Studies on Transport Protocol in the Emerging Network Environments

Hiroyuki Koga

Preface

The Internet has become a common infrastructure that includes worldwide networks consisting of many computers. The major protocols in the Internet protocol suite are IP (Internet Protocol) for routing and delivery, and TCP (Transmission Control Protocol) for reliable end-to-end transmission. Network protocols are normally developed in layers, with each layer responsible for a different facet of the communication.

The Internet has been developed to support the character-based transmission mainly used by applications such as e-mail and WWW (World Wide Web) browsing. As the bandwidth has expanded in recent years, multimedia applications like voice and video streaming have become more attractive. Therefore, Internet users desire a multimedia Internet that supports both of real-time communication and non-real-time communications. On the other hand, the growth of mobile networks has created an expectation for Internet access through mobile access networks. However, TCP may not work well over such networks, because improvement of TCP has aimed only at more efficient use of the bandwidth of a link in wired networks where error rates are very low and packet losses are caused mostly by congestion.

The objective of this dissertation is to study transport protocols in such an emerging network environment. In particular, I focus on the throughput performance and flow control mechanisms of TCP in such networks.

Chapter 3 focuses on the performance of TCP in the multimedia Internet, where

the amount of link bandwidth available for TCP connections changes with changing the amount of CBR traffic. If some CBR connections join a link, the amount of bandwidth available for the TCP connection will decrease abruptly and drastically, causing multiple packet loss in the TCP connection and resulting in timeout. That can lead to throughput degradation in the TCP connection. Therefore, I investigate whether the existing TCP variants can adapt their window flow control mechanism in response to changing available bandwidth so as to use it efficiently.

Chapter 4 focuses on the performance of TCP in the mobile Internet. In such networks, the performance of TCP may be severely affected by the high bit error rate that characterizes wireless links, because TCP is tuned to perform well in wired networks where the error rates are very low and packet losses are caused mostly by congestion. Therefore, I clarify the performance of TCP over wireless links in which Selective-Repeat ARQ is performed for recovering lost PDUs of Layer 2 in the links. The ARQ can cause out-of-order delivery of SDUs, which decreases TCP performance. Hence, I also examine the effect of a timeout timer employed to maintain sequence integrity.

In Chapter 5, I develop an effective receiver-based TCP flow control scheme without any modification of TCP senders in the mobile Internet, like in Chapter 4. Multiple retransmission by Layer 2 ARQ can adversely increase the transmission delay time of TCP packets, which results in an unnecessary increase in the RTO of TCP. Furthermore, the link bandwidth assigned to TCP flows can change in response to changing atmospheric conditions for effective use of wireless links. Thus, TCP has to adapt its transmission rate according to the changing available bandwidth. I therefore propose a TCP flow control scheme using Layer 2 information in the wireless link at the mobile station, and discuss the effectiveness of that scheme.

I hope that this dissertation will be helpful for further study in this field.

March 2003

Hiroyuki Koga

Acknowledgements

First of all, I wish to express my sincere appreciation to Professor Yuji Oie of the Kyushu Institute of Technology. His constant encouragement, guidance through this research, invaluable discussions and advice have greatly helped in accomplishing the research. I also thank him for his careful reading of all papers on the research.

I would also like to express my gratitude to Associate Professor Kenji Kawahara of the Kyushu Institute of Technology for his valuable comments, time, and help in completing the research. His steady support has greatly helped my study.

I wish to thank Professor Hiroshi Ochi and Professor Taiichi Otsuji of the Kyushu Institute of Technology for their invaluable discussions and comments. I also would like to thank Assistant Professor Takeshi Ikenaga of the Kyushu Institute of Technology and Assistant Professor Yoshiaki Hori of the Kyushu Institute of Design for their encouragement, invaluable discussions and comments.

I am very grateful to Assistant Professor Katsuyoshi Iida of the Nara Institute of Science and Technology for his advice and comments. I extend thanks to Mr. Nobuo Ryoki, Mr. Yutaka Fukuda, and all other members of the Information Network Research Group at the Kyushu Institute of Technology for their kindly supports and suggestions.

Finally, I would like to thank my wife, Tomomi, for her understanding and help. My greatest appreciation goes to my parents and friends for their constant support and encouragement.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Internet is a common infrastructure that encompasses worldwide networks consisting of many computers. Each node on the Internet has a unique address and is connected to other nodes through various types of data links such as Ethernet, ATM (Asynchronous Transfer Mode) and so on. The major protocols in the Internet protocol suite are IP (Internet Protocol) for routing and delivery and TCP (Transmission Control Protocol) for reliable end-to-end transmission. Network protocols are normally developed in layers, with each layer responsible for a different facet of communication. The standard model is the ISO/OSI (International Standard Organization/ Open System Interconnect) model, which defines seven network layers. Although the OSI model is widely used and often cited as the standard, the TCP/IP protocol suite has been used by most Unix workstation vendors. The TCP/IP model is designed around a simple five-layer scheme. The two network models are illustrated in Fig. 1.1.

The Internet has been developed to support character-based transmission that is mainly used by applications such as e-mail and WWW (World Wide Web) browsing. What users were demanding during development was higher speed data transmission. Therefore, the transport protocol has been improved to use the bandwidth of a link more efficiently.

In recent years, however, the bandwidth is expanded, making multimedia applica-

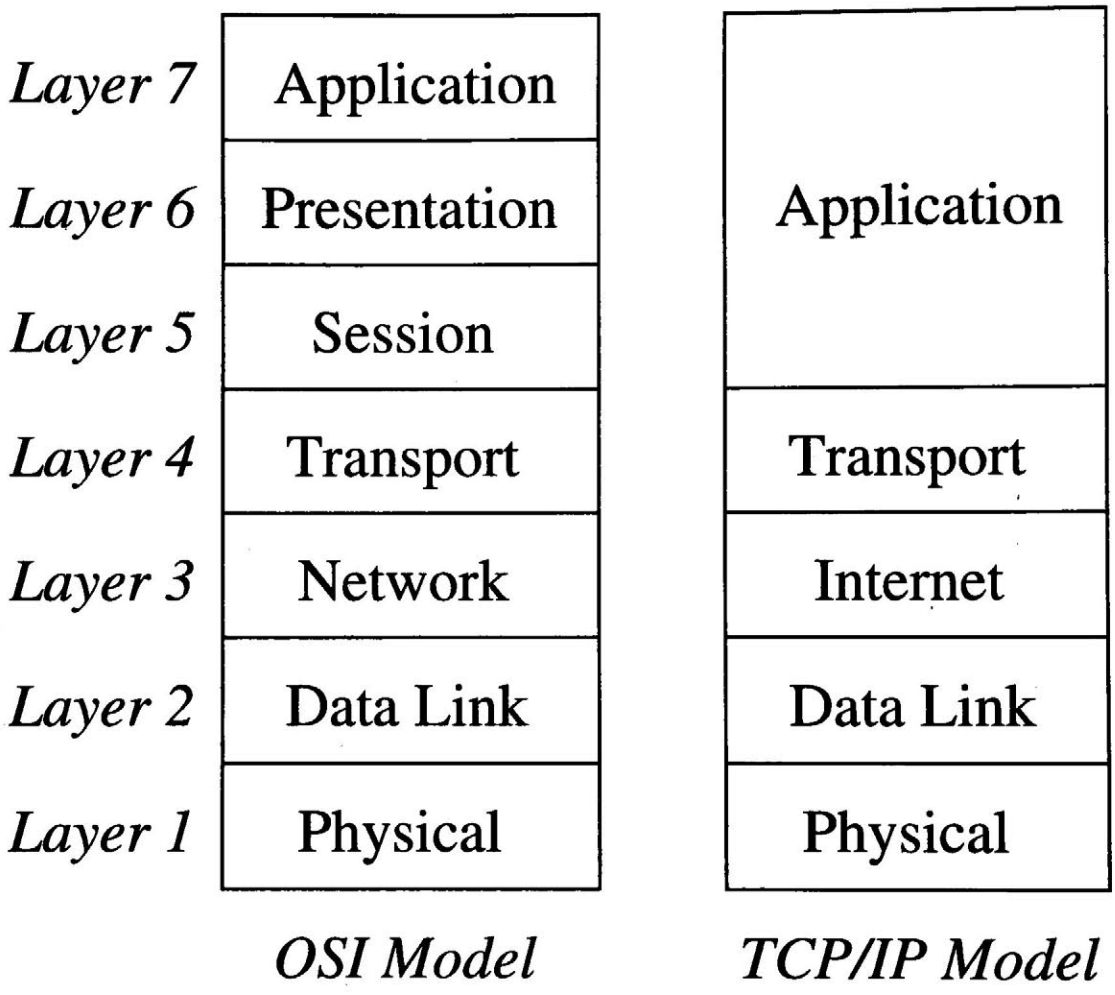


Figure 1.1: Layer Model

tions like voice and video streaming more attractive. The result is that Internet users desire a multimedia Internet that supports both real-time communications and non-real-time communications. Therefore, considering transmission delay time for real-time communications has become important, and priority scheduling schemes are being proposed to provide QoS (Quality of Service) assurance for real-time communications.

On the other hand, the growth of mobile networks has created an expectation for Internet access via mobile access networks. The transport protocol on the Internet has been tuned for wired networks where bit error rates are very low and packet losses occur mostly because of congestion in the networks. However, when a sender transmits packets over a wireless link, which characteristically has high bit error rates, the performance can be drastically degraded. A scheme for improving the performance over wireless networks is needed.

In the next sections, I discuss the emerging network environments in detail.

1.1 Multimedia Internet

As mentioned in the previous section, the UDP (User Datagram Protocol) traffic used by real-time communications will be intermingled with the TCP traffic used by non-real-time communications in the future multimedia Internet. The quality of the real-time traffic severely depends on its loss rate and delay time, whereas the non-real-time traffic is relatively tolerant of them. The performance of UDP traffic sharing a link with TCP traffic is very likely to degrade because TCP controls flow so as to use as much available bandwidth as possible. Therefore, CBR traffic should have priority over TCP traffic because of its stringent QoS requirement in such networks, and TCP will have to use the bandwidth left unused by CBR traffic.

In such QoS networks, a link is used by traffic of several classes, each of which requires a different level quality and is provided with a dedicated buffer on each node.

With ordinary static priority scheduling, lower priority traffic such as TCP may never be served at all, because the higher priority traffic may monopolize the bandwidth, a situation called *starvation*. Class-Based Queueing (CBQ) [FJ95], a variation of the ordinary priority scheduling, was proposed to prevent starvation of lower class traffic.

1.2 Mobile Internet

The growth of mobile networks and the Internet have also made services that transmit data, such as e-mail and mobile computing through the mobile networks more attractive. Therefore, the next-generation mobile system, called IMT-2000 (International Mobile Telecommunications-2000) [IMT], has been standardized by ITU (International Telecommunication Union) so as to provide data transmission that is faster than the current mobile system as well as world-wide roaming capability.

One technology employed by the IMT-2000 system is the W-CDMA (Wideband-Code Division Multiple Access) technology prescribed by 3GPP (3rd Generation Partnership Project) [3GPa] for wireless networks. The current mobile system provides a transmission rate of 9.6 Kb/s, whereas W-CDMA system can provide a maximum of 384 Kb/s outdoors or 2 Mb/s indoors. It also uses strong error recovery techniques to minimize the decreased use of wireless channels due to bit error. In fact, 3GPP employs FEC (Forward Error Correction) coding and ARQ (Automatic Repeat reQuest) as error recovery schemes in Layer 2 of wireless networks.

1.3 Overview of this dissertation

So far, I have described a change in what Internet users demand. As the Internet spreads more widely and the bandwidth increases, users will expect multimedia applications such as the streaming of voice and video, and mobile computing over mobile networks. Mul-

timedia applications require stringent QoS regarding performance. On the other hand, Internet access through mobile networks must exclude the influence of transmission error over wireless links. Whether the existing transport protocol work well even in such an emerging network environment is a new topic for study. Here, I examine the performance of the transport protocol, focusing on the emerging network environment.

In Chapter 3, I focus on the performance of TCP in the multimedia Internet. In such a network, the amount of link bandwidth available for TCP connections will vary with the amount of CBR traffic. If some CBR connections join a link, the amount of bandwidth available for TCP connections will abruptly and drastically decrease, causing multiple packet loss on TCP connection and further resulting in timeout. That can lead to throughput degradation of TCP connection. Therefore, I investigate whether existing TCP variants such as Reno TCP, NewReno TCP, and SACK TCP can adapt their window flow control mechanism in response to changing available bandwidth so as to use it efficiently.

In Chapter 4, I focus on the performance of TCP in the mobile Internet. The performance of TCP can be severely affected by the high bit error rate characteristic of wireless links, since TCP is tuned to perform well in wired networks where the error rates are very low and packet loss is mostly due to congestion. Therefore, I clarify the performance of TCP over wireless links in which Selective-Repeat ARQ is performed for recovering lost PDUs (Protocol Data Units) of Layer 2 on the links. ARQ can cause out-of-order delivery of SDUs (Service Data Units), which degrades TCP performance. Hence, I also examine the impact of the timeout timer employed to maintain sequence integrity.

In Chapter 5, I develop an effective receiver-based TCP flow control scheme without any modification of TCP senders in the mobile Internet, which are probably connected to wired networks, as described in Chapter 4. Multiple retransmissions by Layer 2 ARQ can adversely increase the transmission delay time of TCP packets, which will cause TCP to increase RTO (Retransmission TimeOut) unnecessarily. Furthermore, link bandwidth assigned to TCP flows can change in response to changing atmospheric conditions so as

to use wireless links efficiently. TCP therefore has to adapt its transmission rate according to the changing available bandwidth. Therefore, I propose a TCP flow control scheme that employs Layer 2 information in a wireless link at the mobile station. I also discuss the effectiveness of that scheme.

The results discussed in Chapter 3 are mainly taken from [KHO01], Chapter 4 from [KIHO], and Chapter 5 from [KKO02].

Chapter 2

Transport protocol

In the current Internet, there are two protocols used mainly in transport layer. One is TCP (Transport Control Protocol) [Ste94], a connection-oriented protocol that provides a reliable byte stream service; the other is UDP (User Datagram Protocol) [Pos80], a connectionless protocol that provides a simple datagram service. In the following sections, I describe both of these transport layer protocols.

2.1 TCP

TCP is a reliable end-to-end transport protocol used for many applications like telnet, file transfer, e-mail, and WWW (World Wide Web). It executes flow control using a sliding window mechanism and tries to make effective use of available bandwidth. Its flow control mechanism is summarized below.

- The TCP sender divides user data received from applications into packets and transmits them.
- When the TCP sender sends a packet, it maintains a timeout timer which is used to wait for the TCP receiver to acknowledge receipt of the packet. If an ACK

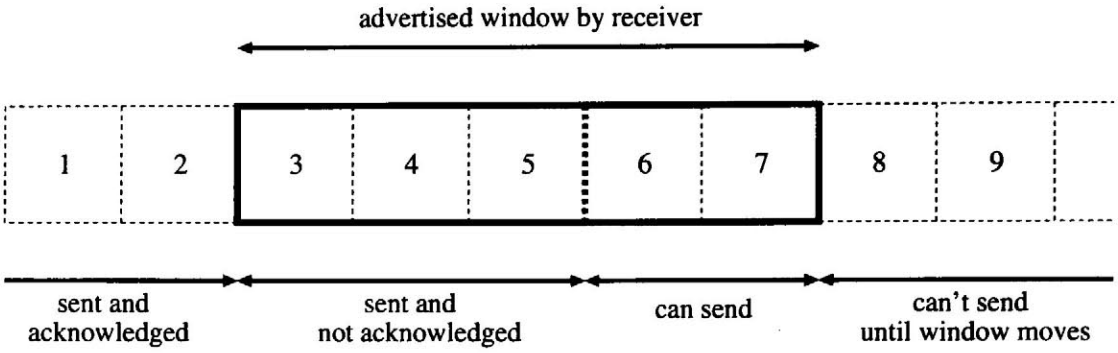


Figure 2.1: Sliding window

(acknowledgment) is not received in time, the packet is retransmitted.

- The TCP receiver sends an ACK when it receives the packet, resequences the packets if necessary, and discards duplicate packets.

TCP receiver advertises the amount of data (*awnd*: advertised window) that it can be received at a time according to its available buffer space. Consider the case that a TCP sender transmits packets up to the limitation advertised by the receiver. When the sender and receiver are connected to the same LAN (Local Area Network), it causes no problems. If there are routers and/or bottleneck links between the sender and receiver, however, the routers along the path must store the transmitted packets in their buffers. That can lead to buffer overflow, which degrades throughput. To prevent that, *slow start* and *congestion avoidance* algorithms have been proposed [JK88]. Those algorithms have been implemented in the TCP variants since 4.3BSD-Tahoe.

2.1.1 Sliding Window

The sliding window algorithm can be visualized as shown in Fig. 2.1. The numbers from 0 to 9 in the figure are the packet sequence numbers. The receiver advertised window size

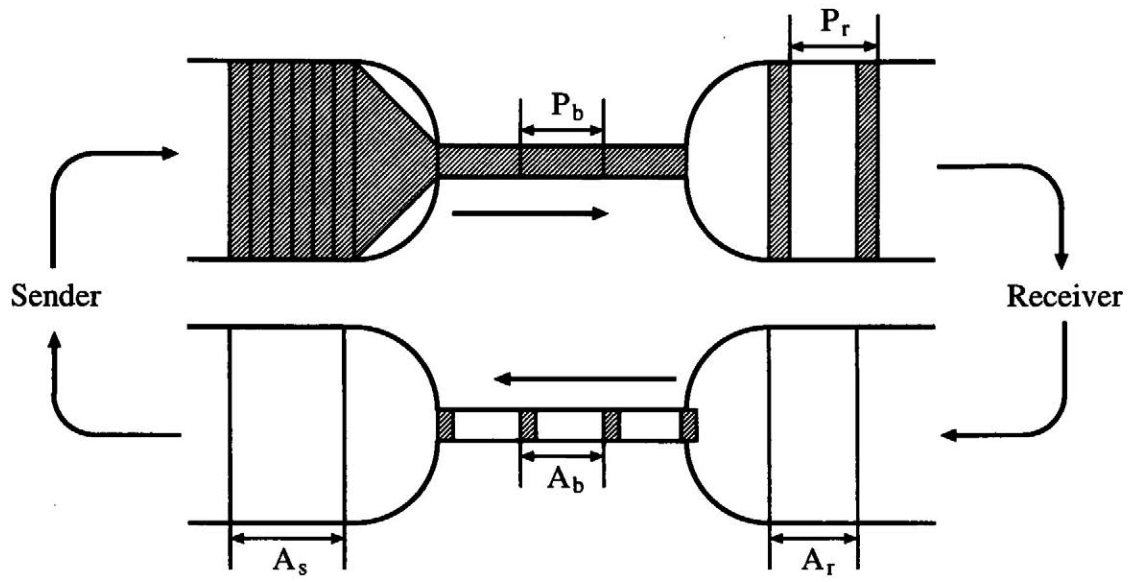


Figure 2.2: Self-clocking

indicates the amount of data that the receiver can receive at a time, which is determined by the available buffer size. In this case, it is five packets. The TCP sender has already received the ACKs corresponding to packets 1 and 2, but it has not yet received the ACKs for packets 3, 4 and 5. Although the packets 6 and 7 can be sent immediately at this time, the packets of sequence number 8 and higher cannot be sent since transmission is limited by the advertised window. When the advertised window moves to the right after the ACK for packet 3 is received, packet 8 can be sent. TCP executes flow control with this sliding window algorithm to prevent the overflow of the receiver's buffer.

2.1.2 Slow Start

A TCP sender executes flow control to use as much available bandwidth as possible and to manage the transmission rate so as to prevent overflow of the receiver's buffer. A typical scenario, in which a big pipe feeds a smaller pipe is illustrated in Fig. 2.2. In that scenario,

a sender transmits packets using all the available bandwidth of the bottleneck link. When the processing delay time at the receiver is the same for each packet, the spacing of the packets over the bottleneck link (P_b) and over the receiver's network (P_r), and the spacing of the ACKs over each network (A_r, A_b, and A_s) become the same. If the sender transmits packets according to the receipt of the ACKs, the transmission rate becomes as the same as the spacing of the packets over the bottleneck link. This is called self-clocking. However, excessive transmission of packets for self-clocking may cause congestion in the networks, whereas insufficient transmission of packets cannot accomplish self-clocking. Therefore, TCP uses a slow start algorithm, which increases the transmission rate gradually.

