.

5.4 Conclusion

In this Chapter, 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 downstream link 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, its effect 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 chapter 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.

Chapter 6

Comparisons of Packet Scheduling Algorithms for Fair Service among Connections

In this Chapter, we focus on *fair* service among connections, and investigate the degree of fairness provided to TCP connections by comparing three packet scheduling algorithms at the router, through the mathematical analysis approach. The first one is FIFO (First In First Out, or Drop-Tail), which is widely used in the current Internet routers because of its simplicity. The second is RED (Random Early Detection) [36], which drops incoming packets at a certain probability. While the original idea of the RED algorithm is to avoid consecutive dropping of packets belonging to the same connection, it also has a capability of achieving a fair service among connections by spreading packet losses. The last one is DRR (Deficit Round Robin), which is a more aggressive one in the sense that it actively maintains per-flow queueing for establishing fair service. For TCP, we consider TCP Reno and TCP Vegas.

In addition to evaluating fairness properties of three algorithms, We propose the enhanced version of RED algorithm using the analysis results, where we set each connection's packet dropping probability dependently on its input link capacity, to avoid the unfairness property of the original RED algorithm. Another enhancement method of RED can be found in [37], where the flow state are maintained for some degree of fairness enhancements. Furthermore, we show that the above method can be used to resolve an unfairness problem of the DRR algorithm, which is inevitable due to an inherent unfairness of FIFO discipline.

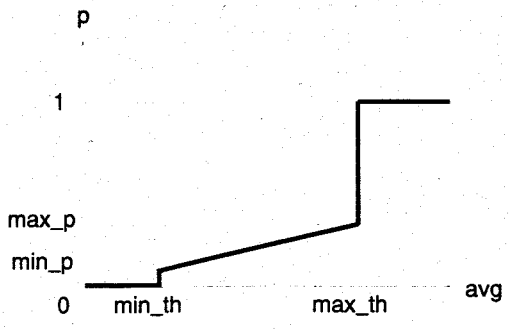


Figure 6.1: RED packet dropping rate $p(x)$

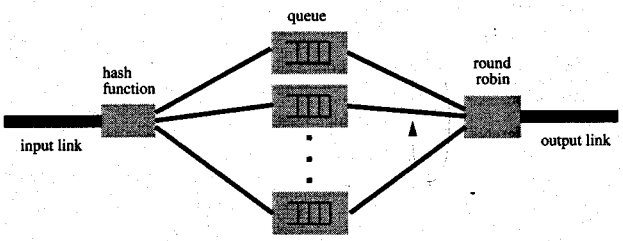


Figure 6.2: DRR (Deficit Round Robin)

6.1 Model Definitions

6.1.1 Packet Scheduling Algorithms

In what follows, we briefly summarize the three packet scheduling algorithms, FIFO, RED and DRR.

FIFO (First In First Out)

A FIFO algorithm is widely used in the current Internet routers because of its simple implementation. The incoming packets are accepted in order of arrivals. When the buffer at the router becomes full, arriving packets are dropped. Therefore, packets belonging to a particular connection can sometimes suffer from bursty packet losses. Then, fast retransmit implemented in TCP does not work effectively. It is also likely to introduce bursty transmission of packets [36], which often results in further packet losses.

RED (Random Early Detection)

The problem mentioned above is solved by RED [36]. The RED algorithm is designed to cooperate with congestion control mechanisms provided in TCP. In RED, the router observes the avarage queue size (buffer occupancy), and the packets arriving at the router

are dropped with a certain probability.

It detects incipient congestion by monitoring the average buffer occupancy at the router, and notifies it of the connections by dropping packets to avoid further congestion. By keeping the average queue size (buffer occupancy) low, buffer overflow can be avoided even when packets from the connection continuously arrive. That is, the algorithm has no bias against bursty traffic. In RED, the packets arriving at the router are dropped with a certain probability. By this mechanism, the packet loss does not become bursty, and the number of lost packets is roughly proportional to the connection's share of the link bandwidth through the router. Therefore TCP connections are expected to effectively reduce the window size according to its bandwidth share.

RED sets the packet dropping probability by a function of average queue size. We define avg [packets] as the average queue length calculated by using a low pass filter with an exponential weighted moving average. The parameter avg is compared to two thresholds: a minimum threshold (th_{min} [packets]) and maximum threshold (th_{max} [packets]). The dropping probability according to the queue size avg is determined in different ways as follows;

1. If $avg < th_{min}$, then all arriving packets are accepted.
2. If $th_{min} < avg < th_{max}$, then arriving packets are dropped with probability $p(x)$, which is a function of average queue length, x. A typical function of $p(x)$ is illustrated in Figure 6.1, but a constant dropping probability is usually used in an actual situation [36].
3. If $th_{max} < avg$, then all arriving packets are dropped.

DRR (Deficit Round Robin)

The DRR algorithm [22] is an extension of the round robin algorithm to be suitable to treat the variable-sized packets. The buffer at the router is logically divided into multiple queues. The arriving packets of each connection are stored in the pre-assigned queue by using a hash function, and those are served in a round-robin fashion. See Figure 6.2. A difference from the *pure* round robin algorithm is that the packets with variable length can be allowed to keep the fairness among connections. In DRR, the bandwidth not used in the round is preserved to be used in the next round if the packet is too large to be served in the current round.

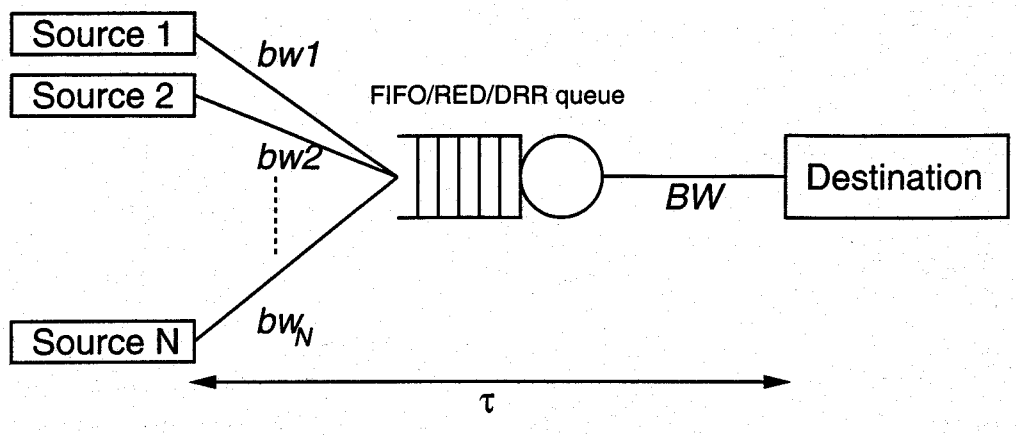


Figure 6.3: Network Model

6.1.2 Network Model

Recalling that our main purpose of the current paper is to investigate the fairness aspect of packet scheduling algorithms, we will use a simple network model as depicted in Figure 6.3.

There are the number N of connections between N sources (SES_1, SES_2 , ..., SES_N) and one destination (DES). N connections share the bottleneck output link of the router. The capacity of the input link between the sources and the router are defined as bw_1, bw_2, ..., bw_N Kbps, and that of the output link between the router and destination is BW Kbps. We assume $bw_1 \leq bw_2 \leq \ldots \leq bw_N$. By the above model, we intend to consider the uplink of the access line of the ISP, which is shared by the subscribers with different capacities. Note that in Section 6.4, we will consider the downlink of the access line.

In the following numerical examples throughout the paper, the propagation delay between SES_i and DES, τ, is identically set to be 100 [msec]. The buffer size of the router is 30 [packets]. A TCP packet size is fixed at 2 [Kbytes]. Every sender is assumed to be a greedy source, that is, it has infinite packets to transmit. We also assume that in the case of DRR, the connection can be identified by the router so that the packets from the connection can be appropriately queued at the per-flow buffer at the router.

Using this network model, we consider the situation where the uplink of the access line of ISP is shared by the subscribers with different capacities. In this paper, the effect of the reverse traffic is also considered. in the model where the downlink is shared by the subscribers. The objective of this investigation is to confirm the applicability of our discussions and analyses in the above are also applicable to this reverse traffic model. The similar model is treated in in [86], but we consider RED and DRR as the packet scheduling algorithm in addition to FIFO algorithm employed in [86]. Further,

we devote the fairness aspects of packet scheduling algorithms which are not considered in [86].

6.1.3 Definition of Fairness

We define the *fair* service by taking account of the input link capacity. Its simplest form is that the throughput is given in proportion to its input link capacity under the condition that the output link capacity is smaller than total of the input link capacities. That is, we say that a *good fairness is achieved* if the throughput of connection i, ρ_i, is given as

$$\rho_i = BW \cdot \frac{bw_i}{\sum_j bw_j}$$

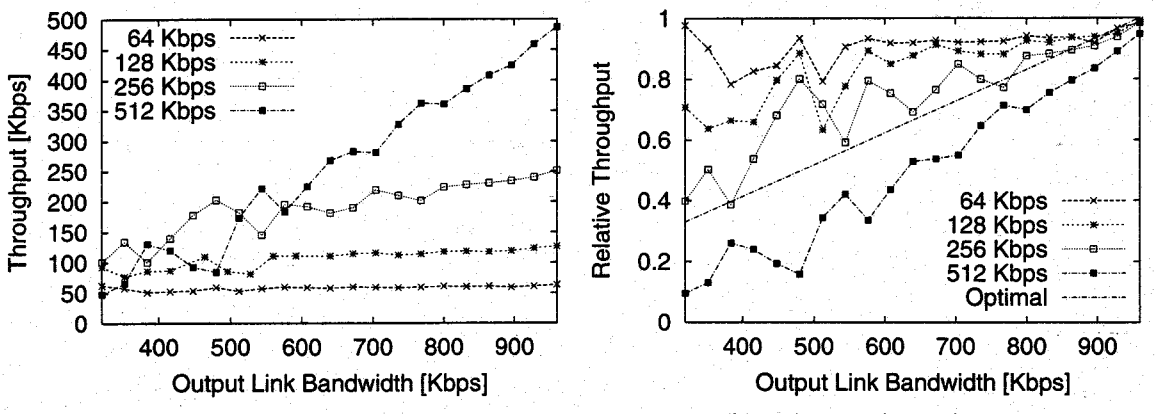
As an example, suppose that there are three sources ($N = 3$) with $bw_1 = 64$ [Kbps], $bw_2 = 128$ [Kbps], $bw_3 = 256$ [Kbps]. If the output link capacity of the router BW is 336 [Kbps], then the perfect fairness is achieved when throughputs of three connections are 48 [Kbps], 96 [Kbps] and 192 [Kbps], respectively.

We note that other definitions of the fairness can be considered. A more natural definition may be the function of subscription fees, which may be determined by (but not be proportional to) the input link capacity in the ISP model. We will not treat such a case for simplicity of presentation, but it is not difficult to incorporate it. For example, the weight factor is allowed to be arbitrary in the DRR case. The RED case can also be treated in this context by utilizing our analysis presented later.

6.2 Case of TCP Reno

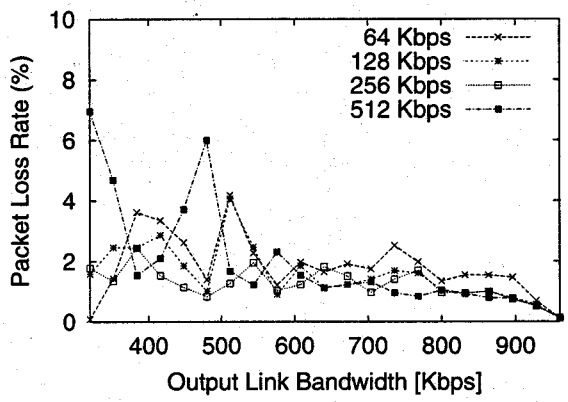
In this Section, we consider TCP Reno to investigate the fairness property of three packet scheduling algorithms. In addition to the simulation results, we develop the analysis result for the RED scheduling algorithm. The analysis results supports observations on the fairness property of the RED algorithm obtained from the simulation results. We then investigate DRR to demonstrate its effectiveness through simulation experiments.

In what follows, we set four TCP connections which have different capacities of 64, 128, 256 and 512 [Kbps]. The output link capacity is varied from 400 [Kbps] to 960 [Kbps] to investigate the effect of the output link capacity on fairness. In the simulation results, we simulated 5,000 [sec] in each experiment to obtain the result, which approximately corresponds to 300,000 packets generation.