The slow start algorithm adds another window to the sender's TCP, called a *cwnd* (congestion window). When a new connection is established, the *cwnd* is initialized to one packet. Each time an ACK is received, the *cwnd* is increased by one packet. The sender can transmit packets up to the minimum of the *cwnd* and *awnd*. The *cwnd* is flow control imposed by the sender, while the *awnd* is flow control imposed by the receiver.

Figure 2.3 shows the relation between the transmission of the packets and the receipt of the ACKs. The sender sets the *cwnd* to one packet and transmits one packet (0RTT). When the ACK corresponding to the sent packet is received, the *cwnd* is incremented from one packet to two packets, and two packets can be sent (1RTT). When both of those two packets are acknowledged, the *cwnd* is increased to four packets (2RTT). This provides an exponential increase.

2.1.3 Congestion Avoidance

When traffic arrives at a router whose output capacity is less than the amount of traffic, congestion can occur. The congestion can lead to overflow of the buffer at the router. For example, when traffic arrives on a big pipe (fast LAN) and gets sent out a smaller pipe (slower WAN: Wide Area Network), or multiple traffic inputs arrive at a router whose

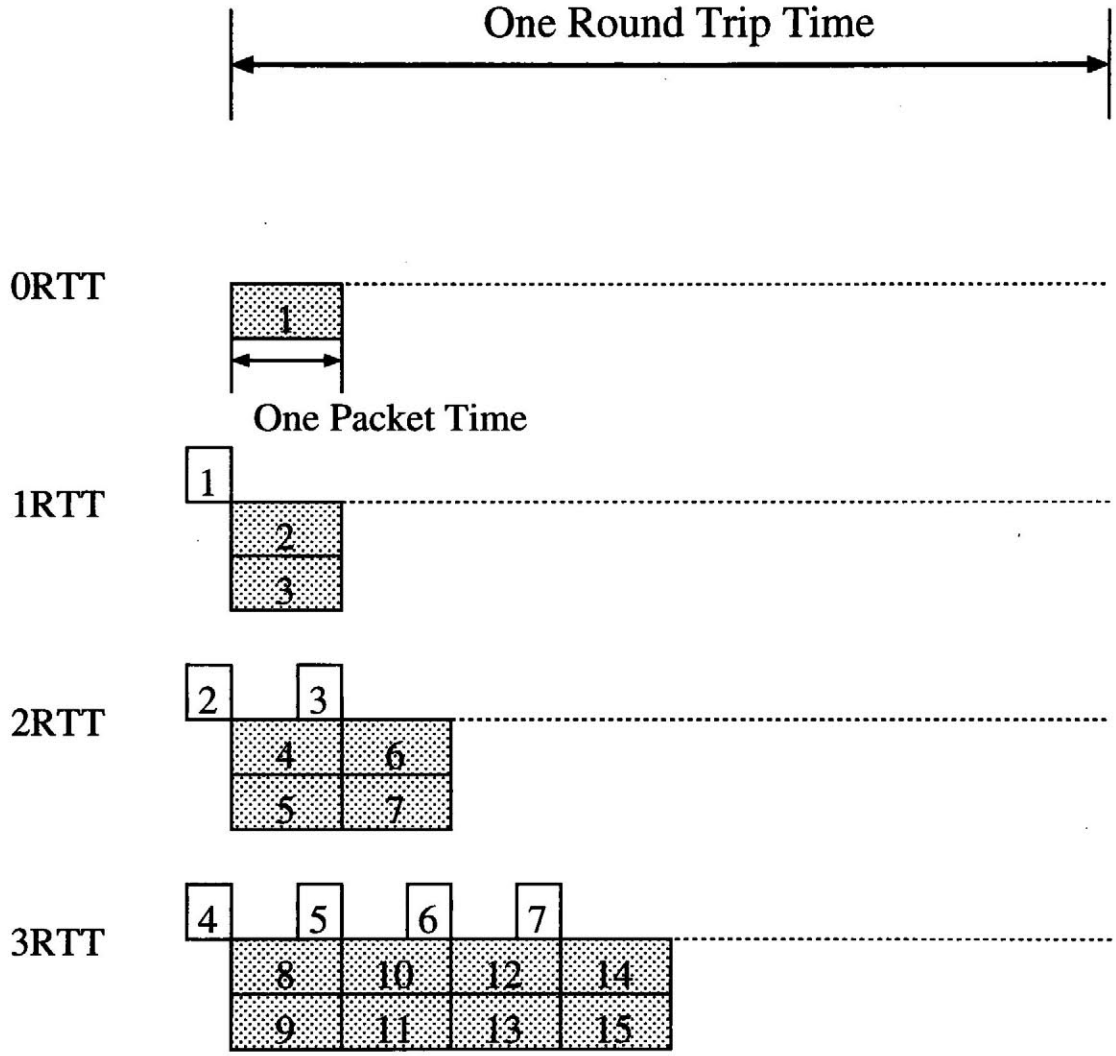


Figure 2.3: Slow start

capacity is less than the sum of the inputs, congestion can occur. The flow control which detects and avoids such congestion is called *congestion avoidance*.

The congestion avoidance algorithm assumes that packet loss caused by transmission error is very small, therefore packet loss signals congestion somewhere in the network between the source and destination. There are two indications of packet loss: a timeout occurring and the receipt of duplicate ACKs.

Congestion avoidance and slow start are independent algorithms with different objectives. However, when congestion occurs, the sender has to slow down the transmission rate, and then invoke slow start. They are thus implemented together in practice.

Congestion avoidance and slow start require the sender to maintain two variables: a congestion window (*cwnd*) and a slow start threshold size (*ssthresh*). The combined algorithm operates in the following way.

1. When a connection is initialized, *cwnd* is set to one packet and *ssthresh* is set to 65535 bytes.

2. The TCP sender transmits packets up to the minimum of *cwnd* and *awnd*.

3. When congestion occurs, half of the current window size (the minimum of *cwnd* and *awnd*, but at least two packets) is saved in *ssthresh*. If the congestion is indicated by a timeout, *cwnd* is set to one packet and slow start is executed.

4. When a new ACK is received, *cwnd* is increased. The way it increases depends on whether slow start or congestion avoidance is performed. If *cwnd* is less than or equal to *ssthresh*, slow start is performed; otherwise congestion avoidance is performed. Slow start continues until *cwnd* becomes larger than *ssthresh*, and then congestion avoidance takes over.

   (a) Slow start has *cwnd* begin at one packet and increment by one packet every time an ACK is received. Again, this provides an exponential increase.

(b) In the case of congestion avoidance, *cwnd* is incremented by $1/cwnd$ each time an ACK is received; that is, it is increased by at most one packet each round trip time. This is an additive increase, as opposed to the exponential increase of slow start.

TCP tries to increase the transmission rate rapidly with slow start while it is considered that congestion does not occur, and then increase the rate gradually with congestion avoidance in order to find the available bandwidth of the network.

An example of the operation of slow start and congestion avoidance is shown in Fig. 2.4. Consider that a timeout finally occurs in Fig. 2.3, and then *ssthresh* is set to four packets (0RTT in Fig. 2.4). First, slow start is performed until *cwnd* becomes smaller than *ssthresh*, and *cwnd* increases exponentially (0RTT – 2RTT). When *cwnd* exceeds *ssthresh*, congestion avoidance is performed, and *cwnd* increases additively (3RTT – 4RTT). If packet loss is detected, *cwnd* is initialized to one packet and *ssthresh* is set to a half of the current window size (5RTT).

2.1.4 Timeout

TCP provides a reliable transport layer. One of the ways it provides reliability is to have each end acknowledge the data packet it receives from the other end. However, data packets and acknowledgments can get lost. TCP handles this by setting a timeout when it sends a data packet, and if the data packet is not acknowledged when the timeout timer expires, it retransmits the data packet.

To decide the RTO (Retransmission TimeOut) value, the RTT experienced on a given connection must be measured. However, it can change over time, as routes might change and network traffic changes, and TCP should track these changes and modify the RTO accordingly. What is needed to implement it is to keep track of the variance in the RTT measurements in addition to the smoothed RTT estimator. Calculating the RTO based on

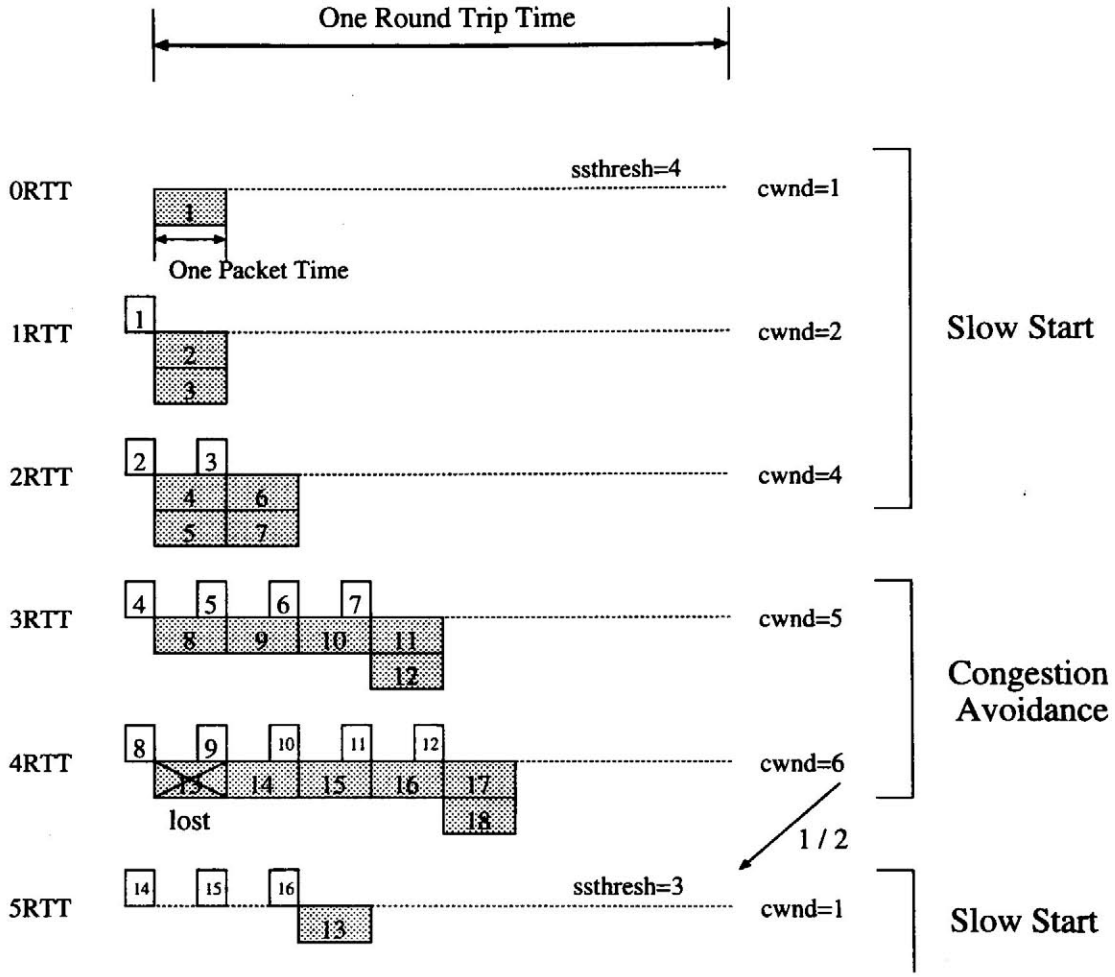


Figure 2.4: Slow start and congestion avoidance

the both the mean and variance provides much better response to wide fluctuations in the RTTs than just calculating the RTO as a constant multiple of the mean. The method of calculating RTO can be expressed as

$$\begin{aligned} Err &= M - A \\ A &\leftarrow A + gErr \\ D &\leftarrow D + h(|Err| - D) \\ RTO &= A + 4D \end{aligned}$$

where M is each RTT measurement, A is the smoothed RTT (an estimator of the average), and D is the smoothed mean deviation. Err is the difference between the measured value just obtained and the current RTT estimator. Both A and D are used to calculate the next RTO. The gain g is for the average and is set to 0.125. The gain h is for the deviation and is set to 0.25. The larger gain for the deviation makes the RTO increase faster when the RTT changes.

If a timeout and retransmission occur, the RTT estimator cannot be updated when the ACK for the retransmitted data finally arrives. This is because TCP does not know to which transmission the ACK corresponds. (Perhaps the first transmission was delayed and not thrown away, or perhaps the ACK of the first transmission was delayed.) Also, when a timeout occurs, the RTO is doubled for each retransmission, with an upper limit of 64 seconds. This doubling is called an exponential back-off and the algorithm is known as Karn's Algorithm [KP87].

2.1.5 Duplicate ACK – Fast Retransmit

The TCP receiver sends consecutive acknowledgments for the receipt of data packets back to TCP sender. For example, when the TCP receiver receives the packets up to the sequence number of 20, it sends the consecutive ACK requiring the packet 21 to TCP

sender. After this, if it receives packet 22 instead of 21, it sends the ACK requiring the unreceived packet 21 again. This acknowledgment is called a duplicate ACK because it requires the same packet. The purpose of this duplicate ACK is to let TCP sender know that a packet was received out of order, and to tell it what sequence number is expected. Therefore, the TCP receiver must generate an immediate acknowledgment (duplicate ACK) when an out-of-order packet is received.

Since the TCP sender does not know whether a duplicate ACK is caused by a lost packet or just an out-of-order packet, it waits for a small number of duplicate ACKs to be received. It is assumed that if there is just an out-of-order packet, there will be only one or two duplicate ACKs before the reordered packet is processed, which will then generate a new ACK. If three or more duplicate ACKs are received in a row, it is a strong indication that a packet has been lost. Then, TCP sender performs a retransmission of the expected packet without waiting for the retransmission timer to expire. This is the fast retransmit algorithm.

2.1.6 TCP implementation

In fact, the standard of TCP/IP implementation is the 4.x BSD (Berkeley Software Distribution) system and the BSD Networking Release distributed by the Computer System Research Group at the University of California at Berkeley. The first BSD system used generally was 4.2 BSD in 1983, and then 4.3 BSD-Tahoe was released with many improvements in 1988. The 4.3 BSD-Tahoe included slow start and congestion avoidance algorithms for the first time, and its source code is open to the public as the BSD Networking Release. Consequently, it is the basis of the TCP used by many computers in the current Internet. In this section, I describe TCP variants such as Reno TCP, NewReno TCP, and SACK TCP which have added many improvements based on Tahoe TCP and are mainly used in the current Internet.

Reno TCP – Fast Recovery

4.3 BSD-Reno was released in 1990. The main improvement to Tahoe TCP is the addition of the Fast Recovery algorithm [Jac90]. In this section, I explain the Fast Recovery algorithm.

There are two ways to detection lost packets as mentioned above; one is the timeout occurrence, the other is the receipt of duplicate ACKs. In the case of the timeout occurrence, heavy congestion is expected because the ACK is not received at all, so the TCP sender needs to reduce the transmission rate immediately. On the other hand, when duplicate ACKs are received, lighter congestion than for when a timeout occurs is expected, so the TCP sender does not need to reduce the transmission rate extremely.

In 4.3 BSD-Reno, when TCP sender receives three duplicate ACKs, it executes the Fast Recovery algorithm after a Fast retransmit as described below, rather than executing the slow start algorithm.

1. When the third duplicate ACK is received, TCP sender sets *ssthresh* to a half of the minimum of the current *cwnd* and *awnd*, and then transmits the lost packet (Fast Retransmit). *cwnd* is set to (*ssthresh* + 3) packets (three packets corresponds to the three duplicate ACKs, which means that three packets are removed from the network).

2. Each time another duplicate ACK arrives, *cwnd* is incremented by one packet, and the TCP sender transmits a packet if it is allowed by the new value of *cwnd*.

3. When the next ACK that acknowledges new data arrives, the TCP sender sets *cwnd* to *ssthresh* (the value set in step 1). This should be the ACK of the retransmission from step 1, one round trip time after the retransmission. Additionally, this ACK should acknowledge all the intermediate packets sent between the lost packet and the receipt of the third duplicate ACK. After this step, congestion avoidance is

performed.

In short, this algorithm slows the transmission rate by half when congestion occurs, and tries to maintain the rate during the congestion.

An example of this algorithm is shown in Fig. 2.5. In the figure, the changing of *cwnd* and the way of sending packets every time an ACK is received are illustrated.

a) It is assumed that *cwnd* is eight packets and the packet of sequence number 1 is lost among the eight sent packets. At this time, the ACKs corresponding to packets 2 to 8 are duplicate ACKs requiring packet 1.

b) When the third duplicate ACK arrives, Fast Retransmit is performed (packet 1 is retransmitted). *ssthresh* is set to four packets (half of *cwnd*), and *cwnd* is set to seven (4 + 3) packets.

c) When the duplicate ACK for the packet 5 arrives, *cwnd* is incremented by one to eight packets. By this, the packets up to the sequence number of 8 can be sent, but their packets have already been sent, so no additional packets can be sent.

d) When the duplicate ACK for packet 6 arrives, *cwnd* increases to nine packets, so that packet 9 can be sent.

e) When the duplicate ACK for packet 7 arrives, *cwnd* increases to ten packets, so that packet 10 is sent.

f) When the duplicate ACK for packet 8 arrives, *cwnd* increases to eleven packets, so that packet 11 is sent.

g) When the ACK for the retransmitted packet 1 (requiring the packet 9) arrives, Fast Recovery is terminated. The left edge of *cwnd* advances to sequence number 9 and *cwnd* is set to the value of *ssthresh* (four packets). Then, congestion avoidance is performed and the packet 12 is sent.

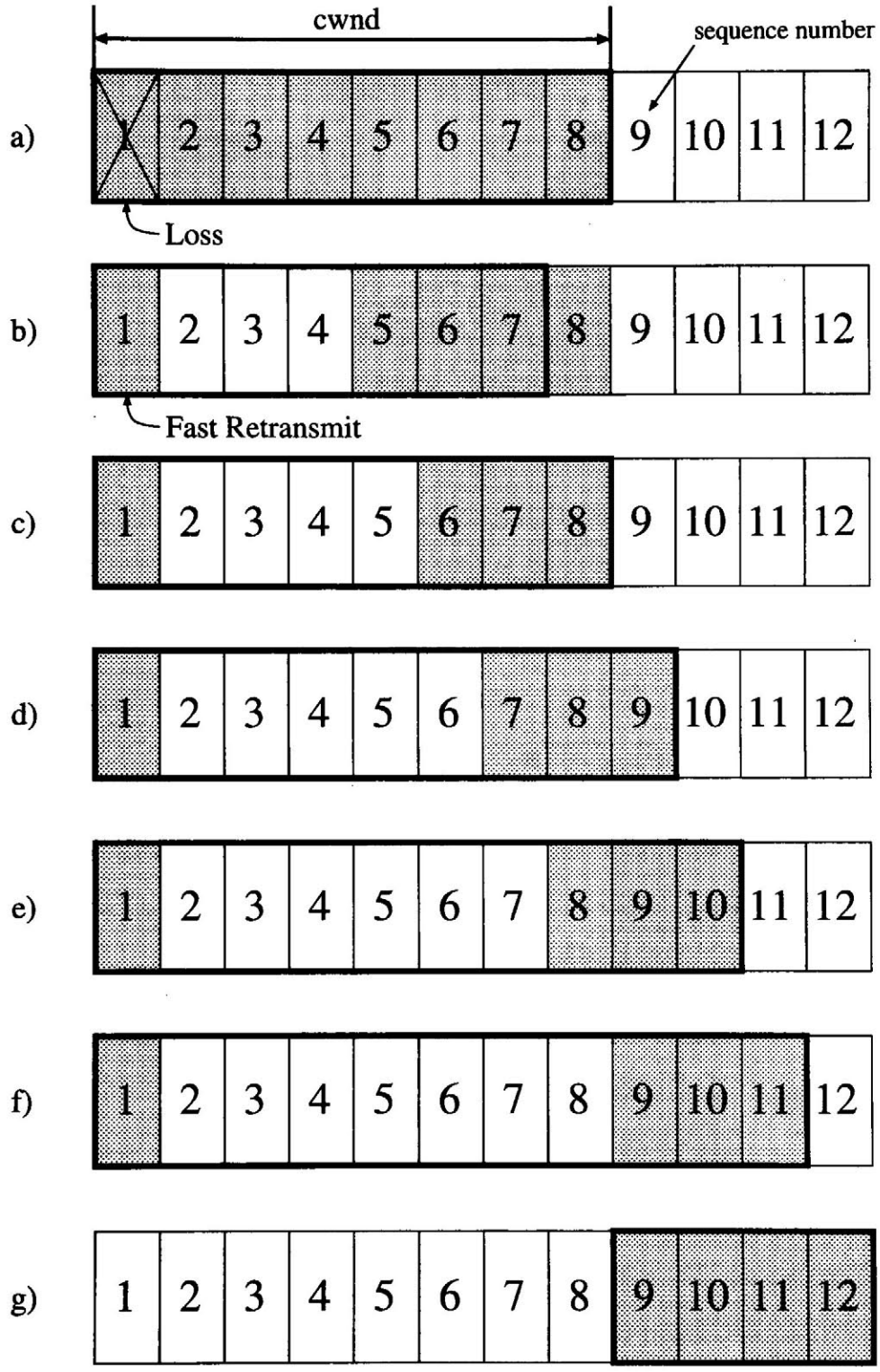


Figure 2.5: Fast recovery

NewReno TCP – Improvement of Fast Recovery

The Fast Recovery described in the previous section works well only if the number of lost packets in a window is one, as shown in Fig. 2.5. In practice, however, there may be multiple packet losses within a window, and Fast Recovery cannot work well. The example of that case is illustrated in Fig. 2.6.

a) It is assumed that *cwnd* is eight packets, the same as in the case of Fig. 2.5, and packets 1, 3 and 4 of the eight sent packets are lost. The ACKs corresponding to packet 2 and packets 5 to 8 are duplicate ACKs requiring packet 1.

b) When the third duplicate ACK arrives, Fast Retransmit is performed (packet 1 is retransmitted). *ssthresh* is set to four packets (half of *cwnd*), and *cwnd* is set to seven (4 + 3) packets.

c) When the duplicate ACK for packet 7 arrives, *cwnd* is incremented by one to eight packets. By this, the packets up to the sequence number of 8 can be sent. However, those packets have already been sent, so no additional packet can be sent. Although the Fast Recovery algorithm tries to keep sending four packets (half of *cwnd* when the packet loss occurs) in the network, the losses of packets 2 and 3 cannot be detected in this case, so there are only two packets in the network.

d) When the duplicate ACK for packet 8 arrives, *cwnd* increases to nine packets, so that packet 9 can be sent.

e) When the ACK for the retransmitted packet 1 arrives after 1RTT from step b), Fast Recovery is terminated. Then, *cwnd* is set to four packets, and congestion avoidance is performed. The left edge of *cwnd* advances to sequence number 3, because the received ACK requires packet 3. At this time, packets 3 to 6 have already sent, and no additional packet can be sent.

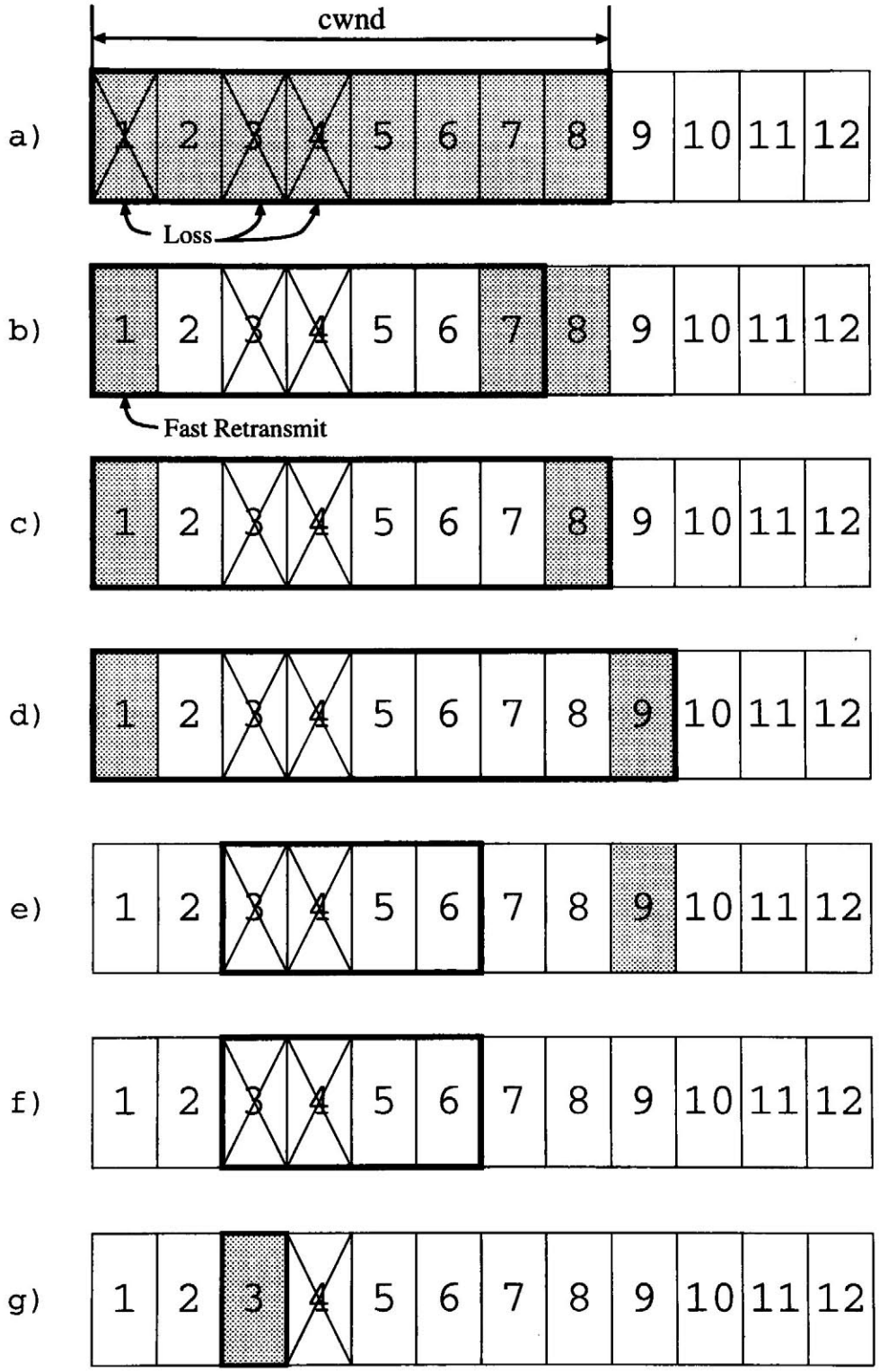


Figure 2.6: Fast recovery: the case of occurring multiple packet losses

f) Although the ACK for packet 9 arrives, *cwnd* does not increase and no packet can be sent because the received ACK is the first duplicate ACK requiring packet 3.

g) Since there is no packet being transmitted in the network, the TCP sender must wait for the timeout timer to expire. After the retransmission timeout, packet 3 is retransmitted, and then slow packet).

As mentioned above, when multiple packet losses occur within a window, Reno TCP cannot maintain the expected transmission rate during the Fast Recovery phase; furthermore there is every possibility of causing a timeout, which can lead to throughput degradation.

That drawback led to the proposal of NewReno TCP, whose Fast Recovery algorithm was modified [FH99]. This algorithm has a new variable *recovery* and works as described below.

1. When the third duplicate ACK is received and TCP sender is not already in the Fast Recovery phase, *ssthresh* is set according to the following equation.

$$ssthresh = \max(FlightSize/2, 2 * MSS)$$

where $FlightSize$ is the amount of data that is sent but not acknowledged, and MSS is the maximum segment size of TCP. In addition, *recover* is set to the highest sequence number transmitted (the right edge of *cwnd*). The lost packet is retransmitted and *cwnd* is set to *ssthresh* + 3 packets.

2. Each time another duplicate ACK arrives, *cwnd* is incremented by one packet, and a packet is sent if allowed by the new values of *cwnd* and *awnd*.

3. When an ACK that acknowledges new data arrives, it could be the acknowledgement elicited by the retransmission from step 1, or that elicited by a later retransmission.

(a) If this ACK acknowledges all of the data up to and including *recover*, then the ACK acknowledges all the intermediate packets sent between the original transmission of the lost packet and the receipt of the third duplicate ACK. The Fast Recovery is terminated and congestion avoidance is started in the same way as in Reno TCP.

(b) If this ACK does not acknowledge all of the data up to and including *recover*, then it is a partial ACK. In that case, the TCP sender retransmits the first unacknowledged packet. *cwnd* is decreased by the amount of new data acknowledged, and then is incremented by one packet. A new packet is sent if permitted by the new value of *cwnd*. This *partial window deflation* attempts to ensure that, when Fast Recovery eventually ends, an amount of data that is approximately equal to *ssthresh* will be outstanding in the network. Fast Recovery is continued from step 2. Also, for the first partial ACK that arrives during the Fast Recovery phase, the retransmit timer is reset.

An example of this algorithm for the same case as shown in Fig. 2.6 is presented in Fig. 2.7.

a) It is assumed that *cwnd* is eight packets and packets 1, 3, and 4 of the eight sent packets are lost. The ACKs corresponding to packet 2 and packets 5 to 8 are duplicate ACKs requiring packet 1.

b) When the third duplicate ACK arrives, Fast Retransmit is performed (packet 1 is retransmitted). *ssthresh* is set to four packets (half of *cwnd*), and *cwnd* is set to seven (4 + 3) packets. In addition, the sequence number of 8 is recorded in *recover*.

c) When the duplicate ACK for packet 7 arrives, *cwnd* is incremented by one to eight packets. By this, the packets up to the sequence number of 8 can be sent. However, those packets have already sent, so no additional packet can be sent.

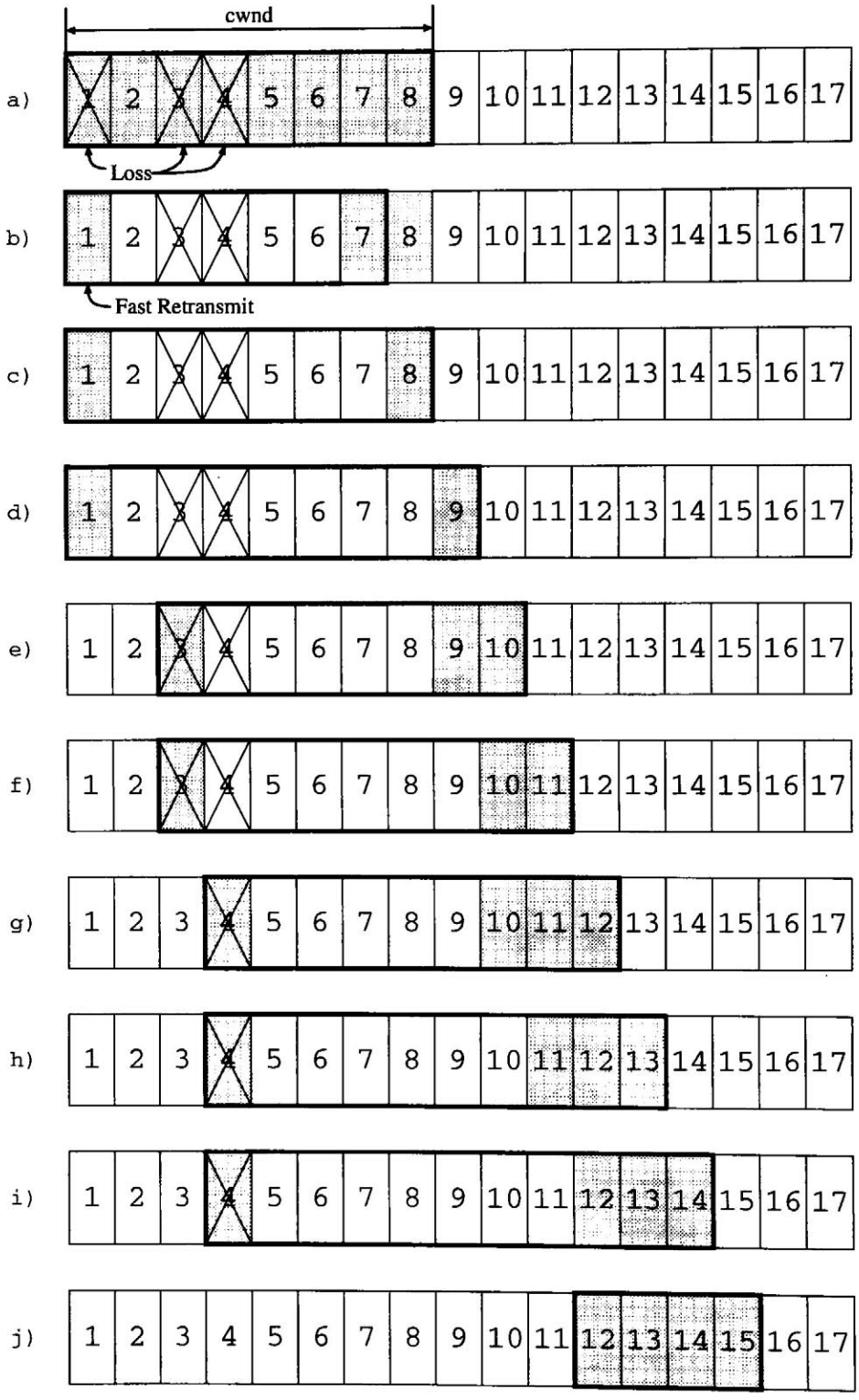


Figure 2.7: Fast recovery: NewReno TCP

d) When the duplicate ACK for packet 8 arrives, *cwnd* increases to nine packets, so packet 9 can be sent.

e) When the ACK for the retransmitted packet 1 arrives after 1RTT from step b), Fast Recovery is continued, because the ACK requires packet 3 and does not include *recover*, and the required packet is retransmitted. The left edge of *cwnd* moves to the sequence number of 3. Also, since *cwnd* is decreased by the amount of the new acknowledged data and then incremented by one packet, it becomes eight packets and packet 10 is sent.

f) When the ACK for packet 9 arrives, *cwnd* increases to nine packets, and packet 11 is sent.

g) When the ACK for the retransmitted packet 3 arrives, it also does not include *recover*, so the same process as step e) is performed. Packet 4 is retransmitted, and the left edge of *cwnd* moves to the sequence number of 4. *cwnd* becomes nine packets (9 - 1 + 1), and then packet 12 is transmitted.

h) When the ACK for packet 10 arrives, *cwnd* increases to ten packets and packet 13 is sent.

i) When the ACK for packet 11 arrives, *cwnd* increases to eleven packets and packet 14 is sent.

j) When the ACK for the retransmitted packet 4, which includes *recover*, arrives, Fast Recovery is terminated. The left edge of *cwnd* moves to sequence number 12 and *cwnd* is set to four packets (the value of *ssthresh*). Then congestion avoidance is started and packet 15 is transmitted.

As mentioned above, the Fast Recovery algorithm of NewReno TCP can maintain transmission without timeout, even when multiple packet losses occur in a window.

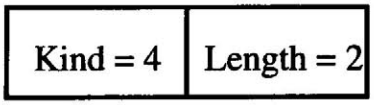


Figure 2.8: SACK TCP header: SACK-permitted option

SACK TCP – Selective-Repeat ARQ

The packet retransmission scheme mainly used by the current TCP variants is Go-back-N ARQ. In this scheme, when the TCP sender detects a packet loss, it retransmits the lost packet and the subsequent unacknowledged packets. Actually, the reason that the TCP sender does not retransmit packets for which correct reception has not been confirmed is effective use of network bandwidth. Since TCP uses a cumulative acknowledgement scheme, multiple packet losses in a window can have a catastrophic effect on TCP throughput when Reno TCP is used. As a means of averting this problem, TCP with SACK (Selective Acknowledgement) option [FF96, MMFR96] is proposed in addition to NewReno TCP, as mentioned previous section.

SACK TCP uses the ARQ scheme based on the effective Selective-Repeat ARQ rather than on Go-back-N ARQ. It is a strategy which corrects this behavior in the face of multiple dropped packets. With selective acknowledgements, the TCP receiver can inform the sender about all packets that have arrived successfully, so that the sender need retransmit only the packets which have actually been lost.

The selective acknowledgement extension uses two TCP options. The first is an enabling option, *SACK-permitted*, which may be sent in a SYN packet to indicate that the SACK option can be used once the connection is established. The other is the SACK option itself, which may be sent over an established connection once permission has been given by SACK-permitted.

The SACK-permitted option, which is a two-byte option as shown in Fig. 2.8, may

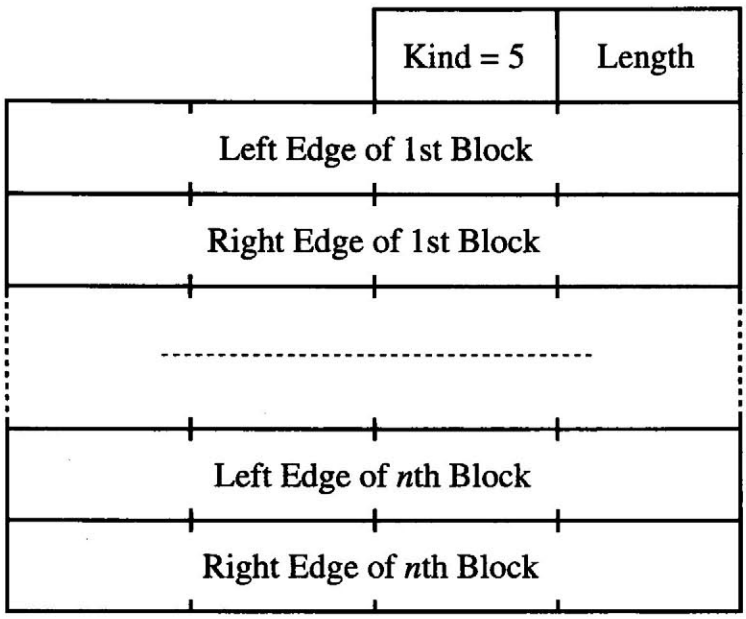


Figure 2.9: SACK TCP header: SACK option

be sent in a SYN by a TCP that has been extended to receive the SACK option once the connection has opened. It must not be sent in non-SYS packets.

The SACK option as shown in Fig. 2.9 is to be used to convey extended acknowledgement information from the receiver to the sender over an established TCP connection. It is to be sent by a receiver to inform the sender of non-contiguous blocks of data that have been received and queued. The receiver awaits the receipt of data to fill the gaps in sequence number between received blocks. When missing packets are received, the receiver acknowledges the data normally by advancing the left window edge in the Acknowledgement Number Field of the TCP header. The SACK option does not change the meaning of the Acknowledgement Number Field.

A SACK option that specifies n blocks will have a length of $(8n + 2)$ bytes, so that the 40 bytes available for TCP options can specify a maximum of 4 blocks. It is expected that SACK will often be used in conjunction with the Timestamp option [JBB92] used for

RTTM (Round-Trip Time Measurement), which takes an additional 10 bytes (plus two bytes of padding); thus a maximum of 3 SACK blocks will be allowed in this case.

2.1.7 Issues on wireless networks

As already briefly mentioned, TCP was designed to operate over highly reliable links and with stationary hosts. It assumes all packet loss to be caused by network congestion.

TCP performs well over such networks by adapting to end-to-end delays and to congestion of the network. It maintains a congestion window which adapts to the current network conditions. However, when a TCP sender transmits packets over wireless links, which are characterized by high bit error rates, the performance can decrease drastically. This is because TCP recognizes packet loss as a result of congestion and reduce its transmission rate, even though the loss may be due to bit error on a wireless link. For example, a W-CDMA system employs FEC (Forward Error Correction) coding and an ARQ (Automatic Repeat reQuest) mechanism as error recovery schemes in Layer 2 of wireless networks to reduce the damage due to bit error on TCP connections. The proposed means of improving TCP over wireless links are generally classified into the following categories.