(a) Average Throughput

(b) Relative Throughput

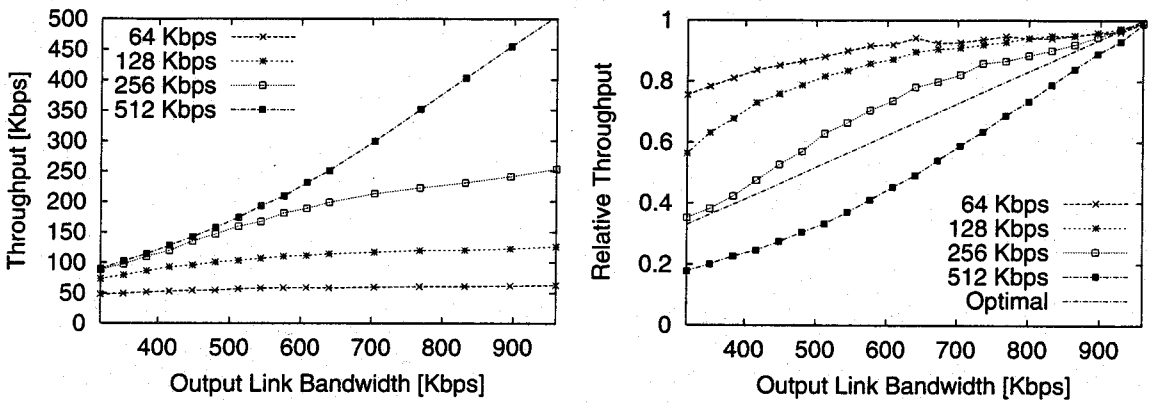


(c) Packet Loss Rate

Figure 6.4: FIFO Case with TCP Reno

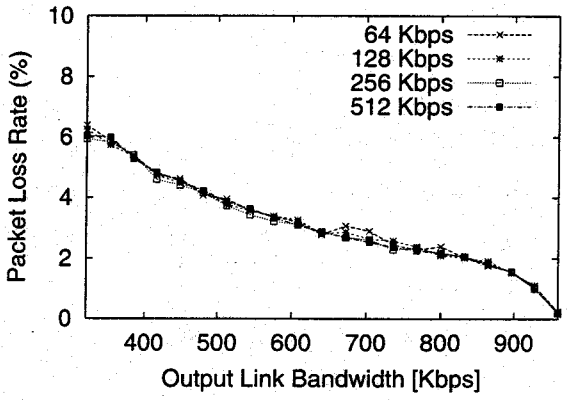
6.2.1 FIFO Case

We first show the FIFO case in terms of the average throughput during the simulation run (Figure 6.4(a)), the relative throughput (Figure 6.4(b)), and packet loss rate (Figure 6.4(c)) for all connections as a function of the output link capacity. Relative throughput means the ratio of the average throughput against the input link capacity. When all connections have identical relative throughput, it is said that the router perfectly provides fair service among connections in our definition. From Figures 6.4(a) and 6.4(b), it is clear that fairness cannot be kept at all. In some region where the output link capacity is small, the throughput of the connection with smaller input link capacity is larger even than that of the connection with larger input link capacity. It can be explained as follows. In the FIFO algorithm, packet loss occurs independently of the packet arrival rate as shown in Figure 6.4(c), and the packet loss becomes bursty. Since the connection with larger input link capacity experiences a higher degree of burstiness of packet losses, its performance degradation becomes larger.



(a) Average Throughput

(b) Relative Throughput



(c) Packet Loss Rate

Figure 6.5: RED Case with TCP Reno

6.2.2 RED Case

Simulation Results

We next investigate the RED case. Recalling that the buffer size of the router is set to be 30 [packets], we set th_{min} = 5 [packets], th_{max} = 15 [packets] and p = 0.02 in simulation. p shows the packet dropping probability defined in RED, with which incoming packets are dropped when the avarage queue length is over the threshold th_{min}. Figure 6.5 shows simulation results of the RED algorithm in that case. By comparing Figure 6.5(a) with Figure 6.4(a), it can be observed that the RED algorithm can attain higher total throughput than that of the FIFO algorithm because RED can avoid bursty packet losses by dropping arriving packets with probability p, which results in that TCP's fast retransmit algorithm works effectively. However, if we focus on the fairness, it is clear that an improvement is very limited. It is especially true when the output link capacity is small; the throughput of all connections becomes almost identical (Figure 6.5(a)). Also, the packet loss rates of all connections are almost equal as shown in Figure 6.5(c).

Of course, this is one of key features that the RED algorithm intends; the number of the lost packets of each connection can be kept in proportion to its input link capacity by its mechanism. The problem is that it leads to the unfairness treatment of connections with different capacities.

The above result is just one example. Also, it is questionable whether simulation time of 300,000 packets generation is adequate or not for examining the fairness degree. To examine its generality, we next show the analysis of the RED algorithm. By this, we will explain why the RED algorithm causes unfairness among connections even when their packet loss rate are almost equal. Through analysis, it is proven that the unfairness observed in simulation is inherent in the RED algorithm.

Analysis

We assume in the following analysis that there are N connections in the network (Figure 6.3) with the input link capacities of bw_1, bw_2, ..., bw_N [packets/sec], where $bw_1 \leq bw_2 \leq$, ..., $\leq bw_N$. We denote the packet dropping probability of the RED algorithm by p, and the propagation delay between sources and the destination by τ [sec]. We also assume that the average queue length is always larger than th_{min} [packets], that is, all arriving packets are dropped with probability p. For analysis, we focus on TCP's typical *cycle* of the window size as shown in Figure 6.6; the cycle begins at the time when the previous packet loss occurs, and terminates when the next packet loss occurs. We consider that the cycle begins at time t = 0 [sec]. We do not take account of the Slow Start phase since the objective of the RED algorithm is essentially to avoid to fall into that phase.

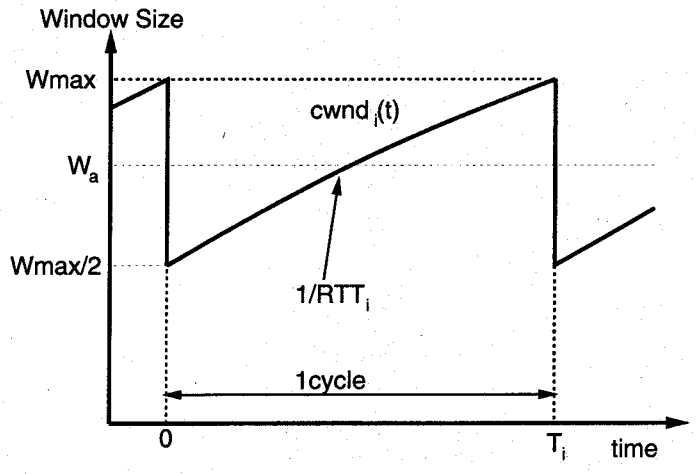


Figure 6.6: TCP's Cyclically Change of the Window Size for Connection i

Since all arriving packets are dropped at the router with probability p by our as-

sumption, the connection can send $1/p$ packets in one cycle (between the events of packet losses). We define the number of packets transmitted during one cycle as N_p [packets], that is,

$$N_p = 1/p \tag{6.1}$$

During the cycle, the window size of connection i, $cwnd_i(t)$ [packets], is increased linearly since we only consider the congestion avoidance phase. The window size is halved when packet loss detected by fast retransmit, and therefore $cwnd_i(t)$ is given as

$$cwnd_i(t) = \frac{W_{max}}{2} + \frac{1}{RTT_i} \cdot t, \qquad 1 \leq i \leq N \tag{6.2}$$

where RTT_i [sec] is an average RTT of packets for connection i, and W_{max} [packets] is the value of the window size at the time when packet loss occurs. Then, the following equation for the total number of the packets in one cycle should be satisfied for connection i;

$$\int_0^{T_i} cwnd_i(t)dt = N_p, \qquad 1 \leq i \leq N \tag{6.3}$$

where T_i is the time duration of the cycle as shown in Figure 6.6. From Equations (6.2) and (6.3), we can obtain W'_{max} [packets], the window size at the time when the next packet loss occurs, as

$$W'_{max} = \sqrt{{W_{max}}^2 + 2N_p} \tag{6.4}$$

From Equations (6.1) and (6.4), we can obtain $\overline{W}_{max}$ [packets], the average value of W_{max} by equating W'_{max} and W_{max}. That is,

$$\overline{W}_{max} \simeq \sqrt{\frac{8}{3p}} \tag{6.5}$$

As a result, we derive W_a [packets], the average window size during the cycle as;

$$W_a = \frac{3}{4}\overline{W}_{max} \tag{6.6}$$

See Figure 6.6. From the equation above, we can see that the change of the window size does not depend on each connection's input link capacity, but on the packet dropping probability of the RED algorithm.

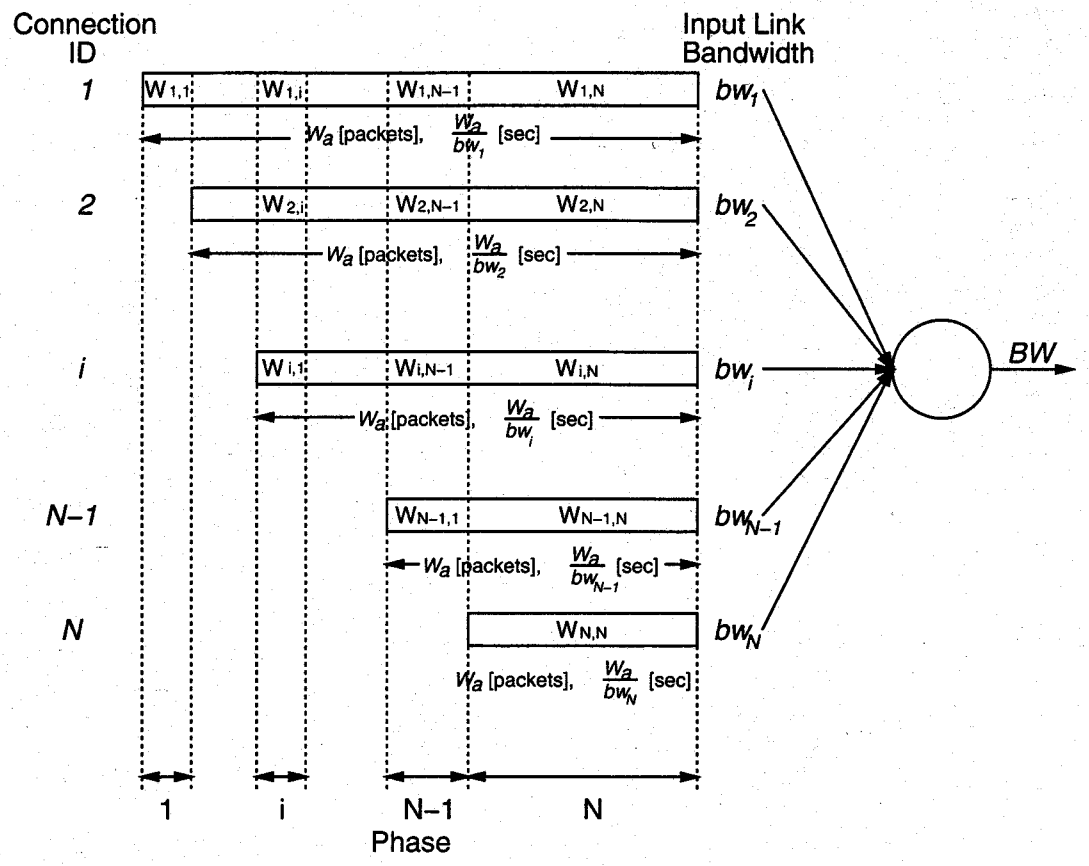


Figure 6.7: Analysis of the RED Algorithm

For further analysis, we make an assumption that each connection's window size is fixed at the average value, W_a. We then derive ρ_i, the throughput of connection i when W_a packets of its window are served at the router. To simplify the analysis, we consider the situation where all connections' first packets of the windows arrive at the router simultaneously as shown in Figure 6.7. In this figure, each square shows the *burst* of connection i's W_a packets, and its length represents the time duration $\frac{W_a}{bw_i}$ [sec]. Since all connections have different capacities bw_i on their links, it takes different time duration $\frac{W_a}{bw_i}$ for all packets of connection i to arrive at the router as illustrated in Figure 6.7. As illustrated in this figure, the packet burst of connection i is not served at the same rate, and it depends on the number of the connections sending their packets simultaneously. We divide all connections' packet burst into N "phases" according to the number of connections which send the packets simultaneously. For example, since the number of connections which transmitting their packets is i in phase i, the router processes i connections' packets at the rate of BW [packets/sec]. We denote the number of packets of connection i belonging to phase j by $W_{i,j}$ [packets] ($1 \leq i, j \leq N$). Since all packets in the phase are dealt at the rate in proportion to its input link bandwidth, we determine