- Split connection schemes
- Proxy schemes
- End-to-end schemes

Split connection schemes

In the split connection approach, the TCP connection from the fixed host is terminated at an intermediate host (e.g. the base station of the mobile network). A separate TCP connection is used from the intermediate host to the mobile host. As the second TCP

connection operates only over the wireless link, it can be highly optimized for that environment.

Indirect-TCP (I-TCP) [BB95] was proposed as one implementation of this scheme. It maintains two completely independent connections between fixed host and intermediate host, and between intermediate host and mobile host.

However, this scheme has the problem that the end-to-end semantic is violated.

Proxy schemes

For the proxy approach, the Snoop protocol [BSAK95] was proposed. It consists mainly of modifications of the network layer software at the base station. The base station caches the packets coming from the fixed host to allow it to retransmit packets lost due to transmission error on the wireless link. When the base station receives a duplicate ACK, it retransmits the lost packet and the ACK is not sent to the fixed host.

In this scheme, the end-to-end semantic is preserved because the ACKs are sent to the fixed host only when the packet has been successfully transmitted over wireless link. However, this scheme cannot use the IPsec, which encodes the payload of an IP datagram, because it requires snooping into the header of TCP packets at the base station.

End-to-end schemes

In this approach, it is important to distinguish whether packet loss is due to transmission error over the wireless link or to congestion of the networks. If a packet loss is due to wireless transmission error, the TCP sender just retransmits the packet but does not decrease its transmission rate. The TCP sender executes its flow control only if it is due to network congestion.

For example, in the case of ELN (Explicit Loss Notification) [BPSK96], the TCP sender is informed by the mobile host with a special TCP option that a previous packet

has been discarded due to wireless transmission error. Also, an intermediate host could add a TCP option to the ACK packets indicating that packets have been discarded due to network congestion. Therefore, the TCP sender can distinguish whether packet loss is due to transmission error or due to congestion, and react appropriately.

However, it is not easy to implement this scheme, and it requires modification of many hosts on the Internet.

2.2 UDP

UDP (User Datagram Protocol) provides a procedure for application programs to send messages to other programs with a minimum of protocol mechanism. The protocol is transaction oriented, and delivery and duplicate protection are not guaranteed. Applications requiring ordered reliable delivery of streams of data should use the TCP as mentioned above.

UDP is mainly used by real-time communication, which attaches more importance to real-time operation than to reliability. There is CBR (Constant Bit Rate) traffic, which sends UDP packets at a constant rate, as one way of realizing the real-time communication.

Chapter 3

Performance Comparison of TCP Implementations in QoS Provisioning Networks

3.1 Introduction

In the future Internet, the real time communication generating such as CBR (Constant Bit Rate) traffic, which will be transmitted by UDP datagram, will spread widely. However, the current Internet supports mainly the non-real time communication and cannot provide QoS (Quality of Service) assurance for real time communication so far. The performance of CBR traffic sharing a link with TCP (Transmission Control Protocol) connections is very likely to degrade due to TCP connections which execute flow control to utilize as much available bandwidth as possible [JK88, WS95, APS99, HSSO98]. Therefore, in QoS networks, CBR traffic should have priority for its stringent QoS requirement over non-real time traffic such as TCP connections, which will use the unused bandwidth left by CBR traffic.

In the QoS networks, a link is used by traffic of several classes, each of which requires different quality and is provided with a dedicated buffer on each node. As a packet scheduling method, Class-Based Queueing (CBQ) [FJ95] was proposed to avoid starvation for low class traffic such as TCP. Then we use the CBQ as the QoS scheduler in this study.

In this context, the amount of a link bandwidth available for TCP connections will change with changing amount of CBR traffic. If some CBR connections newly join a link, the amount of bandwidth available for TCP connections will abruptly and drastically decrease, thereby causing multiple packet loss on TCP connection and further resulting in timeout occurrence. Consequently, this can lead to throughput degradation of TCP connections. The performance of Tahoe TCP has been already examined [FJ95]. However, the performance of other TCP variants such as Reno TCP, NewReno TCP and TCP with SACK option has not been studied yet. Reno TCP is implemented mainly on current Internet computers and NewReno TCP [FH99] has been proposed to solve the Reno's problems that the multiple packet loss in one window causes timeout and throughput degradation. SACK TCP [FF96, MMFR96] employs more efficient retransmission mechanism, i.e., Selective-Repeat ARQ (Automatic Repeat reQuest) different from Go-Back-N ARQ used by Reno TCP and NewReno TCP.

Our major purpose is to investigate whether they work well even in QoS networks where the available bandwidth drastically changes from time to time. In other words, we are interested in whether some of them or all of them can adapt their window flow mechanism in response to changing available bandwidth to use it efficiently. To this end, in the present study, we will clarify how these TCP variants behave in QoS networks by means of simulations and compare their performance.

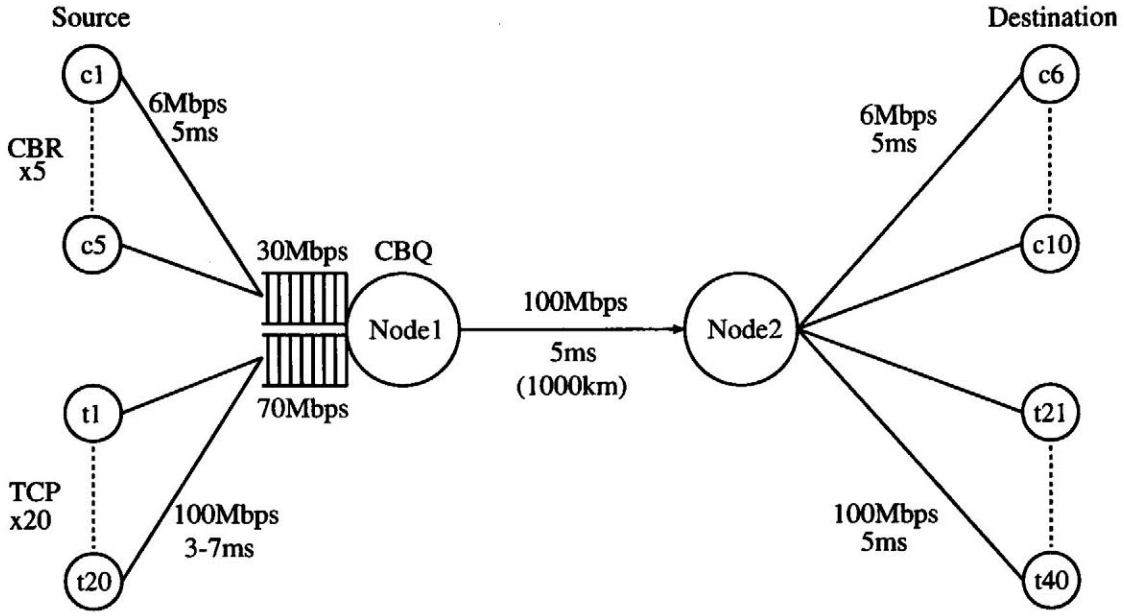


Figure 3.1: Simulation Model

3.2 Simulation Model

In our simulation, we employ the CBQ as a QoS scheduler. A link is shared by 5 CBR connections and 20 TCP connections as shown in Fig. 3.1; CBR connections have priority. The UCB/LBNL/VINT Network Simulator NS Version 2 [NS] is used for this research.

In Fig. 3.1, Node 1 is CBQ Gateway connected to Node 2 through the bottleneck link, which is of 100 Mbps in bandwidth and of 1000 km in length, resulting in 5 msec propagation delay. 30 Mbps of the link bandwidth is allocated to CBR traffic, and 70 Mbps to TCP traffic. The CBQ Gateway is equipped with a buffer of 200 packets in length for each traffic, and packet loss can occur in this CBQ Gateway. We here focus the performance of TCP congestion control scheme in a very common situation in the current Internet, i.e., in a bottleneck link shared by many TCP flows.

Hosts *c1–c5* transmit CBR traffic to *c6–c10*, and *t1–t20* transmit TCP traffic to *t21–t40*. *c1–c5* are connected to Node 1 with links of 6 Mbps and of 5 msec in propagation

delay, which are the same as the links connecting *c6–c10* with Node 2. *t1–t40* are connected with nodes through links of 100 Mbps. Among them, all links between Node 2 and each of *t21–t40* are of 5 msec in propagation delay. When the propagation delay of all TCP connections is the same, their window flow control is very likely to be synchronized [HSSO98]. This is very different from an actual environment. Thus, the propagation delay here varies from 3 to 7 msec in links connected with *t1–t20* in order to prevent such synchronization.

In this simulation, it is assumed that each TCP traffic is used for file transfer, while each CBR traffic is generated by MPEG video stream of 6 Mbps. The TCP variants employed here are 4.3BSD-Reno, NewReno, and SACK as mentioned above. CBR traffic is assumed to have a constant packet interval as a result of shaping with a hardware shaper unless otherwise noted.

The packet size is set at 500 bytes for both of CBR and TCP. We carry out simulation experiments for 20 seconds.

3.3 Simulation Results

In this section, we will show the characteristics of TCP performance in QoS networks by means of simulation results. In particular, some variants of TCP, which are currently or will be very common, will be examined, and the difference are discussed.

Before investigating TCP performance in QoS networks, we give results of TCP performance in networks accommodating only TCP connections just for later comparison.

3.3.1 TCP performance in networks without CBR traffic

We examine the total TCP throughput over a link used by only TCP traffic. The resulting performance can be considered to be very similar to that in the current Internet because

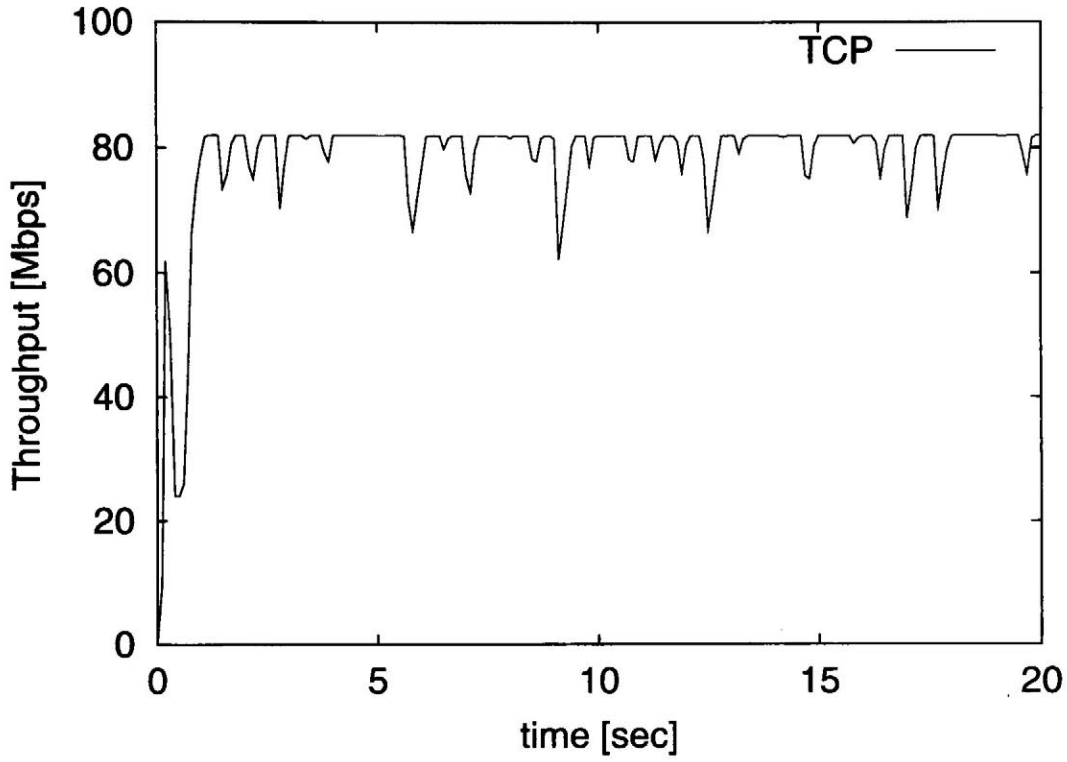


Figure 3.2: Total TCP throughput: Reno

TCP traffic is dominant there. Figures 3.2, 3.3, and 3.4 show the throughput characteristics of Reno TCP, NewReno TCP, and SACK TCP. The available bandwidth is set at 82 Mbps for comparison with the results in the following subsections. In Fig. 3.4, SACK TCP achieves the maximum throughput at almost all time. As shown in Fig. 3.2 and 3.3, Reno TCP and NewReno TCP also provide good performance although SACK TCP outperforms them.

3.3.2 TCP traffic and one CBR stream of large bandwidth

We now consider the case where CBR traffic, which requires a large bandwidth such as video streams, and TCP traffic are sharing the bottleneck link. This case will happen very

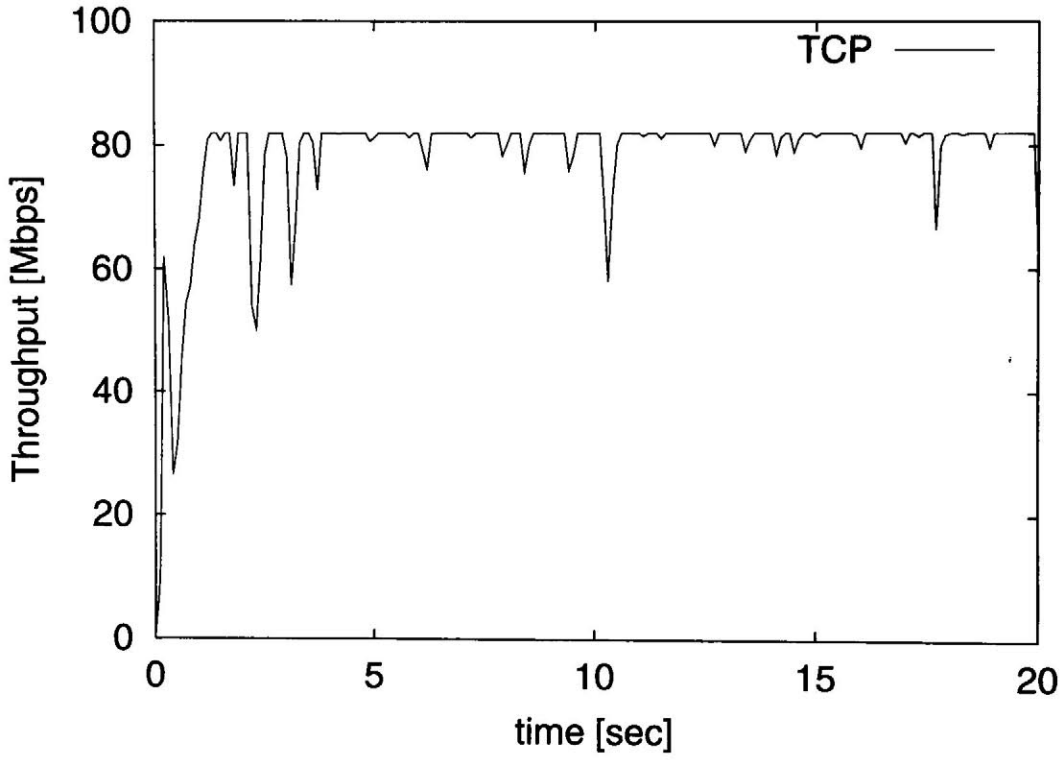


Figure 3.3: Total TCP throughput: NewReno

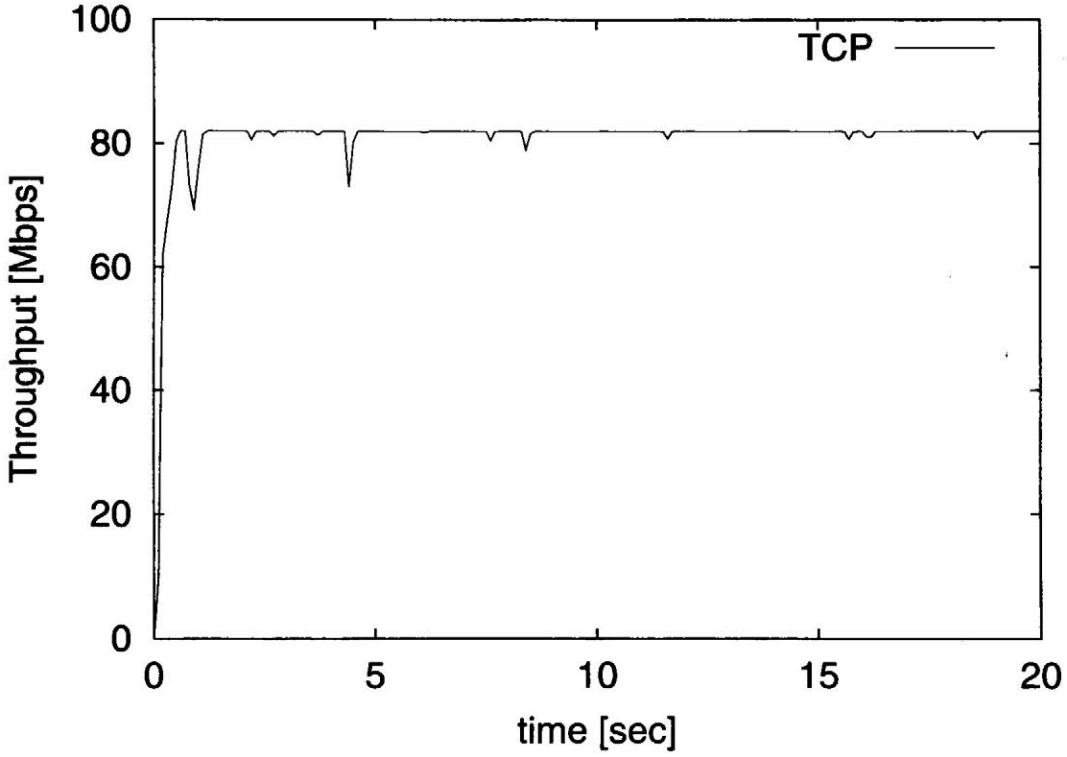


Figure 3.4: Total TCP throughput: SACK

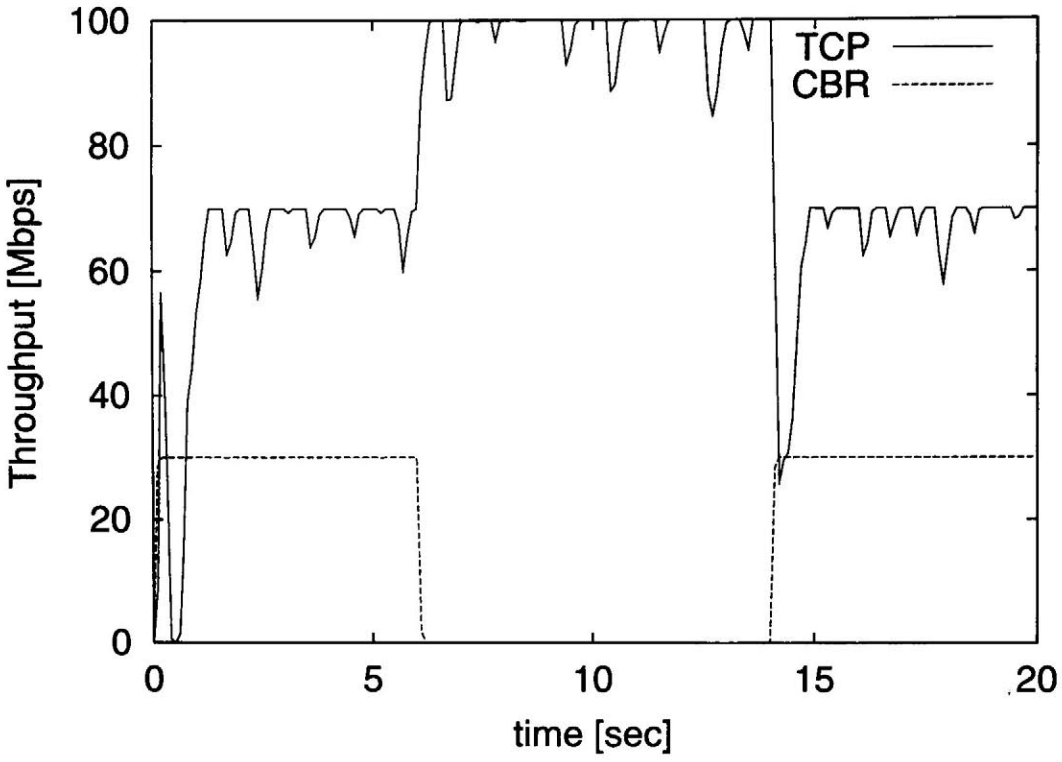


Figure 3.5: Total TCP throughput: Reno

often or be very common in the future Internet. We suppose that CBR traffic is transmitted at 30 Mbps from 0 to 6 second and from 14 to 20 second. Hence, the average rate of CBR traffic is 18 Mbps over 0 to 20 second, and the available bandwidth for TCP traffic is thus 82 Mbps on average, which is the same as that in the previous subsection. Figures 3.5, 3.6, and 3.7 show the total throughput characteristics of Reno TCP, NewReno TCP, and SACK TCP, respectively. The flow of CBR traffic abruptly joins at 14 second and causes the throughput degradation to Reno TCP and NewReno TCP connections, particularly, whereas SACK TCP is very robust against changing available bandwidth.

Let's investigate the reason for the above degradation. For this purpose, the average number of the timeout and Fast Recovery occurrence per TCP flow is shown in Table 3.1. The average throughput over 0 to 20 second is given in "Large CBR" of Table 3.3. From

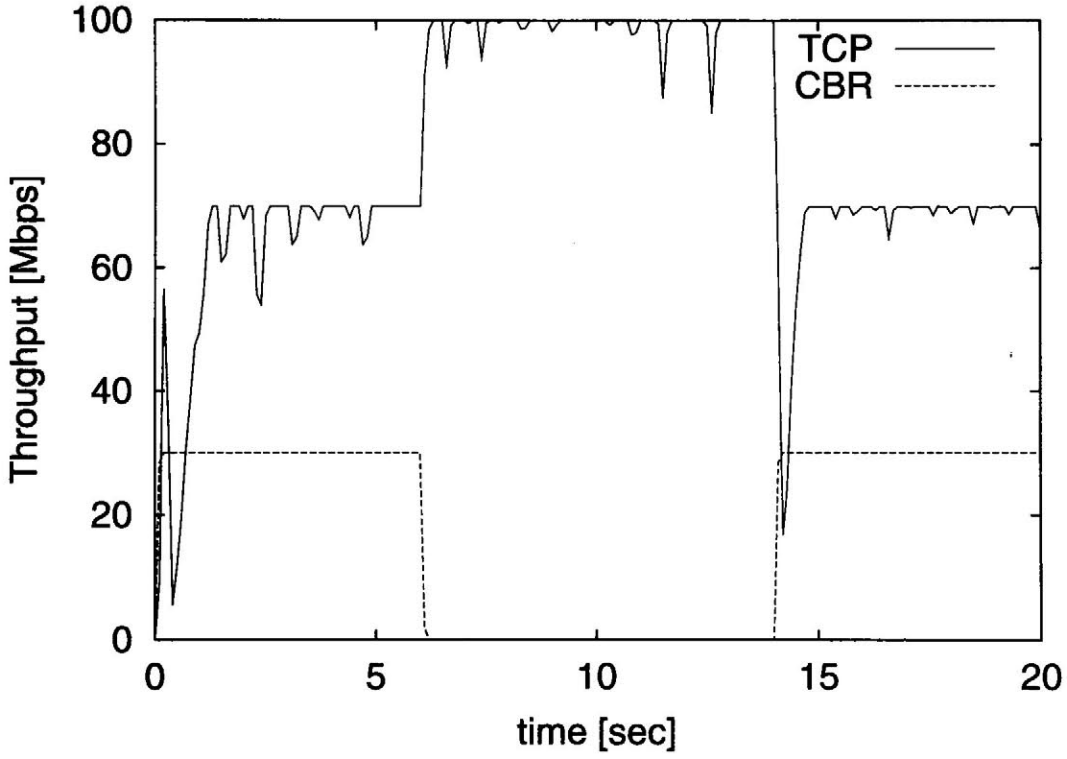


Figure 3.6: Total TCP throughput: NewReno

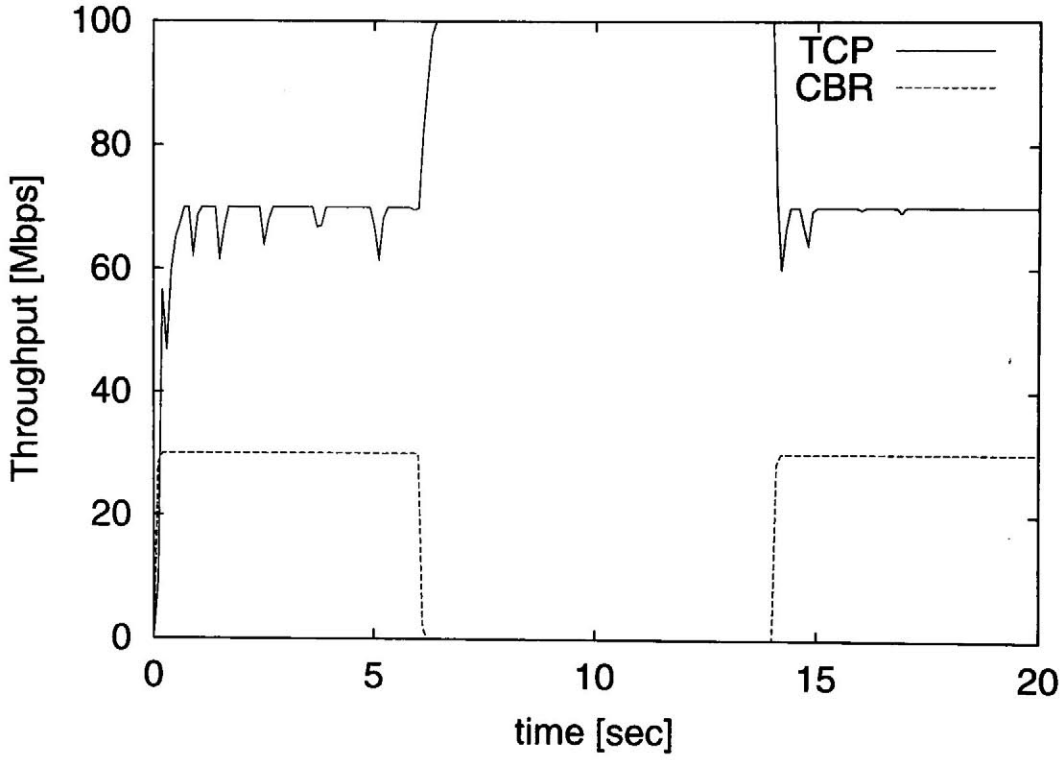


Figure 3.7: Total TCP throughput: SACK

Table 3.1: The average number of Timeout and Fast Recovery per flow

	Reno	NewReno	SACK
Timeout	6.1	0.1	0.7
Fast Recovery	26.0	22.1	23.3

the table, we can see that NewReno TCP and SACK TCP suffer much fewer timeout than Reno TCP. Furthermore, timeout rarely occurs in NewReno TCP. Timeout occurrence on some TCP connection prevents the related sender from transmitting packets for long duration, e.g. few to several seconds. This results in sever throughput degradation as shown in Table 3.3. NewReno TCP successfully avoids timeout occurrence by improving window flow control mechanism. However, since it still employs G-back-N ARQ like Reno TCP, its throughput is limited as shown in the table. Furthermore, SACK TCP achieves excellent performance, whereas it suffers more timeout occurrence than NewReno TCP. Therefore, we see that Selective-Repeat ARQ employed by SACK TCP is very effective in using changing available bandwidth.

3.3.3 TCP traffic and multiple CBR streams

We now consider the case that multiple CBR streams and TCP traffic are sharing the link. We assume that each CBR transfer rate is 6 Mbps and the sum of CBR transfer rates are 30 Mbps in maximum and 18 Mbps on average. Therefore, the available bandwidth of TCP traffic becomes 82 Mbps on average. Figures 3.8, 3.9 and 3.10 show the total throughput characteristics in this case where TCP algorithms are Reno, NewReno and SACK. The total throughput is shown in "Multiple CBR" of Table 3.3. From the figures and table, changing CBR traffic has a great influence on the performance of Reno TCP and NewReno TCP, but not on SACK TCP.

As in the previous subsection, the average number of timeout and fast recovery occur-

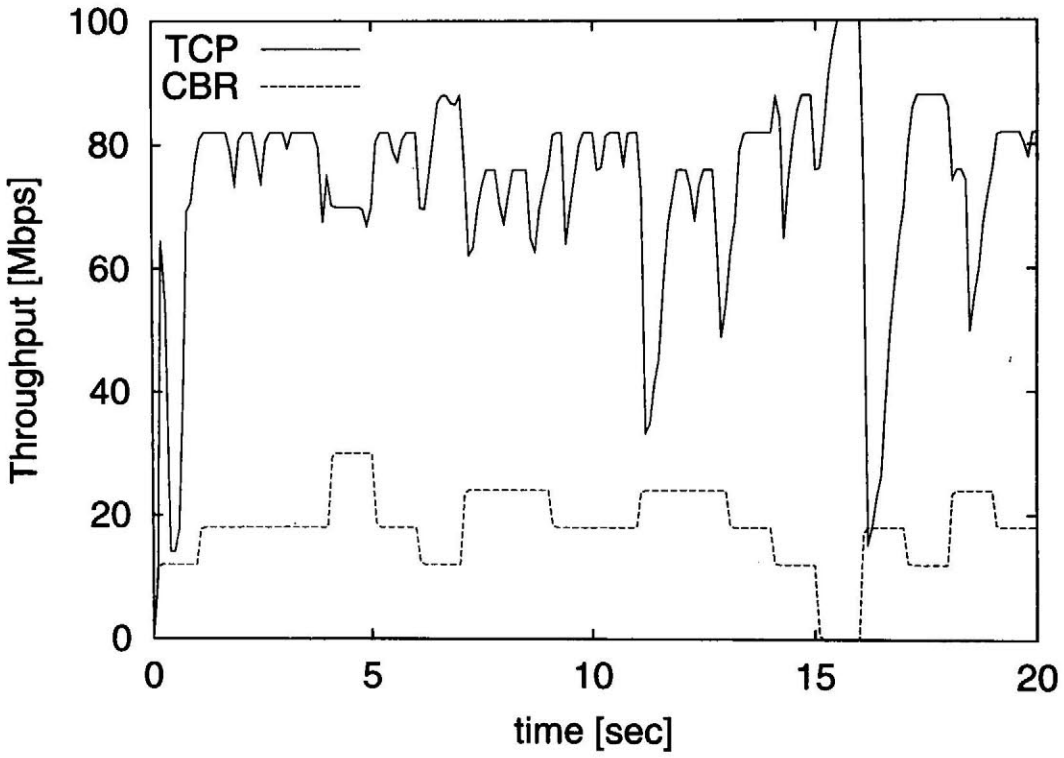


Figure 3.8: Total TCP throughput: Reno

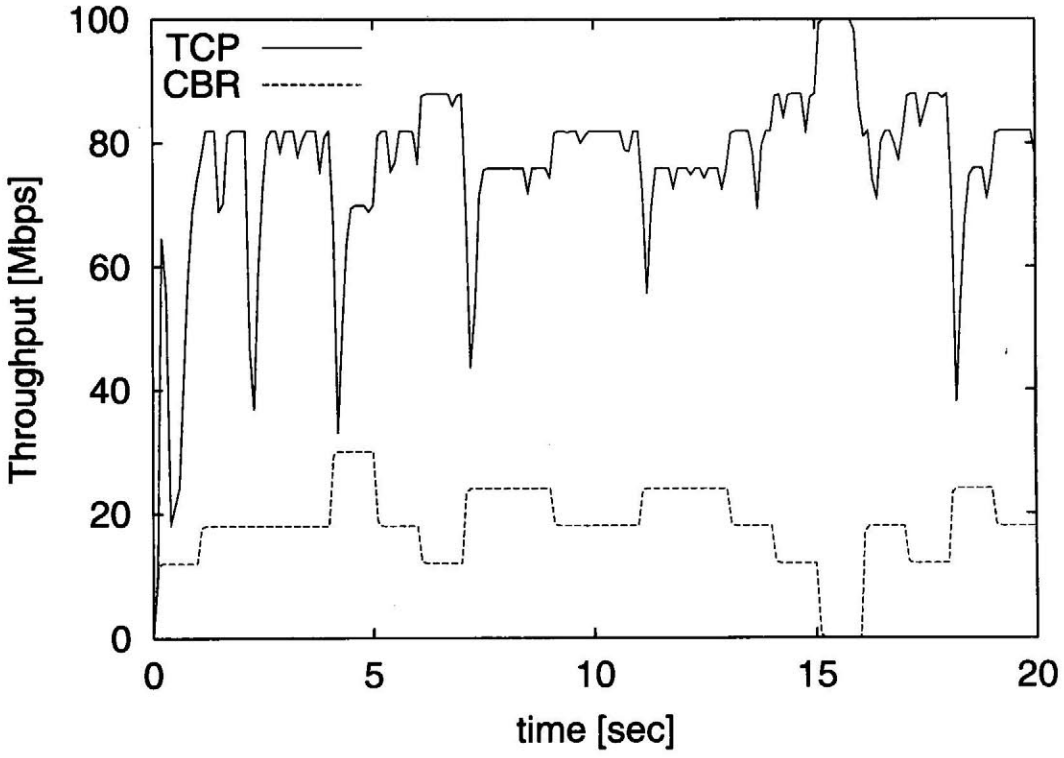


Figure 3.9: Total TCP throughput: NewReno

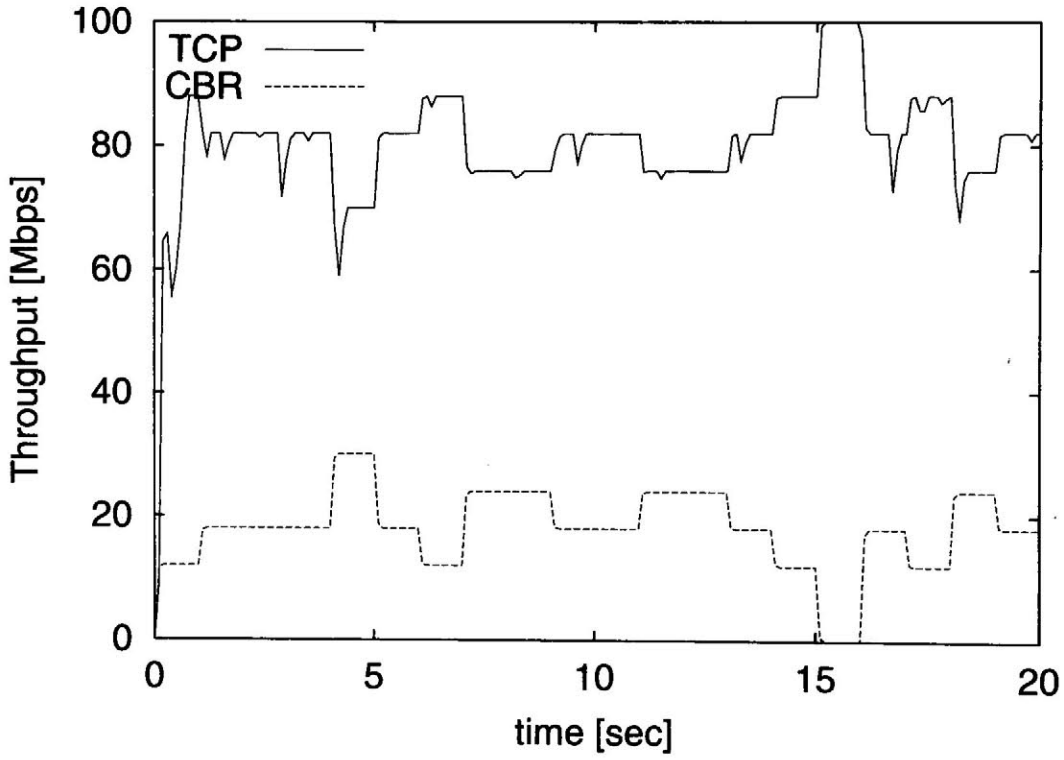


Figure 3.10: Total TCP throughput: SACK

Table 3.2: The average number of Timeout and Fast Recovery per flow

	Reno	NewReno	SACK
Timeout	5.4	0.3	0.5
Fast Recovery	26.0	22.3	24.4

Table 3.3: Total TCP throughput [Mbps]

	Reno	NewReno	SACK
TCP only	78.4	78.6	81.0
Large CBR	76.9	78.2	80.8
Multiple CBR	73.6	76.9	80.3

rence per TCP flow is shown in Table 3.2. NewReno TCP experiences timeout and fast recovery less often than SACK TCP. Nevertheless, as shown in Table 3.3, SACK TCP outperforms NewReno TCP due to effective Selective-Repeat ARQ employed, as mentioned earlier.

Furthermore, from Table 3.3, even as long as the total average bandwidth of CBR is the same, multiple CBR streams with small bandwidth has a greater influence on TCP performance than a single CBR stream with large bandwidth. This must be because multiple CBR streams frequently changes the available bandwidth for TCP and TCP can not adapt its flow control very well in response to the frequent fluctuation.

When CBR traffic decreases, the bandwidth available for TCP traffic increases. TCP can increase its congestion window size and then can transmit many packets consecutively. On the other hand, the increase of CBR traffic in this situation immediately decreases the bandwidth for TCP, which can cause multiple packet loss within one window. It will lead to the throughput degradation, especially in Reno TCP. This is because the multiple packet loss in one window causes Fast Recovery within Fast Recovery phase,

Table 3.4: The fairness of TCP flows

	Reno	NewReno	SACK
TCP only	0.27	0.24	0.20
Large CBR	0.29	0.27	0.19
Multiple CBR	0.17	0.11	0.11

which decreases congestion window size very much, and will often cause timeout. In NewReno TCP, which modified the window flow control to solve this Reno TCP's problem, the multiple packet loss in one window still requires a lot of time for recovering the lost packets in Fast Recovery phase, which in turn causes throughput degradation. Furthermore, Fast Recovery phase including multiple packet loss happens more often in the case of multiple CBR streams than in the case of only one CBR stream of large bandwidth because of frequently changing bandwidth for TCP in the former. From these reasons, the throughput is worse in the former as shown in Table 3.3.

When packet loss occurs, all the TCP variants treated here will use Fast Recovery scheme. Nevertheless, the packet transmission rate is small in Fast Recovery phase, so that throughput performance is degraded if Fast Recovery phase continues for a long duration. Furthermore, the multiple packet loss in one window causes NewReno TCP to require a lot of time for recovering the lost packets because of Go-Back-N ARQ employed. On the other hand, SACK TCP employs more efficient Selective-Repeat ARQ, which retransmits the lost packets selectively, and takes less time for recovering them than Reno TCP and NewReno TCP. For these reasons, SACK TCP can attain high throughput.

Finally, the fairness of TCP flows is shown in Table 3.4, which is defined as a coefficient of variation of throughput; fairness of 0 refers to complete fair share of the link among TCP connections. SACK TCP attains excellent fairness in throughput.