$W_{i,N}$ for phase N as follows;

$$\begin{aligned} W_{N,N} &= W_a \\ W_{i,N} &= W_{N,N} \cdot \frac{bw_i}{bw_N}, \qquad 1 \le i \le N. \end{aligned}$$

In the same manner, we can obtain all of $W_{i,j}$ by solving the following equations;

$$\begin{aligned} W_{j,j} &= W_a - \sum_{k=j+1}^{N} W_{j,k}, \qquad 1 \le j \le N-1 \\ W_{i,j} &= W_{j,j} \cdot \frac{bw_i}{bw_j}, \qquad 1 \le j \le N-1, 1 \le i \le j-1 \end{aligned}$$

The rate at which the packets are served at the router in the phase j, S_j [packets/sec], must depend on the total capacity of the connections of the phase j. Since, in the phase j, all packets belonging to from connection 1 to connection j are served at the router, S_j becomes as follows;

$$S_j = \begin{cases} BW, & \text{if } \sum_{k=1}^{j} bw_k > BW \\ \sum_{k=1}^{j} bw_j, & \text{otherwise} \end{cases} \tag{6.7}$$

Therefore, the throughput of connection i at phase j, $R_{i,j}$, can be determined as follows;

$$R_{i,j} = \frac{W_{i,j}}{\frac{W_{i,j}}{\sum_{k=1}^{j} W_{k,j}} S_j} = \frac{\sum_{k=1}^{j} W_{k,j}}{S_j} \tag{6.8}$$

From Equations (6.7) and (6.8), ρ_i can be calculated as follows;

$$\rho_i = \sum_{k=N+1-i}^{N} \left(\frac{W_{i,j}}{W_a} R_{i,j} \right) \tag{6.9}$$

Although the RED algorithm can eliminate the bursty packet losses leading to TCP's retransmission timeout expiration, timeout expiration cannot be avoided perfectly [39]. Even if timeout expiration rarely happens, the effect of timeout expiration on throughput is large. Therefore, we next consider the throughput degradation caused by retransmission timeout expiration. We denote the probability of occurring timeout expiration

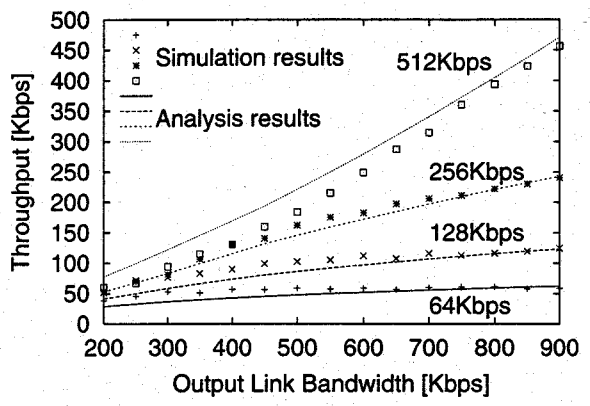


Figure 6.8: Accuracies of Analysis Result in TCP Reno

in the window by P_{to}. We determine P_{to} according to the following simple equation;

$$P_{to} = \sum_{i=2}^{\infty} \binom{W_a}{i} \cdot p^i \cdot (1-p)^{W_a+1-i} \tag{6.10}$$

We assume that RTO_i [sec], the timeout duration for retransmission, becomes twice RTT_i for connection i. RTT_i can be calculated by considering the effect of the other connections' traffic;

$$RTT_i = 2\tau + \frac{\sum_{k \neq i} W_a}{BW} + \frac{W_a}{\rho_i} \tag{6.11}$$

From these results, we finally have ρ_i', the throughput of connection i, by considering the effect of TCP's retransmission timeouts;

$$\begin{aligned} \rho_i' &= (1-P_{to}) \cdot \rho_i + P_{to} \cdot \frac{\frac{W_a}{\rho_i}}{\frac{W_a}{\rho_i} + RTO_i} \rho_i \\ &= \frac{\rho_i \cdot W_a + (1-P_{to}) \cdot \rho_i^2 \cdot RTO_i}{W_a + \rho_i RTO_i} \end{aligned} \tag{6.12}$$

Equation (6.12) is obtained as follows. The first term $(1-P_{to}) \cdot \rho_i$ represents the throughput without retransmission timeout, and the second term $\frac{\frac{W_a}{\rho_i}}{\frac{W_a}{\rho_i}+RTO_i}\rho_i$ is that with retransmission timeout. By Equation (6.12), we can obtain the each connection's TCP throughput under RED algorithm with taking account of the throughput degradation caused by TCP retransmission timeouts.

Figure 6.8 shows the throughput results from our analysis as a function of the output link capacity. In the figure, points represent the simulation results (which correspond

to Figure 6.5(a)), and the lines show analysis results. We can observe from this figure that our analysis can give good agreements with simulation results, and that the unfairness property of the RED algorithm in the case of small output link capacity can be observed. This unfairness can be explained from the analysis result as follows. When the output link bandwidth becomes small, the rate at which the packets are served at the router of phase j becomes BW in almost all the phases. It is clearly shown in Equation (6.7). That is, packets arriving at the router are served at rate BW, which results in that the throughput of all connections become equivalent. Furthermore, the connection whose input link bandwidth is larger can suffer from throughput degradation caused by TCP retransmission timeouts. This is also the reason why the throughput of the connection with the 512 [Kbyte/sec] input link bandwidth is largely degraded, which can be explained by Equation (6.12).

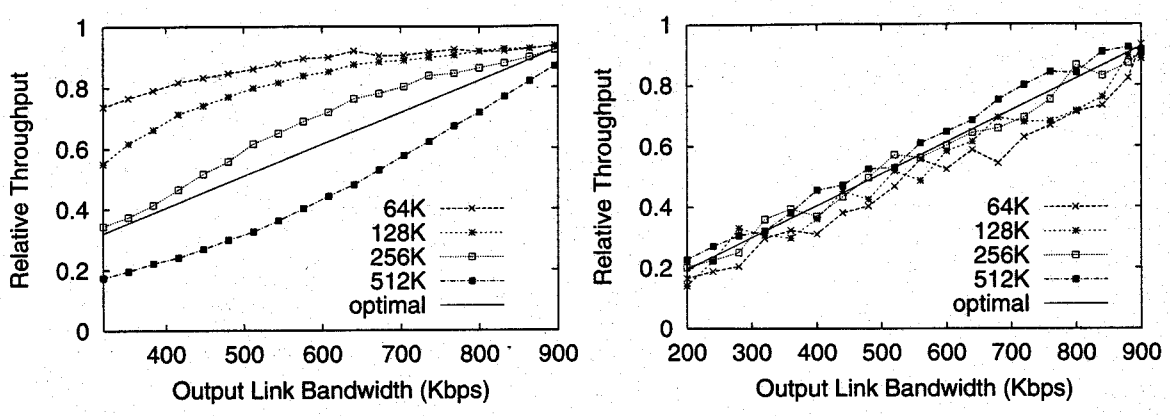
We next consider the enhancement to the RED algorithm (called *enhanced RED*) to avoid this unfairness by setting p dependently on each connection's input link capacity, according to the analysis results. We set p_i, which is the packet dropping probability of connection i, such that each connection's throughput becomes proportional to the its input link capacity. The appropriate values p_i's are calculated for all connections as follows.

1. Initialize p_i's.
2. Calculate ρ_i from the current p_i according to the analysis results. See Equation (6.12).
3. If ρ_i is proportional to the input link capacity, set p_i to the current value.
4. If not, compare ρ_i with the ideal value, and adjust p_i of the connection having the largest difference between ρ_i and the ideal value. That is,
   - If ρ_i is larger than the ideal value, change p_i to $a\,p_i$.
   - If ρ_i is smaller than the ideal value, change p_i to $b\,p_i$.

   The typical values of control parameters a and b are 1.1 and 0.9.

In the enhanced RED algorithm, we calculate the p_i's for all connections from the connections' input link capacities according to this algorithm, and set p_i as the packet dropping probability at the RED router in advance of starting to send the packets.

Figure 6.9 shows the simulation results on the relative throughput of the enhanced RED algorithm. Compared with Figure 6.5(b), it is clear that our enhanced version of RED algorithm gives further better fairness than the original RED algorithm. In simulation, however, we set the control parameter values of a and b intuitively. It is a future research topic to seek an appropriate method to determine those parameters.



(a) Original RED. (b) Enhanced RED.

Figure 6.9: Effect of Enhanced RED

6.2.3 DRR Case

As explained in Subsection 6.1.1, the router buffer is logically divided into several queues in DRR and each connection is assigned its own queue. We first consider the case where the large buffer is equipped with the router so that every connection is given a sufficient amount of buffer. In our model depicted Figure 6.10, four DRR queues are formed in the router, and DRR parameters are set such that each DRR queue is served in proportion to the input link capacity of the assigned connection.

Figure 6.12(a) shows the simulation results of relative throughput. Different from the FIFO (Figure 6.4) and RED (Figure 6.5) algorithms, the DRR algorithm provides very good fairness among connections even when the output link capacity is small. When the output link is large, on the other hand, the degree of the fairness is slightly degraded. It is because TCP's retransmission timeouts tends to frequently occur due to bursty packet loss at the queue since the FIFO discipline is used in each DRR queue. Then, the retransmission timeout degrades the performance more seriously. Thus the degree of performance degradation depends on the bandwidth-delay product of the connection. Furthermore, in the DRR algorithm, the capacity not used by a certain queue due to connection's retransmission timeout can be used by other connections. It increases the total throughput, but it is likely to lead to the unfairness among connections. This is why fairness is degraded in the case of the large output link.

While the DRR algorithm assigns the DRR queues to each connection, several connections should be assigned to one DRR queue as the number of connections grows. It is because the number of DRR queues which can be prepared must be limited by the router buffer size and processing overhead. However, the performance of the DRR algorithm in such a case has not been known. For investigating such an insufficient buffer

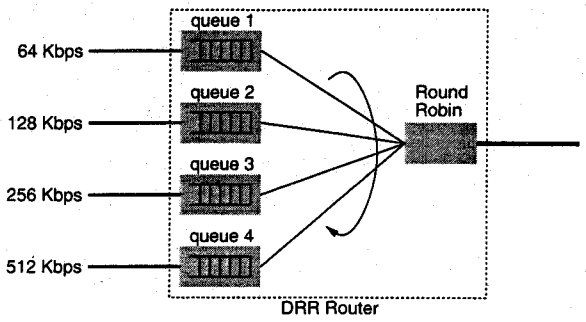


Figure 6.10: DRR Model for Sufficient Buffer Case

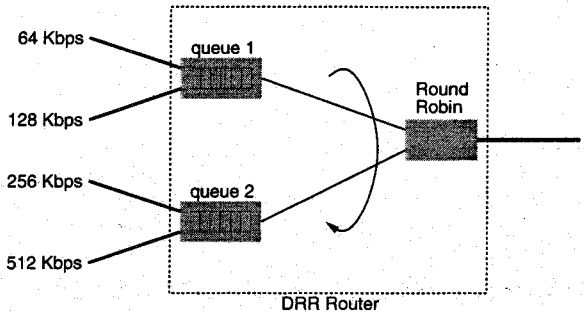
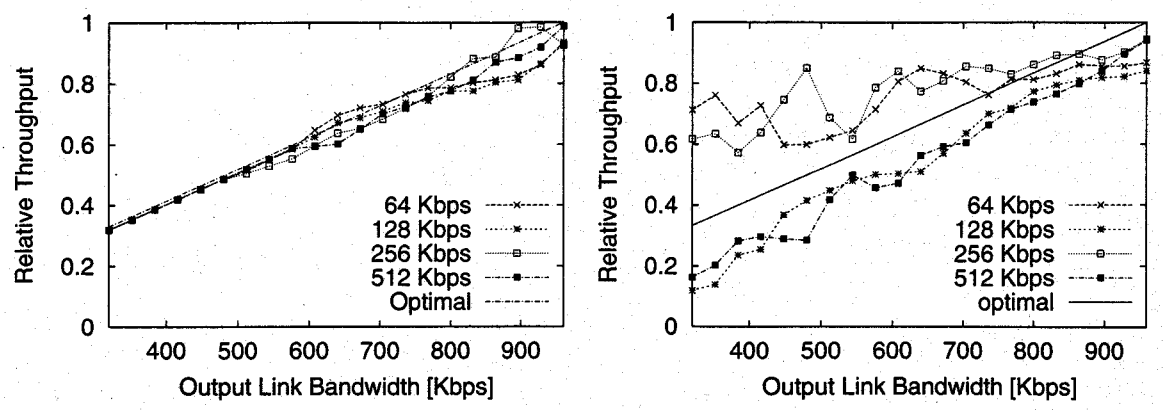


Figure 6.11: DRR Model for Insufficient Buffer Case

case, we assume that there are two queues and four connections, and each connection is assigned to the queue as shown in Figure 6.11. The 64 Kbps and 128 Kbps connections are assigned to one queue (queue 1 in the figure) and the 256 Kbps and 512 Kbps connections to another queue (queue 2). Each queue is assumed to be served in proportion to the total capacity of the assigned connections.

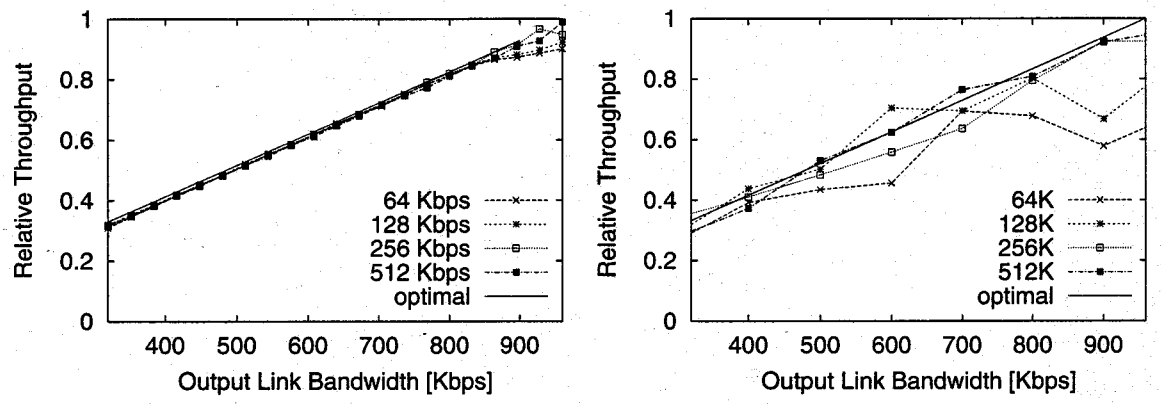
We show the simulation results in the insufficient buffer case in Figure 6.12(b) for the relative throughput. The buffer sizes of two queues are equivalently set to be 30 Kbytes. The lines labeled "total-1" and "total-2" indicate total throughput of two queues, queue 1 and queue 2. Although each queue is served in proportion to the total capacity of the assigned connections, the two connections assigned to the same queue show unfair throughput. This is because we assumed that the arriving packets are served according to a simple FIFO discipline within the DRR queue. As described in Subsection 6.2.1, the FIFO algorithm cannot keep fairness among connection at all.

In this Subsection, we have observed that the DRR algorithm gives much better fairness than FIFO and RED algorithms, but its fairness property is sometimes lost as each connection has different capacity or when multiple connections are assigned to one DRR queue. We henceforth consider to improve the fairness property of the DRR algorithm in the next Subsection.



(a) Sufficient Buffer Case (b) Insufficient Buffer Case

Figure 6.12: DRR Case with TCP Reno



(a) Sufficient Buffer Case (b) Insufficient Buffer Case

Figure 6.13: DRR+ Case with TCP Reno

6.2.4 DRR+ Case

In the previous Subsection, we have shown that the DRR algorithm has some unfairness property. The main reason was that each DRR queue serves packets by the FIFO discipline. In this Subsection, we show some simulation results of DRR+, where the RED algorithm is applied to each DRR queue to prevent unfairness. In simulation, we consider both sufficient/insufficient buffer case. Note that, in the insufficient buffer case, we apply the enhanced RED algorithm to two DRR queues depicted in Figure 6.11. That is, in each queue, we set the assigned connections' packet dropping probabilities according to the enhanced RED algorithm in Subsection 6.2.2.

Figure 6.13 shows the simulation results on the relative throughput. Our proposed method keeps good fairness in the sufficient buffer case (Figure 6.13(a)). Furthermore, when Figure 6.13(b) is compared with Figure 6.12(b), the fairness is significantly improved even in the insufficient buffer case. This results show that our enhanced RED

mechanism can be applied to the DRR+ queue, as well as in RED algorithm.

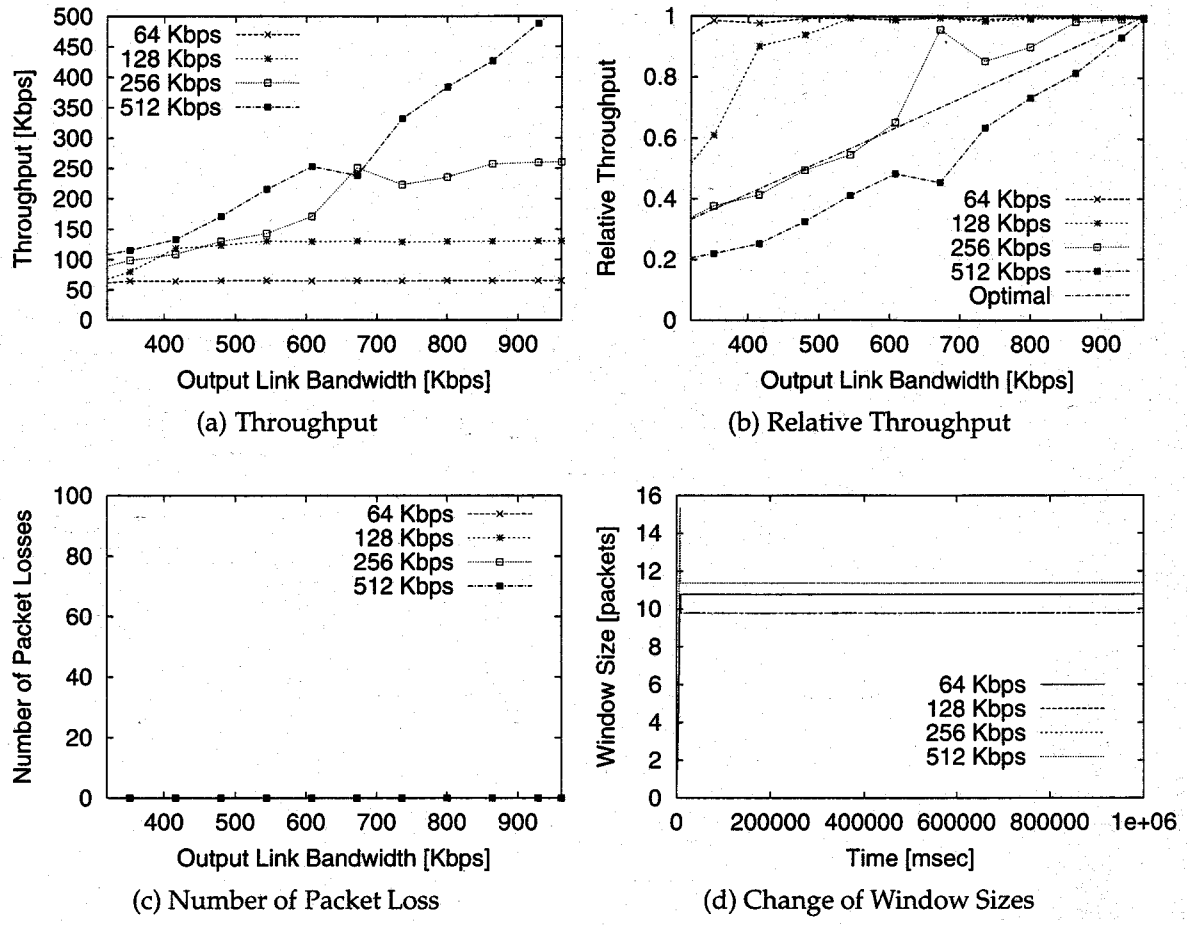
Since this improvement due to introducing RED algorithm corresponds to that without the DRR algorithm (Subsection 6.2.2), the fairness of DRR+ is not so good as the output link capacity becomes small as explained in Subsection 6.2.2. We may have to consider a more effective method to keep fairness as one of the future works. As explained in the analysis of Subsection 6.2.2, the unfairness property of the RED algorithm is caused by the equal packet drop probability independently on the input link capacity of the connections. Therefore, we can set packet dropping probability of RED algorithm dependently on the connection's input link capacity to improve the fairness of the DRR+ algorithm.

6.3 Case of TCP Vegas

In this Section, we change the version of TCP to TCP Vegas to investigate the fairness property of three packet scheduling algorithms. TCP vegas conjectures the available bandwidth for the connection, and therefore its principle is likely to be well fit to the DRR algorithm. On the other hand, the RED algorithm does not help improve the fairness when TCP Vegas is employed since each connection's window size is not dominated by the packet dropping probability of the RED algorithm, but by the essential algorithm of TCP Vegas. The purpose of this Section is to confirm the above observations.

6.3.1 FIFO Case

Figure 6.14 plots simulation results of the FIFO case using TCP Vegas. Note that we omit the graph showing the number of packet loss since no packet loss was observed at the FIFO buffer. Compared with the TCP Reno case (Figure 6.4), it is clear that TCP vegas provides less fairness than TCP Reno. Especially, the connection with has smaller input link bandwidth achieve almost 100% throughput (Figure 6.14(b)). This unfairness property is caused by the essential characteristic of TCP Vegas. In TCP Vegas, no packet loss occurs at the router buffer if the network is stable, because the window size of all connection converges to certain values (Figure 6.14(d)). In Figure 6.14(d), it is noticeable that the converged window size is independent on each connection's input link bandwidth because *base_rtt* of each connection is almost equal. In the current simulation setting, the converged window size is enough large for connections having smaller input link bandwidth to utilize its *bandwidth-delay product*, but it is too small for connections with larger input link bandwidth. Therefore, while the result depends on the network



(a) Throughput (b) Relative Throughput

(c) Number of Packet Loss (d) Change of Window Sizes

Figure 6.14: FIFO Case with TCP Vegas

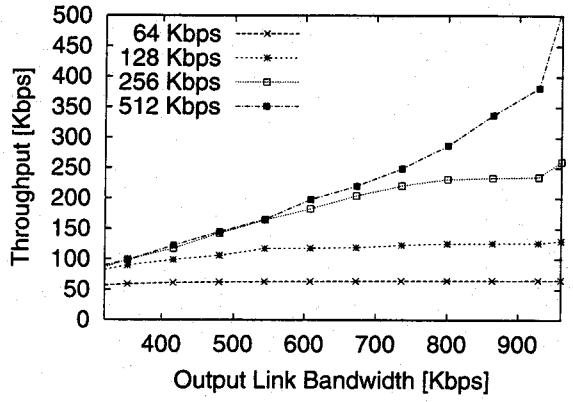
environment, TCP Vegas sometimes fails to achieve fairness among connections due to the essential nature of its congestion control mechanism.

6.3.2 RED Case

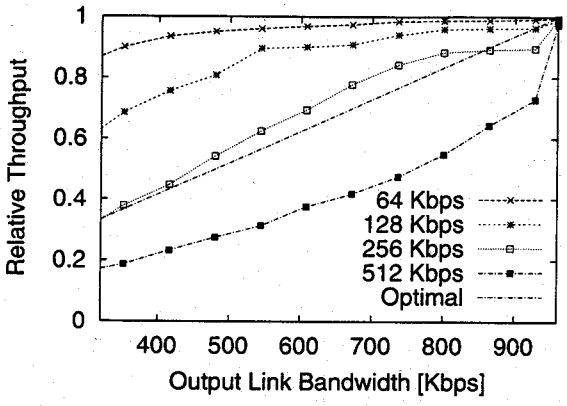
We next show the simulation results of the RED case in Figure 6.14. As in the case of TCP Reno (Subsection 6.2.2), the fairness is slightly improved when compared with the FIFO case (Figure 6.14(b)). However, there still be significant unfairness among connections. This can be explained by the throughput analysis presented in the below. In the following analysis, we use the same notations as those introduced in Subsection 6.2.2.