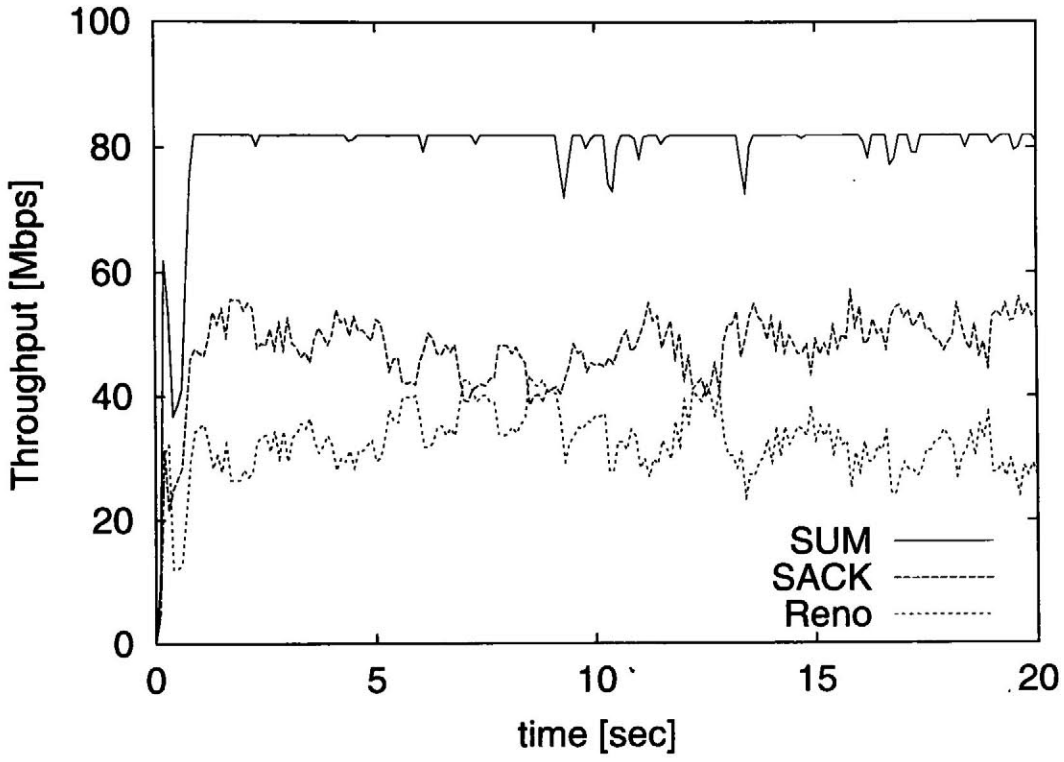


Figure 3.11: Total TCP throughput: TCP only

3.3.4 Coexistence of Reno TCP with SACK TCP

In the previous sections, we treated a homogeneous case in terms of the variant of TCP; i.e, all of 20 TCP connections were the same TCP variant. In that case, Reno TCP is the worst and SACK TCP is the best among TCP variants treated there. SACK TCP can aggressively utilize almost all the available bandwidth, while Reno TCP can not do that very well. We are thus very interested in what will happen on Reno TCP connections when a link is shared by Reno TCP connections and SACK TCP connections. Therefore, in this subsection, we deal with a heterogeneous case of link shared by 10 Reno TCP connections and 10 SACK TCP connections.

Figures 3.11, 3.12, and 3.13 show the throughput characteristics of the cases which

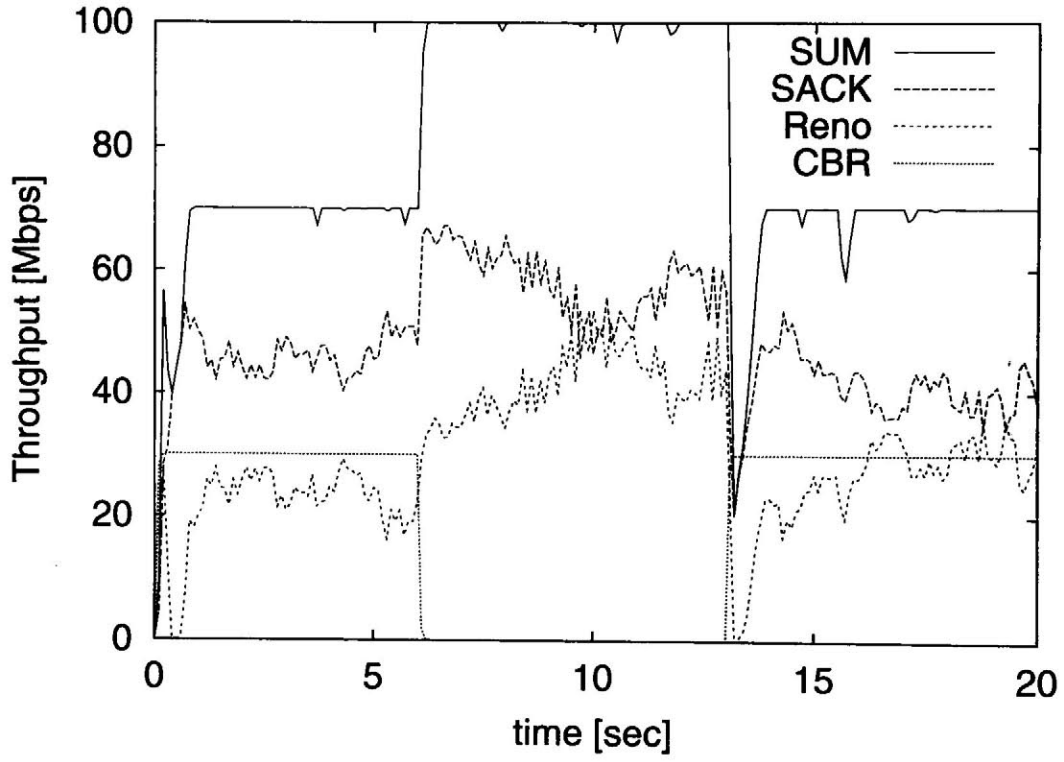


Figure 3.12: Total TCP throughput: Large CBR

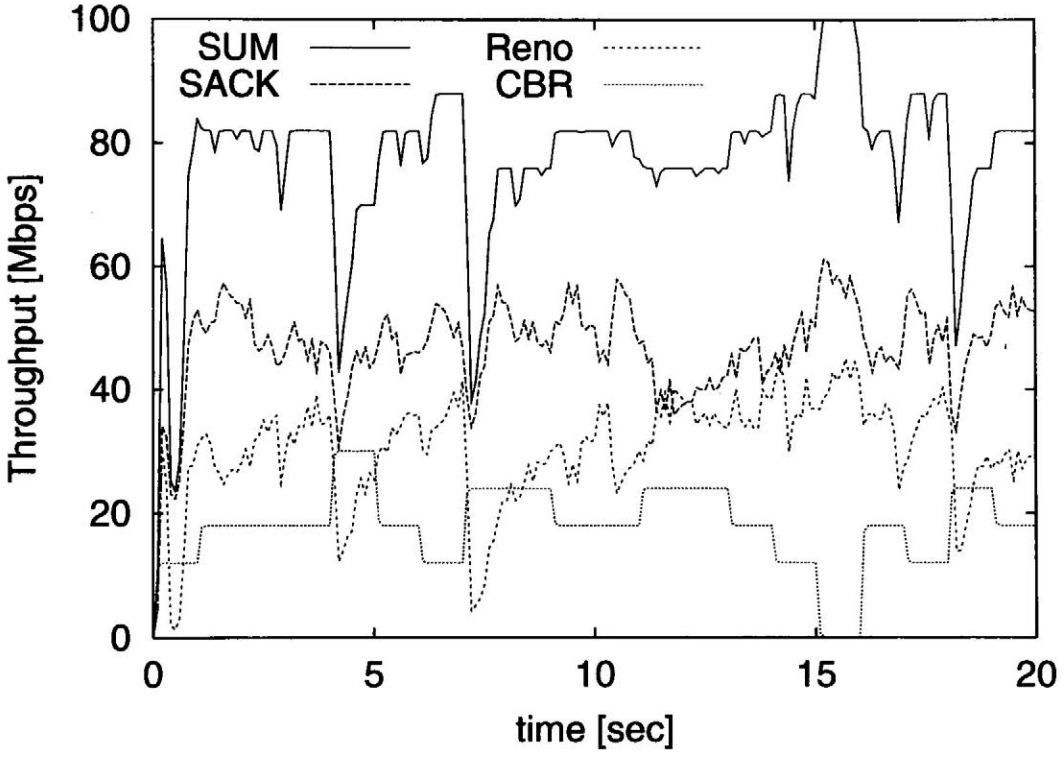


Figure 3.13: Total TCP throughput: Multiple CBR

Table 3.5: The total TCP throughput [Mbps]

	Reno		SACK		sum
TCP only	32.3	40.4%	47.6	59.6%	79.9
Large CBR	29.9	38.3%	48.2	61.7%	78.1
Multiple CBR	30.3	39.2%	47.1	60.8%	77.4

are TCP only, Large CBR, and Multiple CBR as the previous subsections. As shown in Fig. 3.11, 3.12, and 3.13, SACK TCP achieves better performance than Reno TCP at almost all time. The related total throughput performance is given in Table 3.5. By comparing Table 3.5 with Table 3.3, we see that the throughput per connection in SACK TCP is larger in the heterogeneous case than that in the homogeneous case, whereas that of Reno TCP degrades in the heterogeneous case. In addition, the total throughput of the heterogeneous case is between that of Reno TCP and that of SACK TCP in the homogeneous case. Namely, SACK TCP achieves better performance in the heterogeneous case at the cost of Reno TCP performance degradation. Nevertheless, from Table 3.5, the ratio of the total throughput of Reno TCP is kept approximately at 40 percent in any case listed there, regardless of whether CBR traffic exists or not.

Next, the fairness of TCP flows is shown in Table 3.6, which is defined as a coefficient of variation of throughput as mentioned earlier. SACK TCP can achieve fair share of the link among TCP connections.

From the above results, the effect of coexistence with SACK TCP on performance of Reno TCP does not depend upon deployment of QoS networks.

3.3.5 Effect of fluctuation of interarrival times of CBR packets

As mentioned in Sect. 3.2, we assumed that CBR traffic has a constant packet interval time as a result of using a hardware shaper of fine granularity. This looks an ideal model,

Table 3.6: The fairness of TCP flows

	Reno flows	SACK flows	All flows
TCP only	0.25	0.17	0.28
Large CBR	0.22	0.17	0.30
Multiple CBR	0.12	0.04	0.23

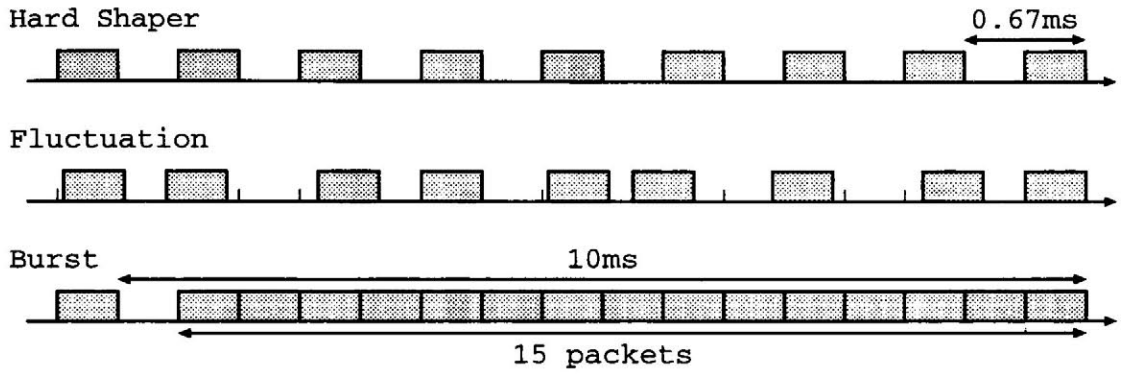


Figure 3.14: The model of the interarrival times of CBR packets

but the interarrival time of CBR packets can generally fluctuate in actual networks. In this section, we consider the case that the interarrival time of CBR packets fluctuates or can be bursty sometimes for the following reason. CBR traffic goes through multiple gateways, in each of which CBR packets can be forced to wait. Thus, interarrival times of packets of some CBR stream will not be constant to some gateway.

We treat two cases as shown in Fig. 3.14; one is the case where packets are randomly delayed in at most one packet generation interval of 0.67 msec. Note that when CBR packets of 500 bytes are transmitted at 6 Mbps, their packet intervals become 0.67 msec. Another is the case where some packets arrive back to back at the gateway. Here, 15 packets are transmitted back to back in an interval of 10 msec. The former is indicated as "Fluctuation", and the latter "Burst" in Table 3.7 and 3.8. When TCP traffic shares

Table 3.7: Total TCP throughput [Mbps]

	Hard Shaper	Fluctuation	Burst
Reno	75.0	73.3	74.5
NewReno	77.5	77.8	77.6
SACK	80.5	80.3	80.6

Table 3.8: The fairness of TCP flows

	Hard Shaper	Fluctuation	Burst
Reno	0.16	0.23	0.22
NewReno	0.13	0.17	0.21
SACK	0.10	0.13	0.16

the link with multiple CBR streams (Multiple CBR), the total throughput of TCP traffic and the fairness of TCP flows in the above two cases are shown in Table 3.7 and 3.8. Fairness in Table 3.8 is defined as coefficient of variation of throughput as mentioned earlier. Table 3.7 and 3.8 show that the total throughput is not affected by the fluctuation of the interarrival times of CBR packets in particular in NewReno TCP and SACK TCP, but the fairness can be improved by use of hardware shaper of fine granularity.

3.4 Conclusions

In our research, we have examined the performance of three TCP variants in QoS networks by means of simulations. In QoS networks, there can be some traffic which has priority over TCP traffic so that TCP will use the unused bandwidth left by high priority traffic; CBR traffic is treated here as high priority traffic. Therefore, the amount of available bandwidth for TCP traffic can change time to time. We have thus shown how they are affected by changing available bandwidth. Among them, Reno TCP degrades most due to

changing available bandwidth. SACK TCP outperforms other two TCP variants in QoS networks as well as in the current Internet. Moreover, we studied how the fluctuation of interarrival times of CBR packets affects the performance of TCP. The total throughput of TCP connections is not affected by it, but it deteriorates the fairness of throughput among them. Nevertheless, SACK TCP is very robust against it.

SACK TCP works very well in QoS networks in terms of its throughput performance.

Chapter 4

Out-of-Sequence in Packet Arrivals due to Layer 2 ARQ and Its Impact on TCP Performance in W-CDMA Networks

4.1 Introduction

With the growth of wireless networks and the Internet, the data transmission service over wireless networks becomes more attractive. In the current Internet, TCP (Transmission Control Protocol) [Ste94], which is a reliable end-to-end transport protocol in the Internet Protocol suite, is widely used in popular applications like Telnet, FTP, and HTTP. TCP has been tuned for wired networks where the error rates are very low and packet losses occur mostly because of congestion in the networks. Therefore, TCP performs well over such networks by adapting its transmission rate to end-to-end delays and packet losses caused by congestion [JK88]. However, when a TCP sender transmits packets over wireless links characterized by high bit error rates, the performance of TCP can be drastically degraded. This is because TCP recognizes packet loss as a result of congestion even

if it happens due to bit error over wireless links, and hence reduces a transmitting rate [CLM99, BKVP97, CZR98].

On the other hand, the next generation mobile system, called IMT-2000 (International Mobile Telecommunications-2000) [IMT], has been standardized by ITU (International Telecommunication Union) so as to provide faster data transmitting service than the current mobile system and worldwide roaming capability. There is W-CDMA (Wideband-Code Division Multiple Access) technology prescribed by 3GPP (3rd Generation Partnership Project) [3GPa] for wireless environments as one of what IMT-2000 system employs. It provides a transmission rate of 384Kb/s maximum outdoors, and uses strong error recovery technologies in order to minimize the damage of the utilization of the wireless channels due to bit error. In fact, 3GPP employs FEC (Forward Error Correction) coding and ARQ (Automatic Repeat reQuest) mechanism as error recovery schemes in Layer 2 of wireless networks. Several alternatives are specified there to implement them and there are many related parameters, while the choice of recovery schemes and the parameterization are left to operators. The performance of TCP over IMT-2000 system has not yet been investigated clearly in that context although one over wireless LAN and old IS-99 system has been examined, e.g., in [BPSK96, CB00, CZ99].

Our major purpose is to investigate how the performance of TCP is affected by the packet losses caused by transmission errors and the ARQ mechanism of Layer 2 over wireless links by means of simulations. Retransmissions should be done if it is necessary, but we suppose that the maximum number of retransmission should be usually limited. Furthermore, packets can arrive at their destination out of order due to Selective-Repeat ARQ employed in W-CDMA. This can make TCP receiver return some duplicate acknowledgments, thereby causing wasteful retransmissions even if no packet gets lost in fact. In order to avoid it, some management will be necessary to keep sequence integrity of packet delivery at the receiver.

Therefore, in this study, we will extensively examine the performance of TCP; in

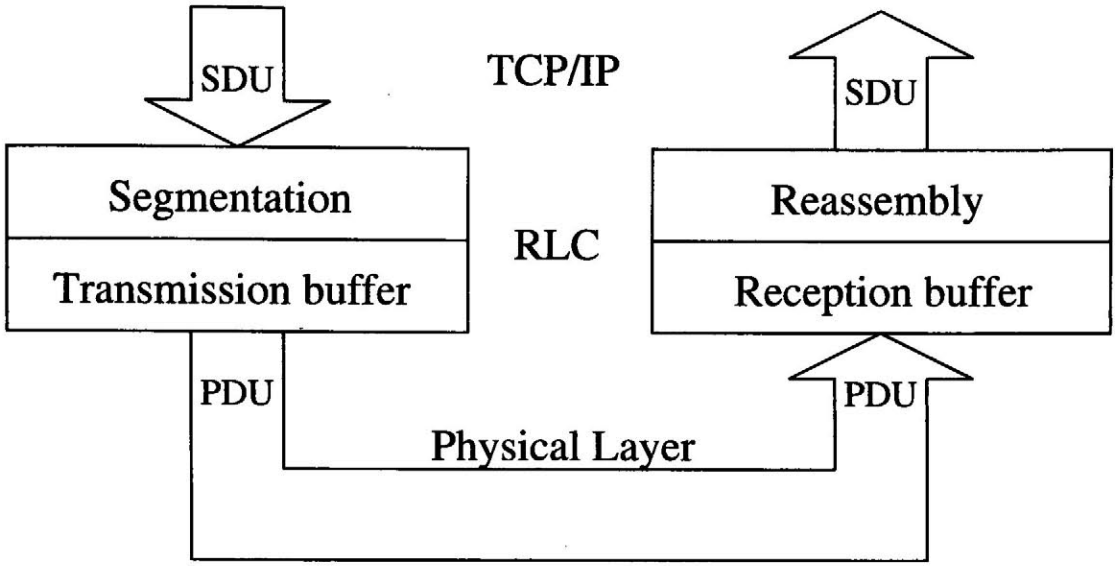


Figure 4.1: SDU/PDU in RLC protocol

particular, the impact of the maximum number of allowable retransmissions on Layer 2, TCP packet size, and the available bandwidth on the wireless link. Through these simulation results, we will show that a packet management for sequence integrity should be executed effectively to avoid the performance degradation. As a result, we further study an effective way for it, and discuss some appropriate parameterization from viewpoints of both throughput and transmission delay performance.

4.2 TCP and Layer 2 in Wireless Networks

In W-CDMA environment, RLC (Radio Link Control) protocol [3GPc] is standardized by 3GPP as Layer 2 protocol. The RLC protocol divides a SDU (Service Data Unit) corresponding to IP datagram received from the upper layer into several PDUs (Protocol Data Unit) of 42 bytes and transmits them as shown in Fig. 4.1. It employs Selective-Repeat ARQ using Polling/Status Report for recovering lost PDUs. The four alternative schemes

of the SDU discard are defined in [3GPc]. They are timer-based discard with explicit signaling, timer-based discard without explicit signaling, discard after maximum number of transmissions, and no discard after maximum number of transmissions. However, any values are not recommended for their related parameters. In addition, 3GPP specifies a way of preserving the order of packet delivery by keeping some PDUs waiting in Layer 2 of the destination if their preceding PDUs suffer bit error and thus are being transmitted. If this function defined in [3GPb] is not used, out-of-order delivery will occur. Therefore, we focus the maximum number of allowable retransmissions on Layer 2 ARQ and the effect of keeping sequence integrity of packet delivery.

In the Internet, Tahoe, Reno, NewReno [FH99], and SACK [MMFR96] are used as Transport Protocol. Tahoe TCP is the oldest implementation and has the most simple algorithm. It uses a fast retransmit mechanism to recover lost packets. Reno TCP has more effective flow control by making use of a fast recovery mechanism. NewReno TCP has been proposed to resolve Reno's problem that the throughput performance is degraded due to multiple packet losses in one window. That is implemented mostly on current Internet computers. SACK TCP employs a more efficient retransmission mechanism, i.e., Selective-Repeat ARQ different from Go-Back-N ARQ used by other TCP variants. We will employ NewReno TCP in this research because it is mainly used in the current Internet.

Recently, several schemes improving the performance of TCP for wireless networks have been proposed. For example, it has been examined that the performance of TCP can be improved if TCP can distinguish packet loss due to bit error over wireless links from that due to network congestion over wired networks [SF98]. The schemes to enable it such as ELN (Explicit Loss Notification) [BPSK96], EBSN (Explicit Bad State Notification) [BKVP97], and ELNR (Explicit Loss Notification to the Receiver) [MV97] are proposed, but are very difficult to implement. Furthermore, the Indirect-TCP (I-TCP) which splits a TCP connection into a connection over wired link and one over wireless link is proposed

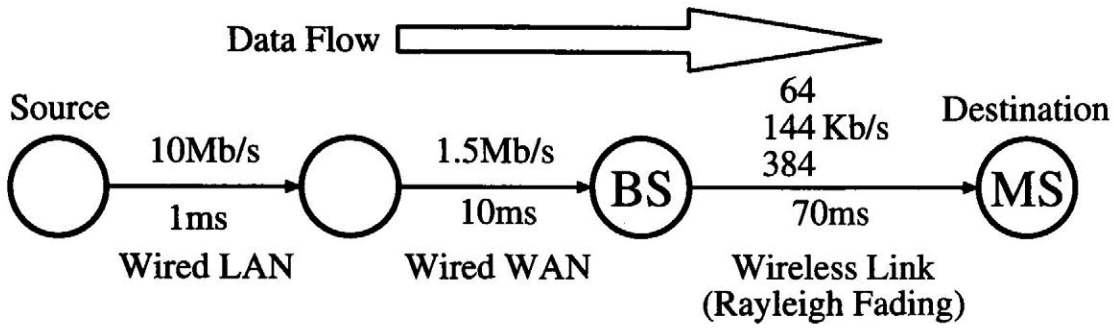


Figure 4.2: Simulation Model

to choose the suitable TCP algorithms for each of wired and wireless links. However, using the scheme breaks the end-to-end model of the TCP connection. Therefore, we focus the performance of TCP variants without any additional schemes, as used on the Internet, over wireless environment.

4.3 Simulation Model

In this section, we describe the simulation model used here. We use the VINT Network Simulator NS Version 2 [NS] after we added some modifications for Layer 2 ARQ scheme over wireless links for our research.

In our simulation, one TCP source transmits packets over from wired link to wireless link as shown in Fig. 4.2. One wired link is 10 Mb/s in bandwidth and 1 msec in propagation delay like in LAN (Local Area Network). Another wired link is 1.5 Mb/s in bandwidth and 10 msec in propagation delay like in WAN (Wide Area Network). The wireless link is 64, 144, or 384 Kb/s in bandwidth and 70 msec in propagation delay including the processing delay in the base station (BS) and the mobile station (MS). The BS in Fig. 4.2 includes RNC (Radio Network Controller). We assume the wireless link suffers a burst error caused by Rayleigh Fading [ASS98]. More specifically, users are walking

Table 4.1: The transport block size and the TTI for each link bandwidth

Link bandwidth [Kb/s]	Transport block size [bit]	Transmission Time Interval (TTI) [msec]
64	1280	20
144	2880	20
384	7680	20

with mobile terminals, which resulting in the two-path fading channel with Doppler frequency of 5 Hz. Also, a power control is performed. The physical layer encodes Layer 2 PDUs in a Transmission Time Interval (TTI), and bit error is occurred in the TTI. The transport block size and the TTI for each link bandwidth are summarized in Table 4.1. The average error rates of Layer 2 considered here are 1.8, 4.1, and 8.6 percent in FER (Frame Error Rate) with FEC for link bandwidth of 64 Kb/s, 1.9, 3.5, and 8.1 percent for that of 144 Kb/s, and 1.2, 2.6, and 5.9 percent for that of 384 Kb/s. It is assumed that the ACK (Acknowledgment) and NACK (Negative ACK) in Layer 2 are error-free.

In this simulation, it is assumed that TCP traffic is used for greedy file transfer. The TCP variant employed here is NewReno as mentioned above. The TCP packet size is set to 256, 512, 1024, and 1500 bytes, and the Layer 2 PDU size is set to 42 bytes; the header is 2 bytes and the payload is 40 bytes.

Each node is equipped with a buffer of infinite capacity, and packet loss thus does not occur in any node. This allows us to focus on the performance of TCP over wireless links. We carry out simulation experiments for 80 seconds, and discuss the average throughput performance of TCP.

4.4 Simulation Results

In this section, we will show the characteristics of TCP performance in wireless networks by means of simulation results. On Layer 2, if a PDU suffers bit error which cannot be recovered by FEC, it will be retransmitted by Selective-Repeat ARQ. It may happen that some PDU needs multiple retransmissions, so that a number of retransmissions per PDU are allowed, while the maximum number is determined.

Therefore, in subsection 4.4.1, we will first discuss the effect of the maximum number of allowable retransmissions. In the Selective-Repeat ARQ employed on Layer 2, packets can inherently arrive out of order at their destination. This can cause retransmission in TCP although any packet in fact does not get lost. This further results in throughput performance degradation of TCP. In subsection 4.4.2, we will thus treat the issue related to how to keep sequence integrity of packet arrivals at the destination, and examine an effective solution to the issue.

4.4.1 Fundamental features of Layer 2 ARQ

In this section, we consider the case that the Selective-Repeat ARQ is used as Layer 2 ARQ and the assembled SDU is sent up to the upper layer as soon as it arrives at the mobile station. Consequently, TCP packets can arrive out of order at their destination.

First, we examine the average throughput performance of TCP when the maximum number of allowable retransmissions on Layer 2 varies from zero to five. Fig. 4.3 shows the throughput characteristics of 64 Kb/s in bandwidth of the wireless link. The TCP packet size is set to 512 bytes. From the figure, Layer 2 ARQ improves the performance of TCP, and it can achieve the best performance of TCP in particular when the number of allowable retransmissions is larger than 1. Therefore, in the following results, we will deal with the case that the maximum number of allowable retransmissions is set to 5.

Next, we examine the throughput performance of TCP in the case that the TCP packet

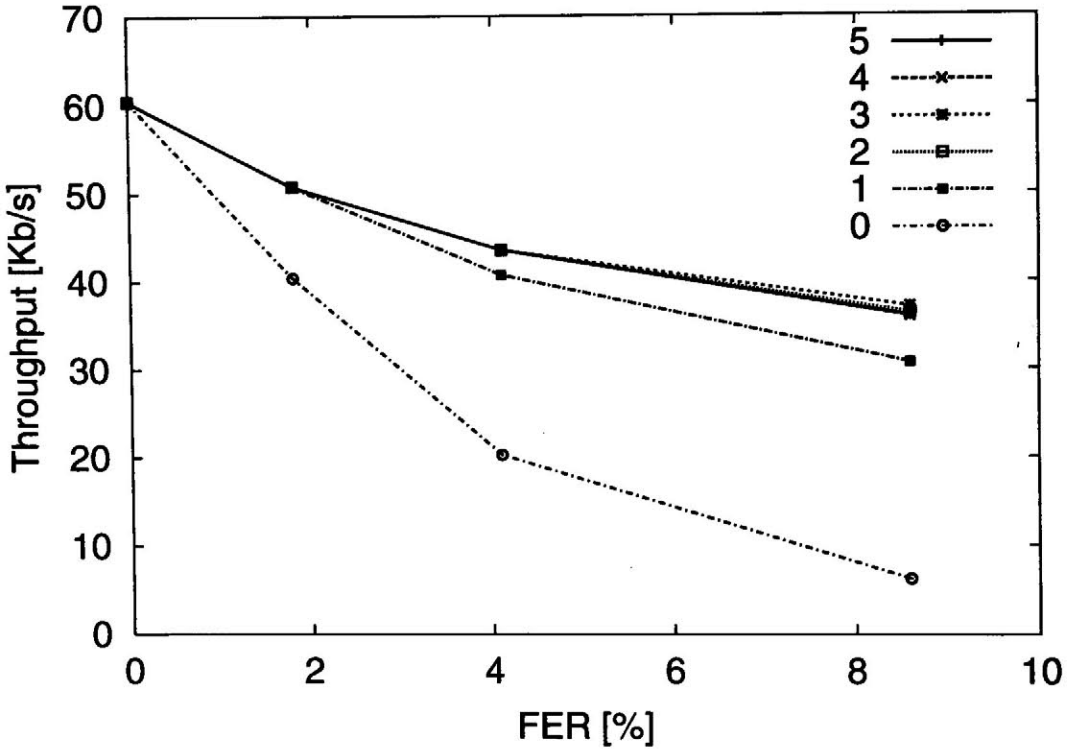


Figure 4.3: The effect of the maximum number of allowable retransmissions (64Kb/s)

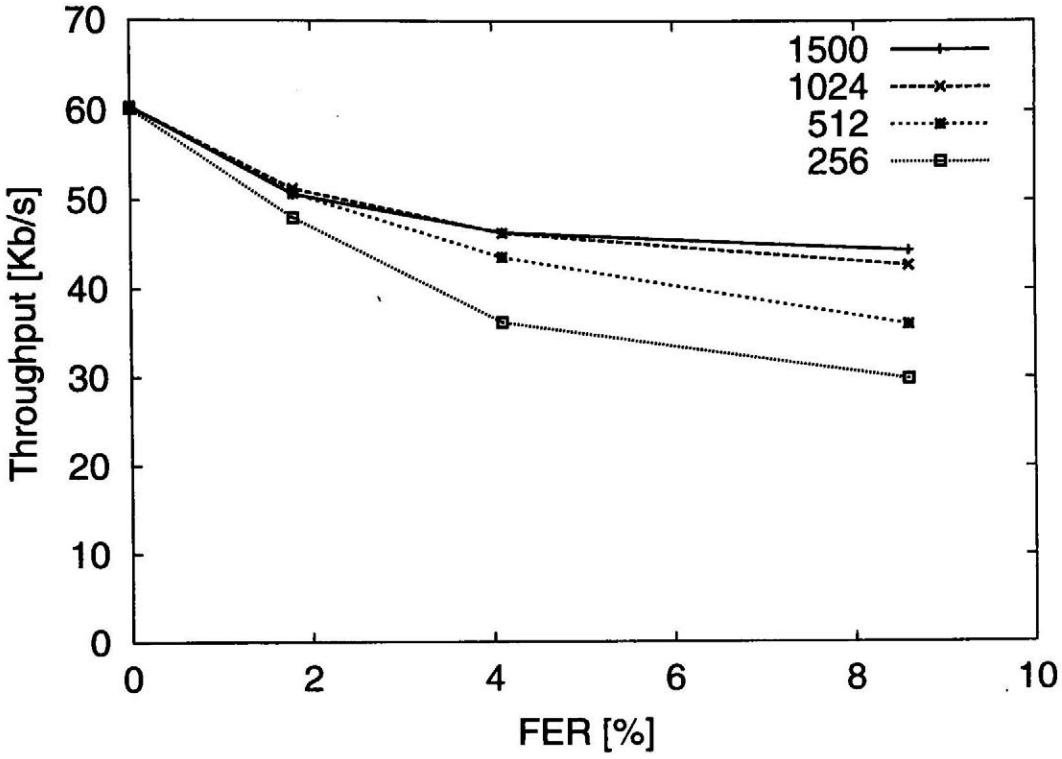


Figure 4.4: The effect of TCP packet size (64Kb/s)

size is set to 256, 512, 1024, or 1500 bytes. Fig. 4.4 shows the average throughput performance of TCP over a wireless link of 64 Kb/s. From this figure, larger TCP packets can attain better throughput performance when Layer 2 ARQ is employed. This is because Layer 2 ARQ gets rid of an adverse effect of transmission error over wireless links for TCP.

Lastly, we summarize the throughput performance of TCP over wireless links of a different bandwidth in Fig. 4.5. In the simulations, retransmissions are performed at most five times per PDU if necessary. The TCP packet size is 1500 bytes. From this figure, a larger bandwidth leads to less improvement; e.g., the achievable throughput is approximately 160 Kb/s (40 percent) over a link of 384 Kb/s for FER of 5.9 percent even if the ARQ is employed.

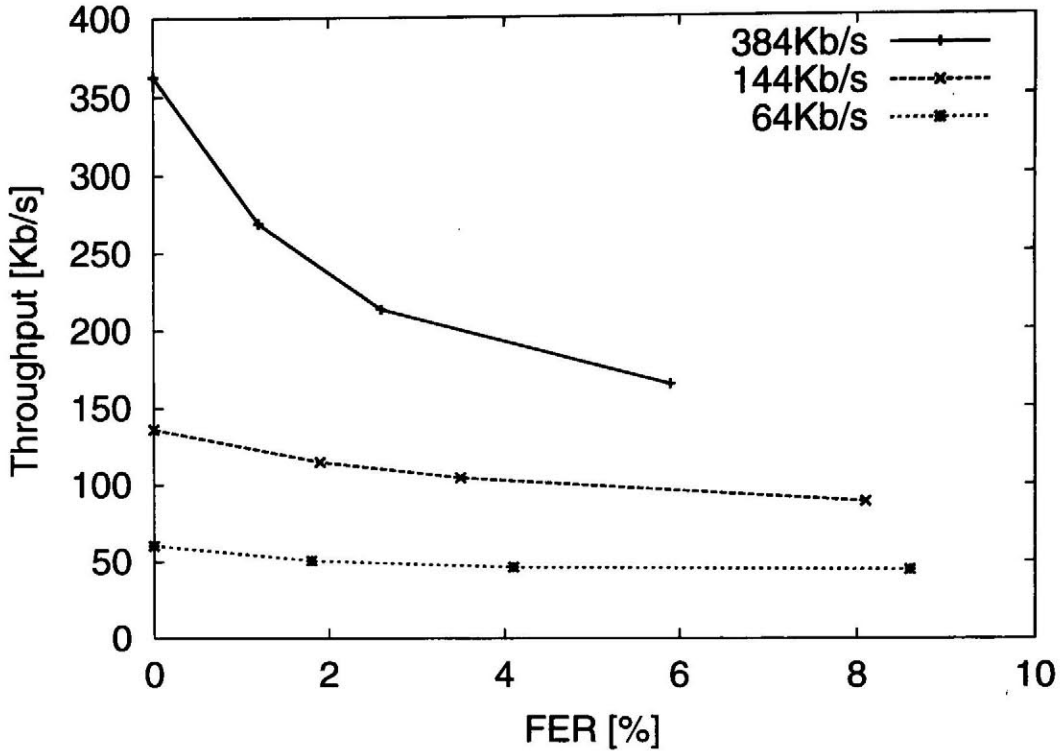


Figure 4.5: The effect of link bandwidth

We will discuss the reason and solution for this problem in the following subsection.

4.4.2 The effect of packet management

Let's investigate the reason for the above phenomenon occurring over a link of a larger bandwidth. For this purpose, we show how the *cwnd* (congestion window size) of TCP is changing in Fig. 4.6. From this figure, it can be seen that the *cwnd* cannot increase monotonously, but is oscillated very frequently over a narrow range of small values. This is because packets can arrive out of order at the mobile station due to the Selective-Repeat ARQ in Layer 2. This causes the TCP receiver to send duplicate ACKs back to the TCP sender, which in turn retransmits the packet associated with the ACK although the packet

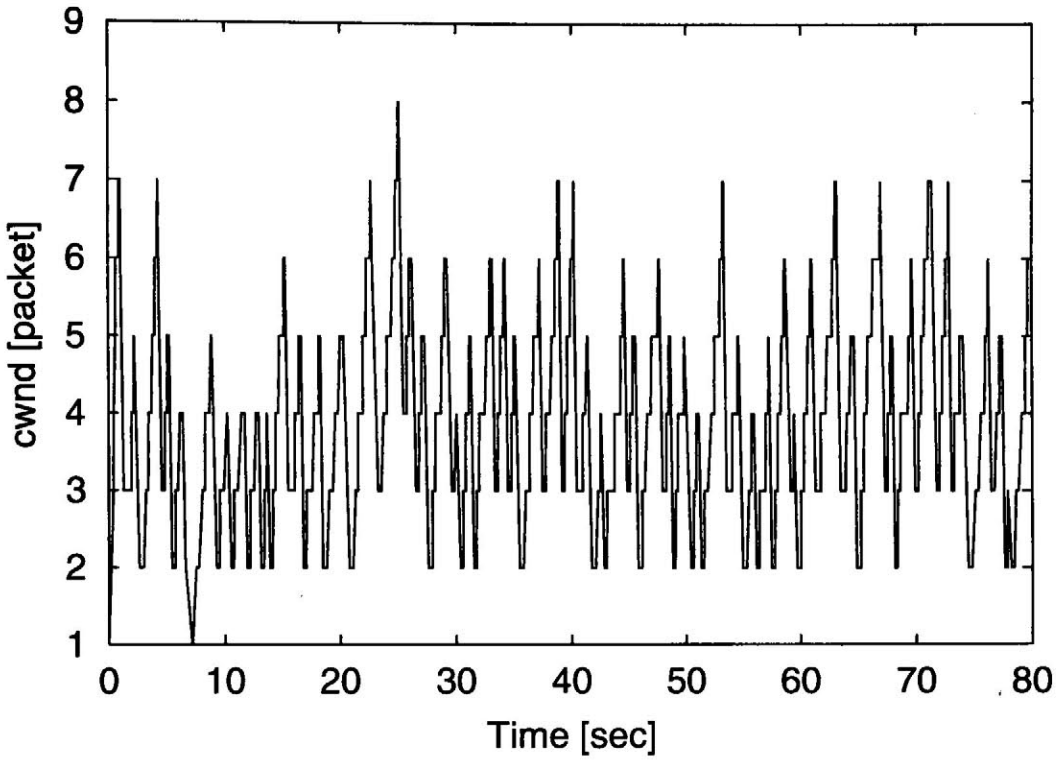


Figure 4.6: cwnd of TCP (FER=5.9%)

is not lost in fact. At the time, its *cwnd* is reduced to half of it. Moreover, the TCP performance is heavily degraded with FER, in particular, on the link of a large bandwidth. Suppose that some PDU is lost. It will be very likely that multiple PDUs following it arrive without any loss before the retransmission of the lost PDU succeeds. That will happen more frequently as the bandwidth is larger because more PDUs can be sent for some duration taken by the retransmission. This prevents TCP from efficiently utilizing the wireless link of a large bandwidth.

Therefore, it is necessary that the order of packet delivery is preserved. To do this, we consider the case of preserving the order of packet delivery in the mobile station with Selective-Repeat ARQ. Some Layer 2 PDUs are kept waiting in Layer 2 of the mobile station until the timeout timer expires, and the IP datagram is sent up to the upper layer

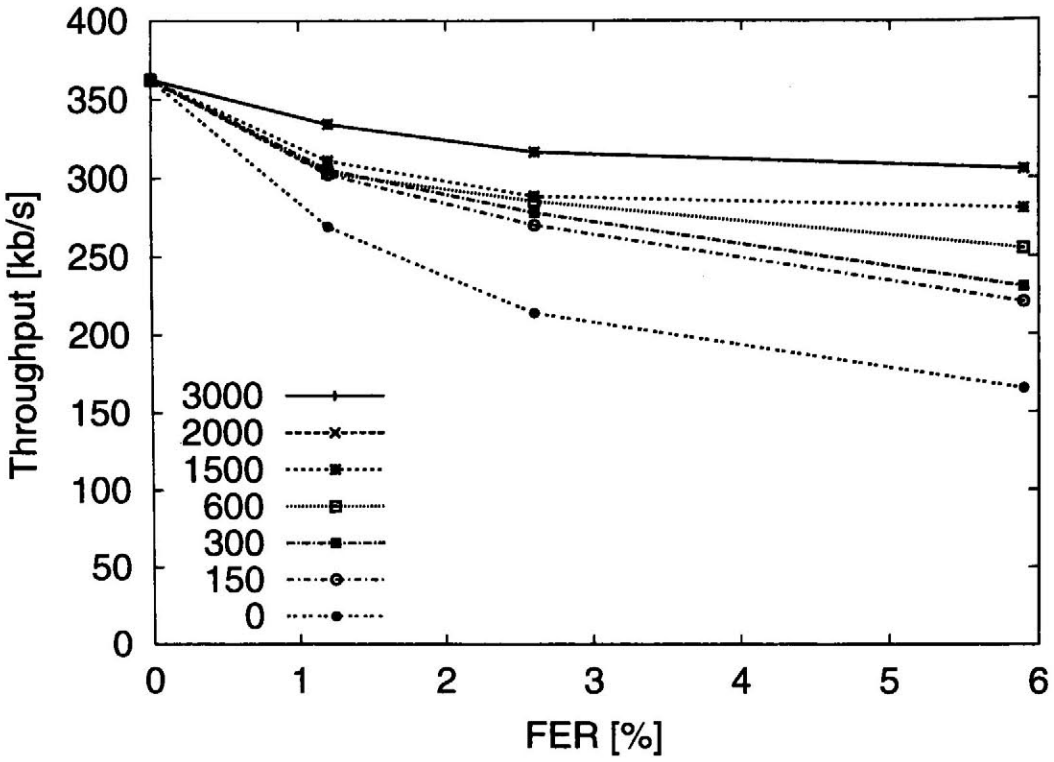


Figure 4.7: The effect of packet management in the mobile station

in order if all of the preceding PDUs transmitted arrive at the mobile station within the Waiting Timeout value (WT).