At the moment, we consider the situation where no packet loss occurs at the router, and each connection's window size converges to a certain value. The packet dropping of the RED will be considered later.

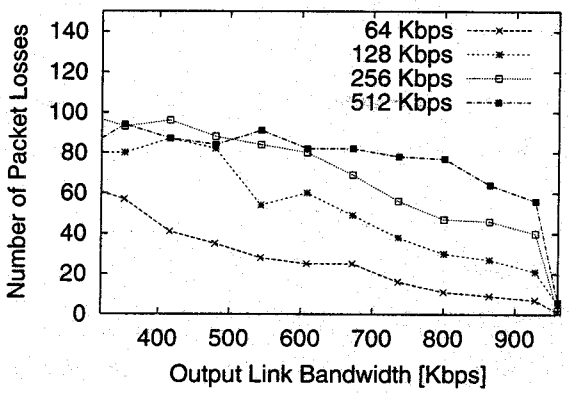
Let l_i [packets] be the number of connection i's packets in the router buffer, and $L = l_1 + \cdots + l_N$. Assume that each connection's throughput ρ_i [packets/sec] is proportional



(a) Throughput



(b) Relative Throughput



(c) Number of Packet Losses

Figure 6.15: RED case with TCP Vegas

to the avarage number of its packets in the router buffer. This assumption is reasonable when the FIFO discipline is applied at the router buffer. Then, the following equation with respect to ρ_i is satisfied;

$$\rho_i = \min\left(bw_i, (l_i/L)BW\right) \tag{6.13}$$

According to the algorithm of TCP Vegas (Equation (1.4)), we obtain;

$$\frac{\alpha}{base\_rtt_i} < \frac{W_i}{base\_rtt_i} - \frac{W_i}{rtt_i} < \frac{\beta}{base\_rtt_i} \tag{6.14}$$

$$base\_rtt_i = 2\tau + 1/BW \tag{6.15}$$

$$rtt_i = 2\tau + l_i/\rho_i \tag{6.16}$$

$$W_i = 2\tau\rho_i + l_i = rtt_i \cdot \rho_i \tag{6.17}$$

where rtt_i [sec] and W_i [packets] are the RTT and the window size of the connection i, respectively. $base\_rtt_i$ [sec] corresponds to $base\_rtt$ of connection i, which is the minimum value of RTTs of the connection. By substituting Equations (6.15)-(6.17) into Equation (6.14), we obtain the following equation;

$$\alpha + \rho_i/BW < l_i < \beta + \rho_i/BW \tag{6.18}$$

From Equation (6.18), L (=$l_1 + \cdots + l_N$) can be calculated as follows;

$$\begin{aligned} N\alpha + \sum_{j=1}^{N} \frac{\rho_i}{BW} < l_1 + \cdots + l_N < N\beta + \sum_{j=1}^{N} \frac{\rho_i}{BW} \\ N\alpha + \sum_{j=1}^{N} \frac{\rho_i}{BW} < L < N\beta + \sum_{j=1}^{N} \frac{\rho_i}{BW} \end{aligned} \tag{6.19}$$

Recalling that $bw_1 \leq bw_2 \leq \ldots \leq bw_N$, Equation (6.13) yields

$$\rho_i = \begin{cases} bw_i & 1 \leq i \leq M \\ (l_i/L)BW & M+1 \leq i \leq N \end{cases} \tag{6.20}$$

Then, from Equations (6.18)-(6.20), we obtain ρ_i for $M+1 \leq i \leq N$ as follows;

$$\rho_i = \frac{l_i}{L - \sum_{j=1}^{M} l_i} \left(BW - \sum_{j=1}^{M} \rho_i \right), \quad M+1 \leq i \leq N \tag{6.21}$$

Therefore, W_i, which is the converged window size of connection i, can be obtained by substituting Equation (6.18) and (6.21) to Equation (6.17).

In the above derivation, however, we do not take account of random packet losses adopted in the RED algorithm. We next consider the effect of throughput degradation caused by probabilistic packet loss of the RED algorithm. Although each connection's window size is controlled to be converged to a certain value in TCP Vegas, it is sometimes decreased by packet loss by the RED algorithm. We assume that the packet loss can be detected by the fast retransmit algorithm. Then, if the packet loss occurs after the window size reaches W_i, the window size is halved to $W_i/2$. That is, if $W_i/2 < 2\tau\rho_i$, the throughput is degraded until the window size reaches $2\tau\rho_i$. In Figure 6.16, we define "one cycle" to be the time duration between two packet losses caused by RED. One cycle is divided into three phases; phase 1, phase 2, and phase 3 as in Figure 6.16. In phase 1, the window size is increasing according to the TCP Vegas' algorithm, but the

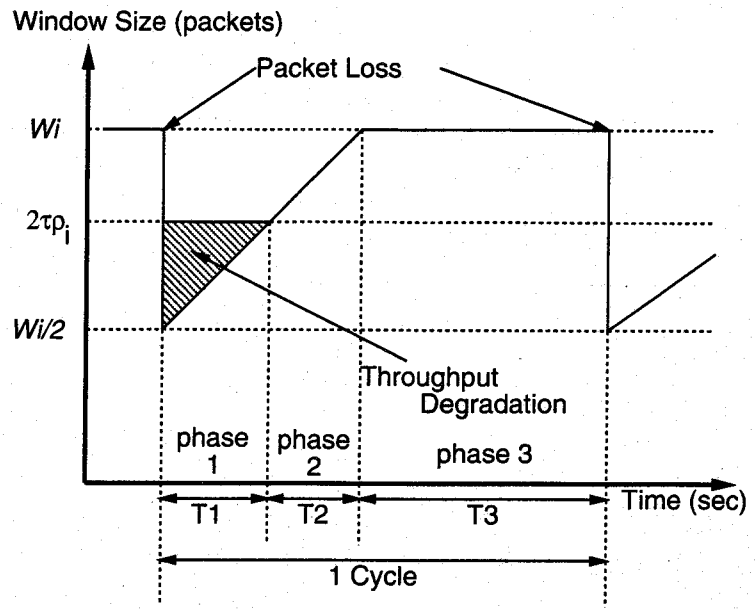


Figure 6.16: Throughput Degradation with RED Packet Loss

window size is less than $2\tau\rho_i$. That is, the throughput is degraded by the packet loss during phase 1. In phase 2, the window size continues to increase as in phase 1, but the window size is larger than $2\tau\rho_i$ and there is no throughput degradation. In phase 3, the window size reaches the converged value, which is obtained from Equation (6.17). It remains unchanged until the packet loss occurs at the end of this phase.

Let T_i [sec] and A_i [packets] be the time duration of phase i, and the number of transmitted packets in phase i, respectively. Furthermore, we introduce $\overline{\rho}_{i,j}$ [packets/sec] as the avarage throughput of connection i during phase j.

In phase 1 and phase 2, the ratio of window size increasing is $1/rtt_i$ [packets/sec] because the window size is increased according to TCP Vegas' congestion avoidance algorithm formulated by Equation (1.4). Therefore, $\overline{\rho}_{i,1}$ is;

$$\overline{\rho}_{i,1} = \left(\frac{\frac{W_i}{2} + 2\tau\rho i}{2}\right) \Big/ \left(2\tau + \frac{1}{\rho_i}\right) \tag{6.22}$$

Because there is no throughput degradation in phase 2 and phase 3, $\overline{\rho}_{i,2}$ and $\overline{\rho}_{i,3}$ are identical to ρ_i, i.e.,

$$\overline{\rho}_{i,2} = \overline{\rho}_{i,3} = \rho_i \tag{6.23}$$

Since the increased rate of window size is $1/rtt_i$ [packets/sec], T_1 and T_2 can be calcu-

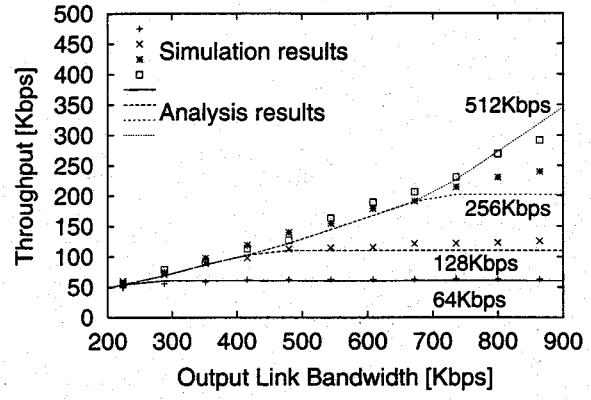


Figure 6.17: Accuracies of Analysis Result in TCP Vegas

lated as follows;

$$T_1 = \left(2\tau\rho_i - \frac{W_i}{2}\right) \cdot rtt_i \tag{6.24}$$

$$T_2 = (W_i - 2\tau\rho_i) \cdot rtt_i \tag{6.25}$$

A_1 and A_2 can also be calculated as follows;

$$A_1 = \frac{1}{2}\left(2\tau\rho_i + \frac{W_i}{2}\right)\left(2\tau\rho_i - \frac{W_i}{2}\right) \tag{6.26}$$

$$A_2 = \frac{1}{2}\left(W_i + 2\tau\rho_i\right)\left(W_i - 2\tau\rho_i\right) \tag{6.27}$$

In phase 3, the window size is converged to W_i, and packet loss occurs at the router caused by the RED algorithm at the end of this phase. Since the avarage number of transmitted packets during 1 cycle is $(1/p)$, A_3 and T_3 can be obtained as;

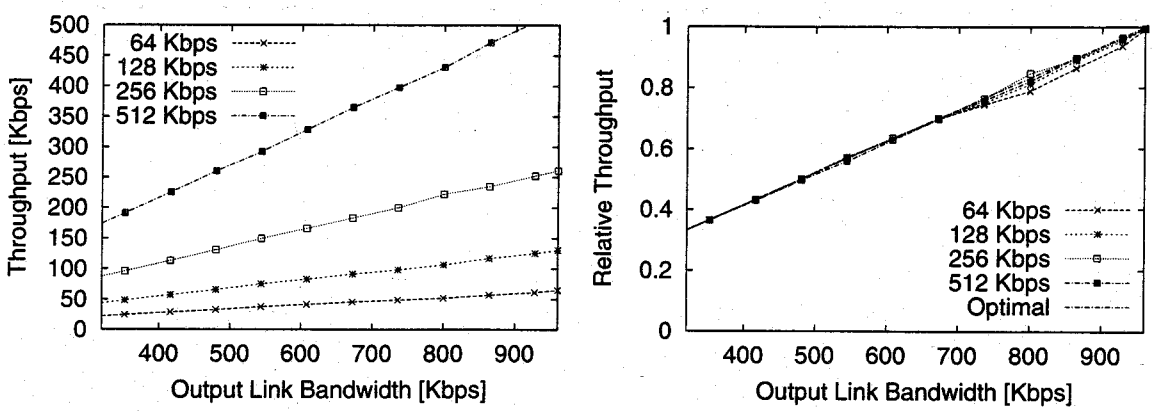
$$A_3 = 1/p - A_1 - A_2 \tag{6.28}$$

$$T_3 = (A_3/W_i) \cdot rtt_i \tag{6.29}$$

Finally, we can obtain $\hat{\rho}_i$, the throughput of connection i from Equations (6.22)-(6.25), (6.29) as follows;

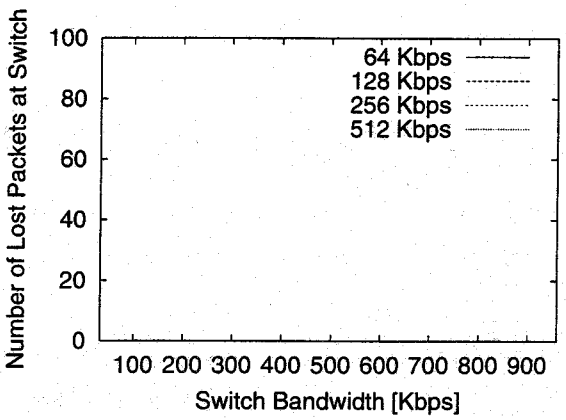
$$\hat{\rho}_i = \frac{T_1\overline{\rho}_{i,1} + T_2\overline{\rho}_{i,2} + T_3\overline{\rho}_{i,3}}{T_1 + T_2 + T_3} \tag{6.30}$$

Figure 6.17 shows the result of the analysis as a function of the output link capacity. Compare with Figure 6.8. Our analysis again gives good agreements with simulation results, and it confirms the unfairness property of TCP Vegas when applied to the RED



(a) Throughput

(b) Relative Throughput



(c) Number of Packet Losses

Figure 6.18: DRR case with TCP Vegas

algorithm. In TCP Reno (Subsection 6.2.2), we could improve the fairness by setting p (the packet dropping probability) dependently on each connection's input link capacity according to the analysis results. In TCP Vegas, however, we cannot apply it because the converged window size is independent on p as shown in Equation (6.17). That is, we cannot control each connection's throughput by p. Therefore, if we want to remove the unfairness property in the RED algorithm with TCP Vegas, we may have to give some modifications to the algorithm of TCP Vegas itself. Otherwise, we need to use the DRR algorithm as will be presented in the next Subsection.

6.3.3 DRR Case

Figure 6.18 shows the case of DRR. It can be observed from the figure that fairness among connections is fairly good (Figure 6.18), and better than TCP Reno case (Figure 6.12(a)). With TCP Reno, some connections could not utilize all amount of bandwidth assigned by the DRR mechanism due to packet loss. With TCP Vegas, on the other hand, no

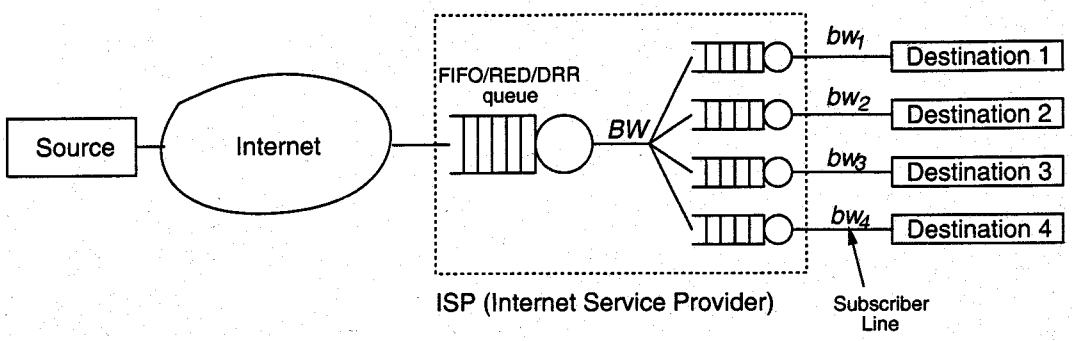


Figure 6.19: Network Model for Reverse Traffic

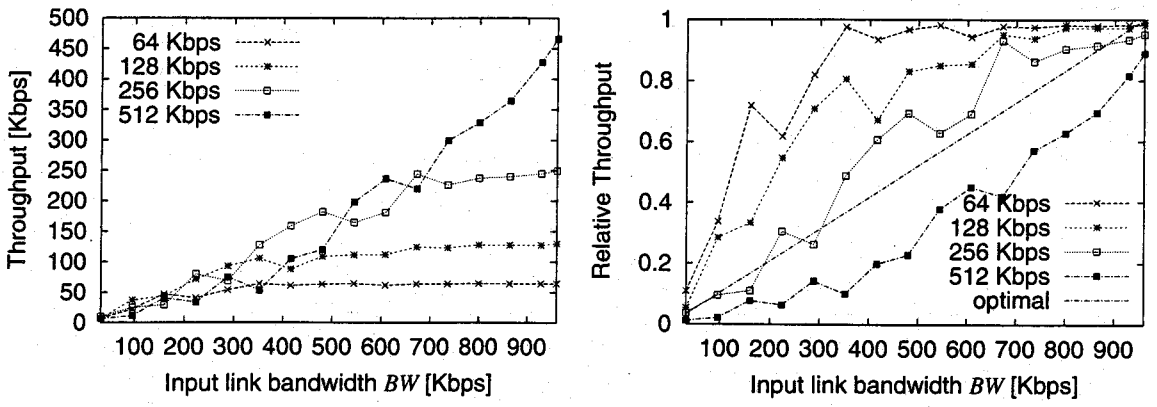
packet loss occurs at the router buffer, and then each connection can completely utilize the bandwidth assigned by the DRR mechanism. However, as the number of connections becomes large, the scalability problem is introduced as having been explained in Subsection 6.2.3. In Subsection 6.2.4, we have succeeded to avoid the unfairness by applying the RED mechanism to each DRR queue. In the current case, however, we cannot apply it because of the essential incompatibility of TCP Vegas to the RED algorithm as explained in Subsection 6.3.2. We need further investigation on this problem.

6.4 Effect of Reverse Traffic

In this section, we investigate the effect of the reverse traffic. That is, the downlink of the access line of the ISP is shared by the subscribers with different capacities as opposed to the previous case where the uplink of the access line is shared. The purpose of this section to confirm the applicability of the discussion and analysis described in Section 6.2 to the reverse traffic. We use the network model depicted in Figure 6.19, where the number of connection is 4. The input link bandwidth is BW [packets/sec], and the output link bandwidth of connection i is bw_i [packets/sec]. As in Section 6.2, we consider FIFO, RED and DRR algorithms at the bottleneck queue. In this section, we use TCP Reno version, and compare the results with those presented in Section 6.2 to investigate the effect of the traffic direction.

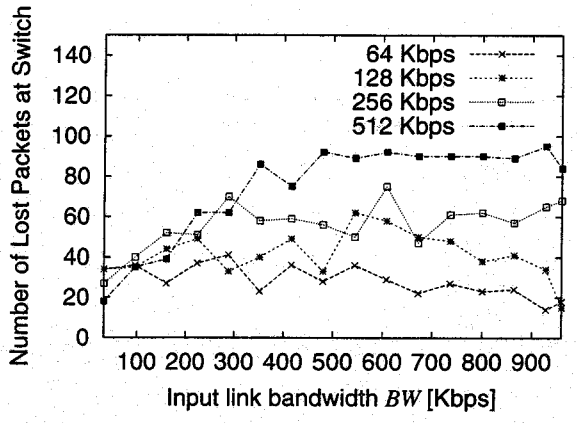
6.4.1 FIFO Case

Figure 6.20 shows the simulation results of the FIFO case. The fairness characteristic is very similar to the previous case shown in Figure 6.4. The network model shown in Figure 6.19 has the same bottleneck point as in Figure 6.3, which is shared by four connections having the different link bandwidths. Therefore, the characteristics of packet



(a) Throughput

(b) Relative Throughput



(c) Number of Packet Loss

Figure 6.20: FIFO Case with Reverse Traffic

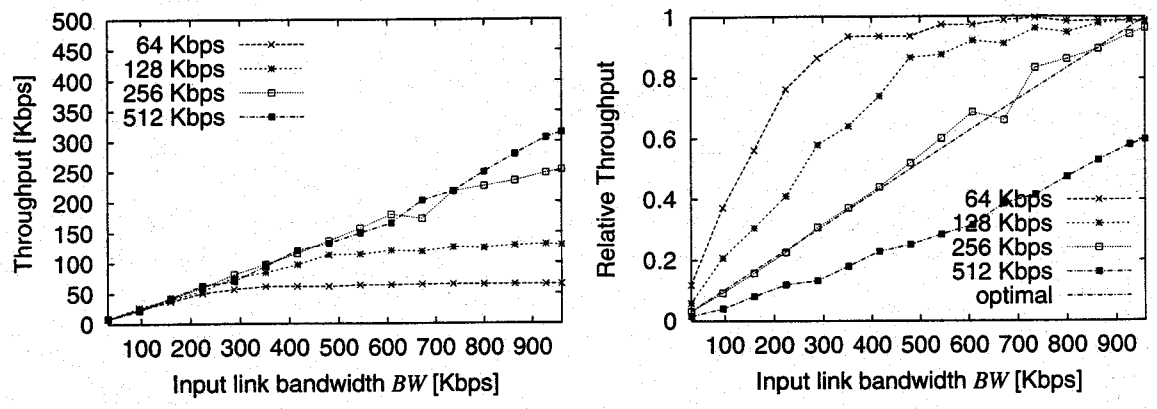
loss at the bottleneck queue becomes similar in the case of Subsection 6.2.1.

6.4.2 RED Case

The simulation results are shown in Figure 6.21, where the RED algorithm is applied at the bottleneck queue. The figure clearly exhibits that the RED algorithm can not provide fairness, which is a same tendency with the previous case in Section 6.2.2 (Figure 6.5).

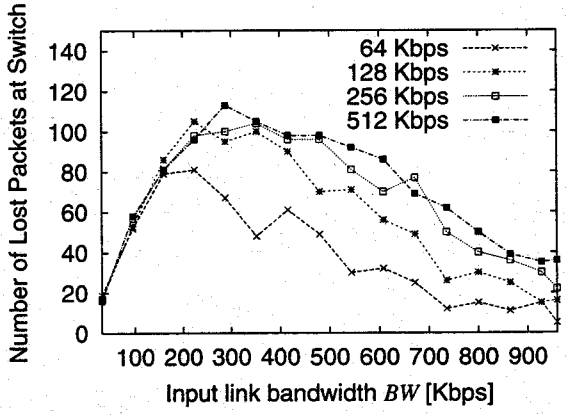
In Section 6.2.2, we have derived the throughput of each connection with TCP Reno and the RED algorithm using the network model depicted in Figure 6.3 through analysis approach. Since the network model in this subsection 6.19 has the same bottleneck point, the analysis in Section 6.2.2 can also be applied to the network model of reverse traffic. This applicability can be proved by comparing Figure 6.21 with Figure 6.5, which shows the similar characteristics in terms of the fairness.

To explain the applicability of our analysis results more clearly, we also tested the



(a) Throughput

(b) Relative Throughput



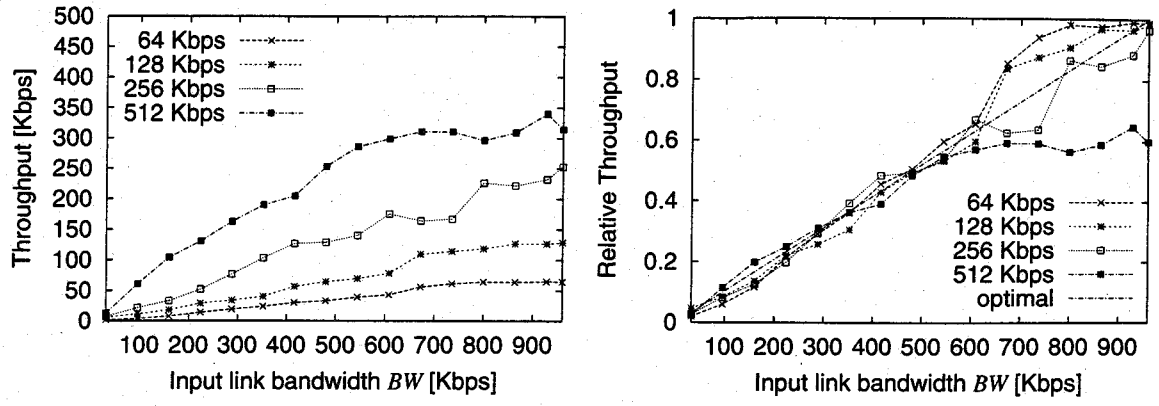
(c) Number of Packet Loss

Figure 6.21: RED Case with Reverse Traffic

Enhanced RED algorithm that we described in Subsection 6.2.2. To improve the fairness of the RED router, we set the packet dropping probability of each connection according to the algorithm in Subsection 6.2.2. The simulation result is depicted in Figure 6.22. The fairness improvement is fairly good (Figure 6.22(b)), which indicates the robustness of our proposed algorithm.

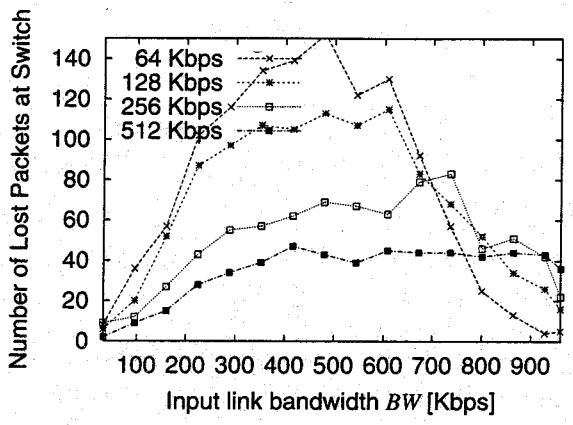
6.4.3 DRR Case

Figure 6.23 shows the simulation results of the DRR case. As is the case of FIFO and RED, the results again shows the similar tendency with the previous case in Section 6.2.3 (Figure 6.12). This also shows that the model depicted in Figure 6.19 can be dealt in the same way as that in Figure 6.3.



(a) Throughput

(b) Relative throughput

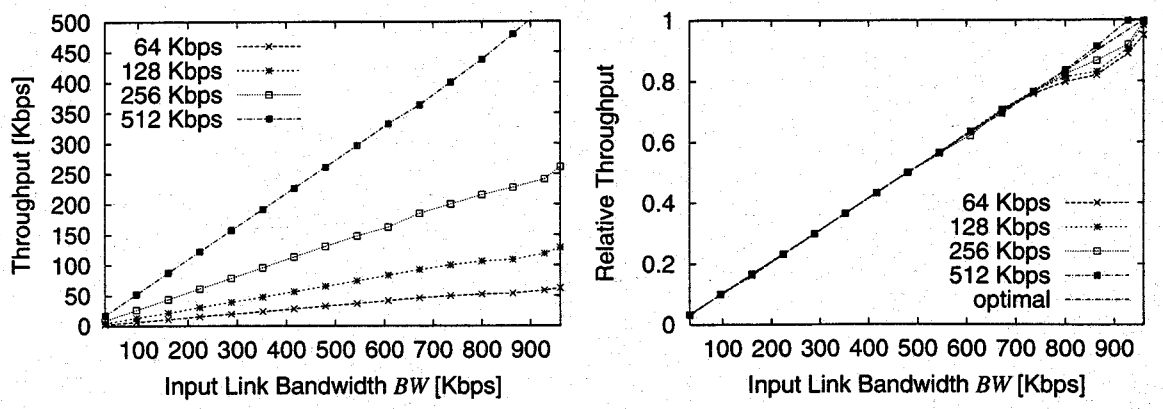


(c) Number of Packet loss

Figure 6.22: Enhanced RED case with Reverse traffic

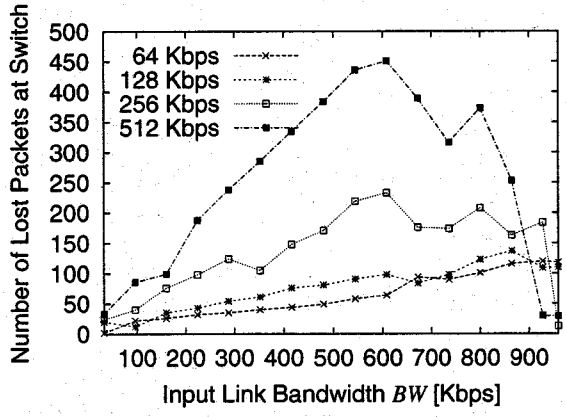
6.5 Conclusion

In this Chapter, we have evaluated the performance of the router packet scheduling algorithms for fair service among connections through the simulation and analysis. We have obtained the following results on TCP Reno version; the FIFO algorithm cannot keep fairness among connections at all. The RED algorithm can improve fairness to some degree, but it fails to keep fairness in the different capacity case. The DRR algorithm offers better fairness than the FIFO algorithm and the RED algorithm, but its fairness property is lost when each connection has different capacity and/or when multiple connections are assigned to one DRR queue. Accordingly, we have proposed the DRR+ algorithm, where the RED algorithm is applied to each DRR queue to prevent unfairness, and show that it can improve fairness among connections in the different capacity case. We have also investigated the effect of TCP Vegas, which is expected to get higher throughput than TCP Reno, and have made clear through the simulation and analysis results that TCP Vegas cannot help improving the fairness among connections in FIFO



(a) Throughput

(b) Relative Throughput



(c) Number of Packet Loss

Figure 6.23: DRR Case with Reverse Traffic

and RED cases.

TCP Vegas has a good feature to attain the better performance than TCP Reno, as discussed in Section 6.3, it fails to keep the good fairness among the connections with different input (and output) line capacities. For TCP Vegas to be introduced in the future Internet where the RED algorithm is widely deployed, the algorithm of TCP Vegas should be modified in order to improve the fairness among connections, which is a future research topic.

Chapter 7

Conclusion

TCP is very traditional transport-layer protocol which was first designed in early 1970s, and many efforts of researchers, developments and standardization have been extensively devoted to the TCP technology. TCP is now one of the most popular protocols in the current Internet, and it is used by many of popular Internet services including WWW using HTTP, file transfer service using FTP and e-mail transfer service using SMTP. Therefore, even if network infrastructure may significantly change in the future Internet, TCP and its applications would be continuously used. This observations have made many researchers study actively about TCP, and many papers have been published in the literature.

One of the most important issues regarding TCP is its congestion control mechanisms, and most of past researches focused on it. However, there remains many big problems which have not been solved yet; Can TCP be applied to the future high-speed network without any changes? If no, how it should be changed to adapt to the next-generation Internet? Is a new transport-layer protocol necessary?

Furthermore, the evolution of network technologies have derived new demands from the Internet users. That is, we now need to provide more complicated network services than simple best-effort services which is provided in the current Internet. Those services it to provide *fair* services among connections, or provide *differentiated* services to each connection. Therefore, it is important to investigate whether those new services can be served with existing TCP/IP technology. Consequently, many active researchers have focused on applicabilities of TCP to the future high-speed network, but most of past studies have concentrated only on high speed data transfer using TCP. As well as effectiveness, however, stability and fairness are also important issues for future commercial usage of the Internet.

In this thesis, we have first focused on evaluating the essential characteristics of TCP, and have investigated the nature of congestion control mechanisms of TCP through

mathematical analysis approach. In Chapter 2, we have considered simple network model where two connections share the bottleneck link, and have analyzed their behavior to evaluate TCP's essential property in terms of *fairness* and *stability* of congestion control mechanisms. We have obtained the results that TCP Tahoe and TCP reno, which are used in the current Internet, lack stability of their window sizes, and that TCP Vegas sometimes fails to keep fairness among connections. From there results, we have proposed the enhanced version of TCP Vegas, which has better stability than TCP Tahoe and Reno, and better fairness than the original TCP Vegas.

Next, in Chapter 3, we have looked at the TCP's *mis-retransmission* problem, and analyze the reason mis-retransmission occurs and the degree of performance degradation through mathematical approach. We have found that TCP's mis-retransmission may occur both in the current best-effort Internet, and in the future networks where a fixed amount of bandwidth is assigned to a TCP connection. We have also derived the results that the performance degradation caused by mis-retransmission cannot be ignored, and TCP can avoid it with the minimum change of its congestion control algorithm we have proposed.

We have next investigated TCP's applicability to the future high-speed network, and future new Internet services. First, in Chapter 4, we have focused at the performance of *TCP over ATM* networks for data transmission, since we have wanted to evaluate interaction between the congestion control mechanisms of TCP, and ATM's rate control mechanisms. We have considered TCP over plain UBR service class, TCP over UBR with EPD, and TCP over ABR service class. We have found that TCP and ATM can be co-exist without large problems, if ABR service class is selected at ATM layer, and its rate control parameters are appropriately regulated.

Second, in Chapter 5, we have also investigated the performance of TCP under *asymmetric* networks, which provides *asymmetric* bandwidth for upstream and downstream by new-emerging network services at access network, including ADSL and cable modem technology. The main concern was to make clear the effect of the network asymmetry on TCP performance, and we have obtained the results that large degree of network asymmetry may degrade TCP performance significantly, and it affects also on HTTP performance.

Finally, we have focused on realizing *fair* service among connection at the router buffer, and have compared three packet scheduling algorithm, in terms of fairness among each TCP connection. We have selected FIFO, RED, and DRR algorithms as scheduling discipline, and investigated their ability through mathematical analysis approach. One of the obtained results is that RED can provide fairness among connections, if we set packet dropping probability of each connection according to the analysis results. Even in DRR algorithm, fairness property may sometimes degrades because FIFO discipline

is used at each round robin queue of DRR. However, we have found that RED algorithm is also useful to avoid unfairness in DRR. That is, we can improve fairness among connections at the router buffer without any changes of TCP.

From all results of this thesis, we believe that TCP can survive any dramatical changes of the Internet infrastructure, and that any kind of new services in the advanced Internet can be accommodated by TCP technology. We don't have to replace TCP with a quite new transport-layer protocol.

Abbreviation List

AAL	ATM Adaptation Layer
ABR	Available Bit Rate
ACK	Acknowledgement
ACR	Allowed Cell Rate
ADSL	Asymmetric Digital Subscriber Line
AIR	Additive Increase Rate
AIRF	Additive Increase Rate Factor
ATM	Asynchronous Transfer Mode
BSD	Berkeley Software Distribution
CAC	Call Admission Control
CATV	Cable Television
CBQ	Class-based Queueing
CBR	Constant Bit Rate
CWND	Congestion Window
DES	Destination End System
DRR	Deficit Round Robin
EFCI	Explicit Forward Congestion Indication
EPD	Early Packet Discard(ing)
EPD/A	EPD with per-VC Accounting
ER	Explicit Rate
FDDI	Fiber-Optic Digital Device Interface
FIFO	First In First Out
FRED	Fair Random Early Drop
FTP	File Transfer Protocol
HTTP	Hyper Text Transfer Protocol
ICR	Initial Cell Rate
IP	Internet Protocol
ISDN	Integrated Services Digital Network
ISP	Internet Service Provider
LAN	Local Area Network

MCR	Minimum Cell Rate)
MPEG	Motion Picture Expert Group
PCR	Peak Cell Rate
QoS	Quality of Service
RDF	Rate Decrease Factor
RDFF	RDF Factor
RED	Random Early Detection
RFC	Request For Comments
RIF	Rate Increase Factor
RM	Resource Management
RTO	Retransmission Timeout
RTT	Round Trip Time
SES	Source End System
SMTP	Simple Mail Transfer Protocol
SW	Switch
TCP	Transmission Control Mechanism
TDF	Timeout Decrease Factor
TOF	Time Out Factor
UBR	Unspecified Bit Rate
USD	User Share Differentiation
VBR	Variable Bit Rate
VC	Virtual Connection
WAN	Wide Area Network
WDM	Wavelength Division Multiplexing
WRR	Weighted Round Robin
WWW	World Wide Web
XDF	Xrm Decrease Factor
XTP	Xpress Transport Protocol

Bibliography

[1] J. B. Postel, "Transmission control protocol," *Request for Comments 793*, September 1981.

[2] V. Jacobson, "Congestion avoidance and control," in *Proceedings of ACM SIGCOMM '88*, pp. 314–329, August 1988.

[3] D. D. Clark, V. Jacobson, J. Romkey, and H. Salwen, "An analysis of TCP processing overhead," *IEEE Communications Magazine*, vol. 27, pp. 23–29, June 1989.

[4] 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.

[5] T. V. Lakshman and U. Madhow, "Performance analysis of window–based flow control using TCP/IP: Effect of high bandwidth–delay product and random loss," in *Proceedings of HPN'94*, pp. 135–149, June 1994.

[6] J. Hoe, "Start-up dynamics of TCP's congestion control and avoidance schemes," Master's thesis, MIT, June 1995.

[7] V. Paxson, "Measurements and analysis of end-to-end Internet dynamics," *Ph.D. Thesis*, April 1997.

[8] V. Jacobson and R. Braden, "TCP extensions for long-delay paths," *Request for Comments 1072*, October 1988.

[9] R. Braden, "Requirements for Internet hosts – communication layers," *Request for Comments 1122*, October 1989.

[10] V. Jacobson, R. Braden, and D. Borman, "TCP extensions for high performance," *Request for Comments 1323*, May 1992.

[11] D. E. McDysan and D. L. Spohn, *ATM Theory and Application*. McGraw-Hill, 1994.

[12] S. S. Sathaye, "ATM forum traffic management specification version 4.0," *ATM Forum Contribution 0056.000*, April 1996.

[13] L. Kleinrock, "The latency/bandwidth tradeoff in gigabit networks," *IEEE Communications Magazine*, vol. 30, pp. 36–41, April 1992.

[14] 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," in *Proceedings of IEEE INFOCOM'97*, pp. 1201–1211, April 1997.

[15] W. chang Feng, D. D. Kandlur, D. Saha, and K. G. Shin, "Understanding TCP dynamics in an integrated services Internet," in *Proceedings of The 7th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'97)*, pp. 375–385, May 1997.

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

[17] Diffserv Home Page, available from `http://diffserv.lcs.mit.edu/`.

[18] D. D. Clark and W. Fang, "Explicit allocation of best effort packet delivery service," available from `http://diffserv.lcs.mit.edu/Papers/exp-alloc-ddc-wf.ps`, 1998.

[19] Z. Wang, "Toward scalable bandwidth allocation on the Internet," *On The Internet*, pp. 24–32, May/June 1998.

[20] S. Floyd and V. Jacobson, "Link-sharing and resource management models for packet networks," *IEEE/ACM Transactions on Networking*, vol. 3, pp. 365–386, August 1995.

[21] M. Katevenis, S. Sidiropoulos, and C. Courcoubetis, "Weighted round–robin cell multiplexing in a general–purpose ATM switch chip," *IEEE Journal on Selected Areas in Communications*, vol. 9, pp. 1265–1279, October 1991.

[22] M. Shreedhar and G. Varghese, "Efficient fair queuing using deficit round robin," *IEEE/ACM Transactions on Networking*, vol. 4, pp. 375–385, June 1996.

[23] XTP Home Page, available from `http://www.ca.sandia.gov/xtp/xtp.html`.

[24] R. Ahuja, S. Keshav, and H. Saran, "Design, implementation, and performance of a native mode ATM transport layer," in *Proceedings of IEEE INFOCOM '96*, pp. 206–214, March 1996.

[25] L. S. J., M. K. McKusick, M. J. Karels, and J. S. Quarterman, *The Design and Implementation of the 4.3BSD UNIX Operating System*. Reading, Massachusetts: Addison-Wesley, 1989.

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

[27] S. Floyd and T. Henderson, "The NewReno modification to TCP's fast recovery algorithm," *Request for Comments 2582*, April 1999.

[28] 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.

[29] 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.

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

[31] G. Hasegawa, M. Murata, and H. Miyahara, "Fairness and stability of congestion control mechanisms of TCP," *The Transactions of IEICE* (in Japanese), vol. J82-B, pp. 1646–1654, September 1999.

[32] G. Hasegawa, M. Murata, and H. Miyahara, "Fairness and stability of congestion control mechanisms of TCP," to appear in *Telecommunication Systems Journal*, 2000.

[33] 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.

[34] G. Hasegawa, M. Murata, and H. Miyahara, "Fairness and stability of congestion control mechanisms of TCP," in *Proceedings of 11th ITC Specialist Seminar*, pp. 255–262, October 1998.

[35] G. Hasegawa, M. Murata, and H. Miyahara, "Fairness and stability of congestion control mechanisms of TCP," *Technical Report of IEICE* (IN97-213) (in Japanese), pp. 127–132, March 1998.

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

[37] D. Lin and R. Morris, "Dynamics of random early detection," in *Proceedings of ACM SIGCOMM'97*, pp. 127–137, October 1997.

[38] J. C. Hoe, "Improving the start-up behavior of a congestion control scheme of TCP," *ACM SIGCOMM Computer Communication Review*, vol. 26, pp. 270–280, October 1996.

[39] 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.

[40] A. S. Tanenbaum, *Computer Networks, 3rd edition.* Upper Saddle Riverm New Jersey, 07458: Prentice Hall, 1996.

[41] D.-M. Chiu and R. Jain, "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks," *Journal of Computer Networks and ISDN Systems*, pp. 1–14, June 1989.

[42] G. Hasegawa, M. Murata, and H. Miyahara, "Improvement of the congestion control mechanism of TCP for ER-based ABR service class in ATM networks," in *Proceedings of 2nd IFIP Workshop on Traffic Management and Synthesis of ATM Networks*, September 1997.

[43] Go Hasegawa and Hiyoyuki Ohsaki and Masayuki Murata and Hideo Miyahara, "Improvement of the congestion control mechanism of TCP to avoid mis–retransmission," *The Transactions of IEICE* (in Japanese), vol. J82-B, pp. 2074–2084, December 1999.

[44] G. Hasegawa, H. Ohsaki, M. Murata, and H. Miyahara, "A study on flow control and error recovery mechanisms of TCP," *Technical Report of IEICE* (SSE97-7) (in Japanese), pp. 1–6, April 1997.

[45] H. Ohsaki, M. Murata, and H. Miyahara, "Designing efficient explicit-rate switch algorithm with max-min fairness for ABR service class in ATM networks," in *Proceedings of IEEE ICC '97*, pp. 182–186, June 1997.

[46] D. H. K. Tsang and W. H. F. Wong, "A new rate-based switch algorithm for ABR traffic to achieve max-min fairness with analytical approximation and delay adjustment," in *Proceedings of IEEE INFOCOM '96*, pp. 1174–1181, March 1996.

[47] H. Ohsaki, G. Hasegawa, M. Murata, and H. Miyahara, "Parameter tuning of rate-based congestion control algorithms and its application to TCP over ABR," in *Pro-*

ceedings of First Workshop on ATM Traffic Management IFIP WG 6.2, pp. 383–390, December 1995.

[48] C. Ikeda, H. Suzuki, G. Hasegawa, M. Murata, and A. Lin, "Is FEC really effective to the data transmission?," *ATM Forum Contribution 95-1481*, December 1995.

[49] G. Hasegawa, H. Ohsaki, M. Murata, and H. Miyahara, "Performance of TCP over ATM networks," in *Proceedings of IEEE GLOBECOM '96*, pp. 1935–1941, November 1996.

[50] G. Hasegawa, H. Ohsaki, M. Murata, and H. Miyahara, "Performance evaluation and parameter tuning of TCP over ABR service in ATM networks," *IEICE Transactions on Communications*, vol. E79-B, pp. 668–683, May 1996.

[51] G. Hasegawa, H. Ohsaki, M. Murata, and H. Miyahara, "Performance evaluation and parameter tuning of TCP over ABR service in ATM networks," *Technical Report of IEICE* (SSE95-117), pp. 7–12, December 1995.

[52] G. Hasegawa, H. Ohsaki, M. Murata, and H. Miyahara, "More results on performance of TCP over ATM networks," *Technical Report of IEICE* (SSE95-170) (in Japanese), pp. 53–60, March 1996.

[53] P. Newman, "Traffic management for ATM local area networks," *IEEE Communications Magazine*, vol. 32, no. 8, pp. 44–51, 1994.

[54] The ATM Forum, *ATM User-Network Interface Specification Version 3.1*. Prentice Hall, 1993.

[55] The ATM Forum Technical Committee, "ATM traffic management specification version 4.0," *ATM Forum Contribution*, April 1996.

[56] Thomas M. Chen, Steve S. Liu, and Vijay K. Samalam, "The available bit rate service for data in ATM networks," *IEEE Communications Magazine*, vol. 34, pp. 56–71, May 1996.

[57] H. Ohsaki, M. Murata, and H. Miyahara, "Robustness of rate-based congestion control algorithm with binary-mode switch in ATM networks," in *Proceedings of IEEE GLOBECOM '96*, vol. 2, pp. 1097–1101, November 1996.

[58] A. Li, L.-Y. Siu, H.-Y. Tzeng, C. Ikeda, and H. Suzuki, "A simulation study of TCP performance in ATM networks with ABR and UBR services," *IEICE Transactions on Communications*, vol. E79-B, pp. 658–667, May 1996.

[59] Y. Chang, N. Golmie, L. Benmohamed, and D. Su, "Simulation study of the new rate-based EPRCA traffic management mechanism," *ATM Forum Contribution 94-0809*, September 1994.

[60] C. Fang and A. Lin, "A simulation study of ABR robustness with binary-mode switches: Part II," *ATM Forum Contribution 95-1328R1*, October 1995.

[61] H. Ohsaki, M. Murata, H. Suzuki, C. Ikeda, and H. Miyahara, "Performance evaluation of rate-based congestion control algorithms in multimedia ATM networks," in *Proceedings of IEEE GLOBECOM '95*, pp. 1243–1248, November 1995.

[62] A. Charny, K. K. Ramakrishnan, and A. G. Lauck, "Scalability issues for distributed explicit rate allocation in ATM networks," in *Proceedings of IEEE INFOCOM '96*, vol. 3, pp. 1198–1205, March 1996.

[63] G. Hasegawa, M. Murata, and H. Miyahara, "Performance evaluation of HTTP/TCP on asymmetric networks," *Proceedings of SPIE'99*, vol. 3530, pp. 423–432, September 1999.

[64] G. Hasegawa, M. Murata, and H. Miyahara, "Performance evaluation of HTTP/TCP on asymmetric networks," *International Journal of Communication Systems*, vol. 12, pp. 281–296, July–August 1999.

[65] G. Hasegawa, M. Murata, and H. Miyahara, "Performance evaluation of HTTP/TCP on asymmetric networks," in *Proceedings of 1998 CQR International Workshop*, May 1998.

[66] G. Hasegawa, M. Murata, and H. Miyahara, "Performance evaluation of TCP on asymmertic network," *Technical Report of IEICE* (CQ97-68) (in Japanese), pp. 69–76, December 1997.

[67] G. Hasegawa, M. Murata, and H. Miyahara, "Performance evaluation of HTTP/TCP on asymmertic network," *Technical Report of IEICE* (IN97-172) (in Japanese), pp. 83–90, February 1998.

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

[69] 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.

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

[71] G. Hasegawa, T. Matsuo, M. Murata, and H. Miyahara, "Comparisons of packet scheduling algorithms for fair service among connections on the Internet," to be presented at *IEEE INFOCOM 2000*, March 2000.

[72] T. Matsuo, G. Hasegawa, M. Murata, and H. Miyahara, "Comparisons of packet scheduling algorithms for fair service among connections," in *Proceedings of Internet Workshop '99*, pp. 193–200, February 1999.

[73] T. Matsuo, G. Hasegawa, M. Murata, and H. Miyahara, "Performance evaluation of router packet scheduling algorithms for fair serivce among connections," *Technical Report of IPSJ*, October 1998.

[74] G. Hasegawa, T. Matsuo, and M. Murata, "Comparisons of packet scheduling algorithms for fair service among connections on the Internet," *The 59th National Conventions of IPSJ*, pp. 27–34, March 1999.

[75] G. Hasegawa, T. Matsuo, M. Murata, and H. Miyahara, "Performance evaluation of packet scheduling algorithms and improvement of RED algorithm for fair service among connections," *Technical Report of IEICE* (IN98-172) (in Japanese), pp. 13–18, March 1999.

[76] G. R. Wright and W. R. Stevens, *TCP/IP Illustrated, Volume 2: The Implementation.* Reading, Massachusetts: Addison-Wesley, 1995.

[77] A. Romanow and S. Floyd, "Dynamics of TCP over ATM networks," *IEEE Journal on Selected Areas in Communications*, vol. 13, pp. 633–641, May 1995.

[78] H. Ohsaki, M. Murata, H. Suzuki, C. Ikeda, and H. Miyahara, "Rate-based congestion control for ATM networks," *ACM SIGCOMM Computer Communication Review*, vol. 25, pp. 60–72, April 1995.

[79] S. Floyd and V. Jacobson, "On traffic phase effects in packet-switched gateways," *Internetworking: Research and Experience*, vol. 3, pp. 397–413, August 1992.

[80] L. Kleinrock, "The latency/bandwidth tradeoff in gigabit networks," *IEEE Communications Magazine*, vol. 30, pp. 36–41, April 1992.

[81] H. Ohsaki, M. Murata, H. Suzuki, C. Ikeda, and H. Miyahara, "Analysis of rate-based congestion control methods in ATM networks, Part 1: steady state analysis," in *Proceedings of IEEE GLOBECOM '95*, pp. 296–303, November 1995.

[82] H. Ohsaki, M. Murata, H. Suzuki, C. Ikeda, and H. Miyahara, "Analysis of rate-based congestion control methods in ATM networks, Part 2: initial transient state analysis," in *Proceedings of IEEE GLOBECOM '95*, pp. 1095–1101, November 1995.

[83] L. G. Roberts, "Computation of Xrm and ICR in ABR signaling," *ATM Forum Contribution 95-0817*, August 1995.

[84] 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.

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

[86] D. P. Heyman, T. V. Lakshman, and L. Neidhardt, "A new method for analyzing feedback-based protocols with applications to engineering web traffic over the Internet," in *Proceedings of ACM SIGMETRICS'97*, pp. 24–38, February 1997.

Biography

Go Hasegawa was born in Himeji City, Hyogo, Japan in 1974. He received M.E. degree from Graduate School of Engineering Science, Osaka University, Osaka, Japan in 1997. He has been a Research Associate of Graduate School of Economics, Osaka University since 1997. He has also been a Research Associate of Graduate School of Informatics and Mathematical Science, Osaka University since 1999. He is a member of IEICE and IEEE.