Fig. 4.7 shows the average throughput performance of TCP in the case that the Waiting Timeout value (WT) varies from 0 msec to 3000 msec; the case that timeout value is 0 msec indicates that packets are not reordered in the mobile station. Table 4.2 shows how many packets arrived out of order and how many times the TCP retransmission occurred in the simulations. From the figure and the table, it can be seen that the larger WT value of reordering packets achieves the excellent throughput performance of TCP because of avoiding wasteful retransmission caused by the out-of-order packet delivery.

Next, we consider the buffer sizes of the mobile station and the base station required for reordering packets. Fig. 4.8 shows the maximum queue length of TCP packets waiting

Table 4.2: The number of out of order and TCP retransmission occurrence

WT value	Total packet	Out of Order		TCP Timeout	TCP Fast Recovery
0	1199	182	15.2%	1	67
150	1596	50	3.1%	0	42
300	1700	22	1.3%	0	34
600	1831	9	0.5%	0	15
1500	1981	1	0.1%	0	5
2000	2085	0	0.0%	0	0
3000	2085	0	0.0%	0	0

in the mobile station during the simulation. And, Fig. 4.9 shows the maximum queue length of TCP packets in the base station. From these figures, the buffer sizes needed for reordering packets in both of the mobile station and the base station increases as the *WT* value increases for any FER.

Lastly, we have made simulations 1000 times in each of which a file of 500 Kbytes like a typical web page is transmitted over a wireless link of 384 Kb/s suffered from different error patterns. Fig. 4.10 shows the transmission delay time when the FERs are 1.2, 2.6, and 5.9 percent. From this figure, as a *WT* value increases, the transmission delay time increases, in particular for FER of 5.9 percent. When FER is relatively large, it may happen that Layer 2 ARQ cannot recover packets damaged by bit error with a limited number of retransmissions. This will lead to packet losses, and will keep TCP receivers waiting during a long time if a large timeout timer is employed. Moreover, some lost packets cause the timeout in TCP, by which TCP performance will be further degraded. Fig. 4.11 shows how the *cwnd* of TCP is updated when the *WT* value is 2000 msec and FER is 5.9 percent (see Fig. 4.10 for the corresponding TCP performance). The figure illustrates that duplicate ACKs caused the fast recovery, and further timeout in

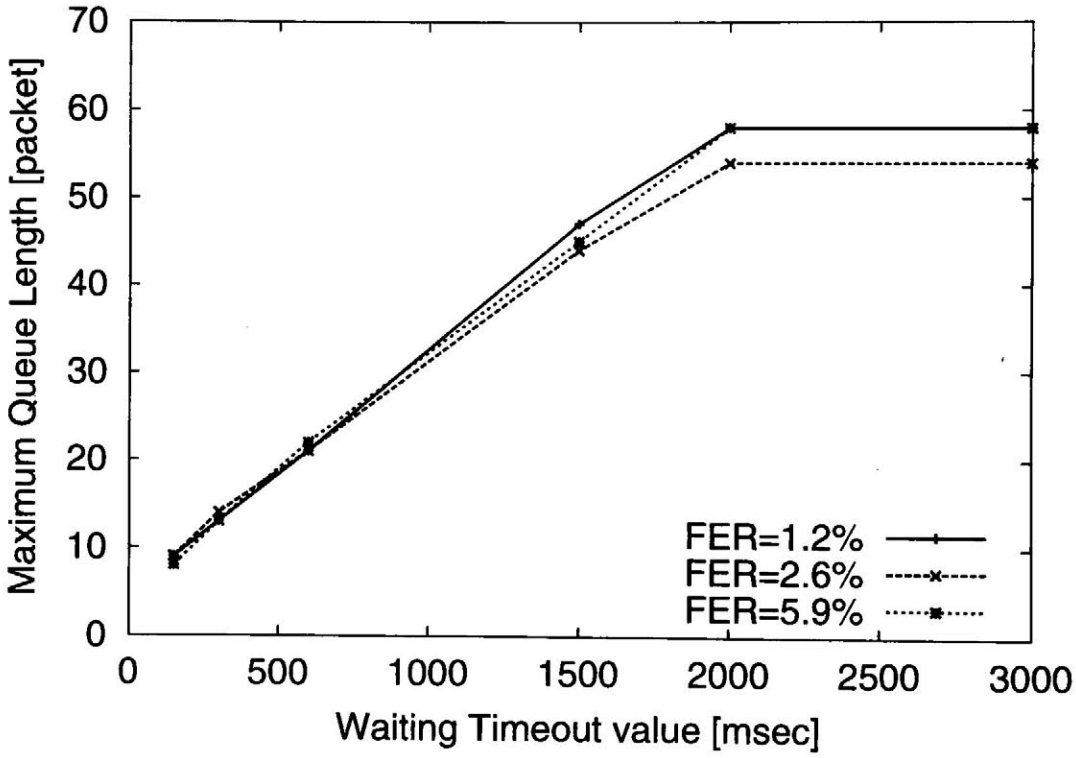


Figure 4.8: Maximum queue length in the mobile station

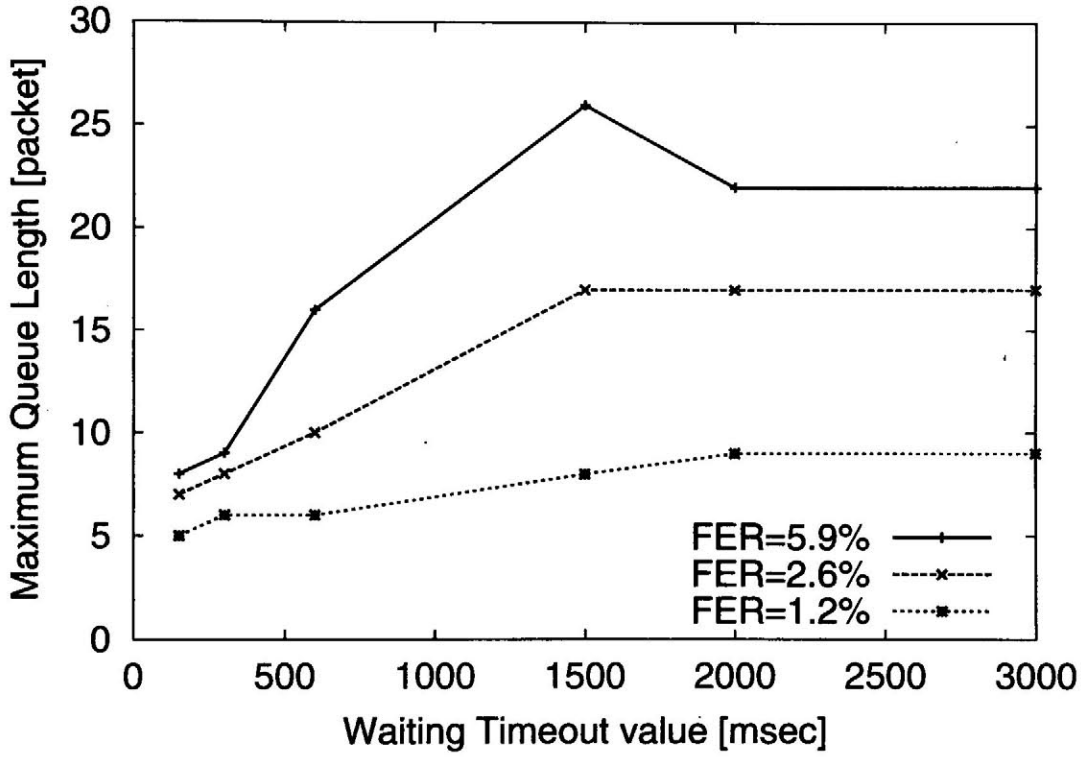


Figure 4.9: Maximum queue length in the base station

TCP because packet loss unfortunately occurs again during the fast recovery phase. This leads to a large transmission delay time, so that the characteristics of transmission delay time can be degraded as shown in Fig. 4.10. In contrast, the larger WT value can attain better throughput performance of TCP as mentioned above. Therefore, there is a trade-off between the throughput and the transmission delay time performance, resulting in choice of an adequate timeout value.

Finally, let us study the impact of packet length again. As we have seen in Fig. 4.4, the throughput performance is getting better with the increase of packet length. All packets can be recovered there with a limited number of retransmissions in all the cases. On the other hand, when FER is very large, packets of 1500 bytes can suffer from large transmission delay time as shown in Fig. 4.12. This is because longer packets more likely contain Layer 2 PDUs damaged by bit error so that some of them further may not be recovered with a limited number of retransmissions. Therefore, there exists adequate packet length in terms of achievable throughput and transmission delay time. In the obtained figures, packets of 1024 bytes or so can be recommended.

4.5 Conclusions

In our research, we have extensively examined the performance of TCP over wireless networks of IMT-2000 by means of simulations. In wireless networks, the packet losses occur due to the burst errors caused by Rayleigh Fading. We have thus shown how the TCP performance is affected by both the packet losses caused by transmission errors and ARQ mechanism of Layer 2 over such networks. Furthermore, packets can arrive at their destination out of order due to Selective-Repeat ARQ employed in W-CDMA. This can make TCP receiver return some duplicate acknowledgments, thereby causing wasteful retransmission even if no packet gets lost in fact. Therefore, we have studied an effective way of managing packets for sequence integrity to improve the performance of TCP, and

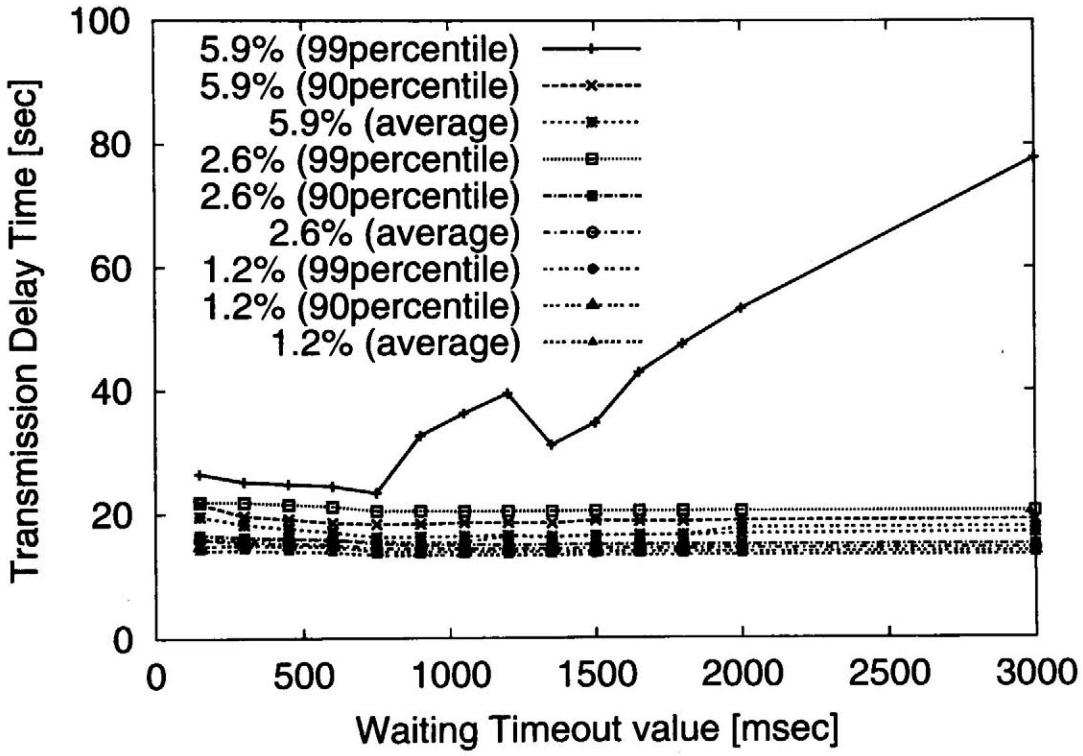


Figure 4.10: The transmission delay time

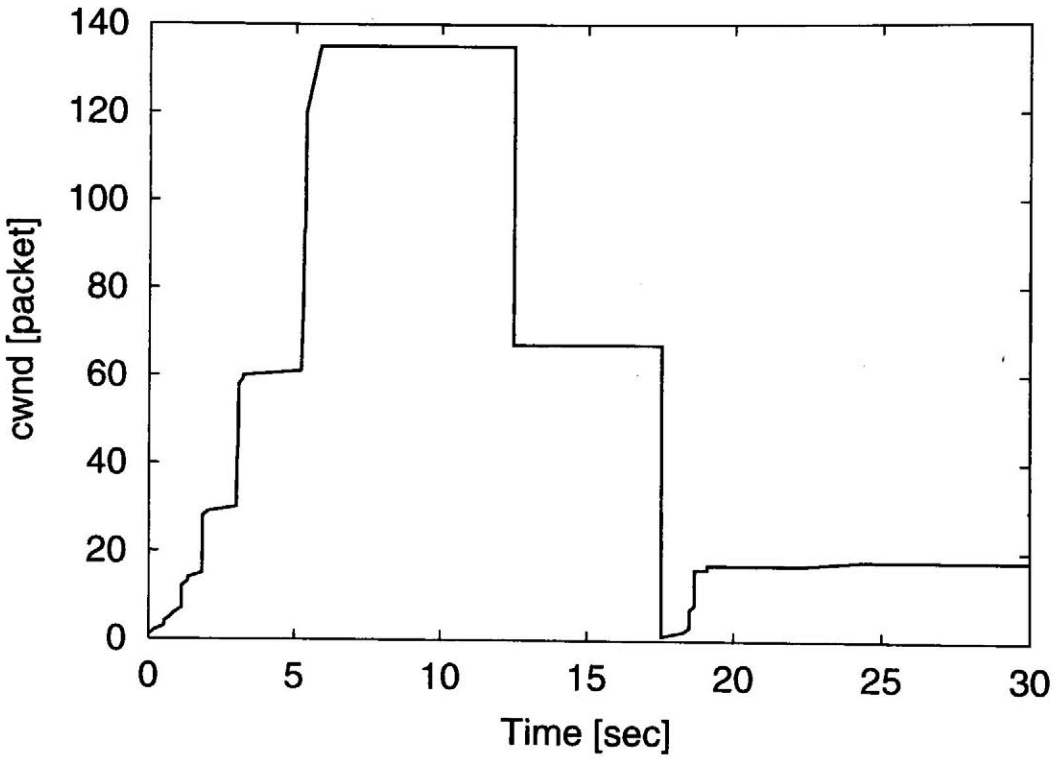


Figure 4.11: cwnd of TCP (WT value=2000msec)

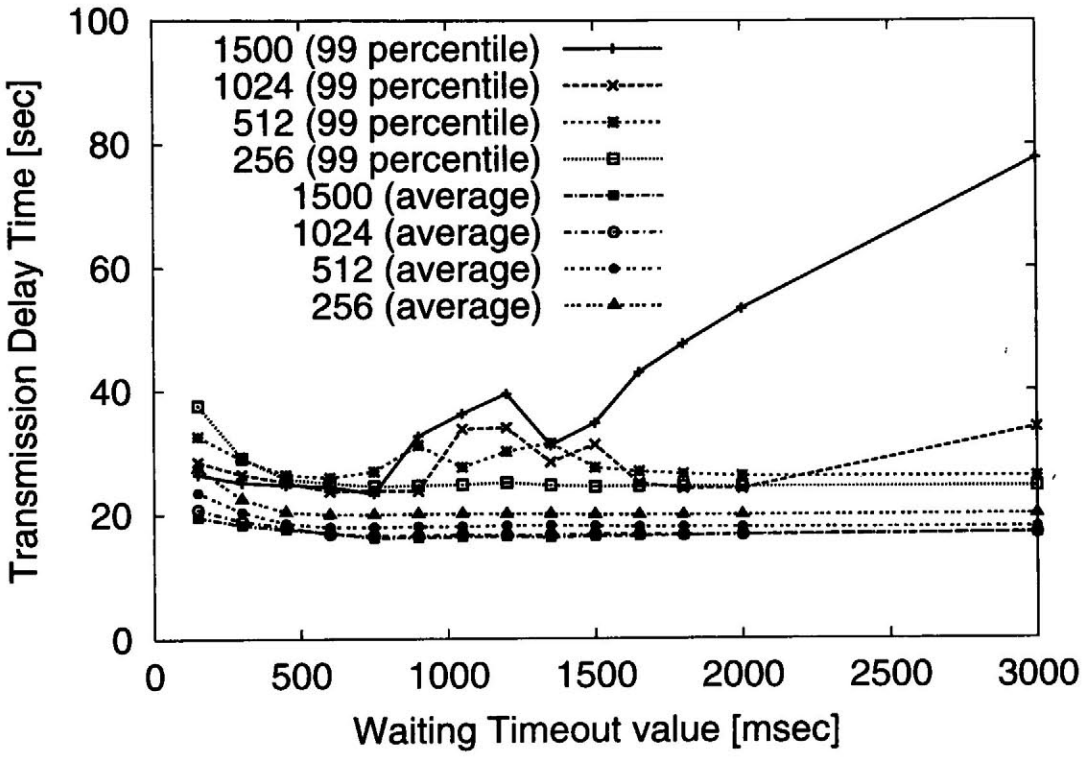


Figure 4.12: The transmission delay time: effect of TCP packet size (FER=5.9%)

discussed some appropriate parameterization from viewpoints of both achievable throughput and transmission delay performance.

First, through our simulations, with Layer 2 ARQ, TCP can achieve better throughput performance with packets of larger size. This is because Layer 2 ARQ gets rid of an adverse effect of transmission error over wireless links for TCP. Next, Layer 2 ARQ is less effective in improving the throughput performance of TCP on the link of a larger bandwidth. However, the reordering packets for keeping sequence integrity of packet arrivals at the mobile station improves the throughput performance of TCP even if the link bandwidth is large. The timeout timer is employed to quit waiting for successful retransmission on Layer 2. The larger Waiting Timeout value (WT) needs the reorder buffer of larger size in both the base station and the mobile stations, and can also degrade the transmission delay time for TCP packets when FER is relatively large. In addition, we have studied the impact of packet length when FER is relatively large. In that case, it can happen that long packets results in large transmission delay.

Chapter 5

TCP Flow Control Using Link Layer Information in Mobile Networks

5.1 Introduction

With the growth of mobile networks and the Internet, the data transmitting service, i.e., e-mail, web browsing, mobile computing, etc., over wireless networks becomes more attractive. Therefore, the next generation mobile system, called IMT-2000 (International Mobile Telecommunications-2000) [IMT], has been standardized by ITU (International Telecommunication Union) so as to provide faster data transmitting service than the current mobile system and world wide roaming capability. There is W-CDMA (Wideband-Code Division Multiple Access) technology prescribed by 3GPP (3rd Generation Partnership Project) [3GPa] for wireless networks as one of what IMT-2000 system employs. The current mobile system provides a transmission rate of 9.6 Kb/s, whereas W-CDMA system can provide the maximum of 384 Kb/s outdoors. It also uses strong error recovery technologies in order to minimize the damage of the utilization of the wireless channels due to bit error.

On the other hand, TCP (Transmission Control Protocol) [Ste94], which is a reliable end-to-end transport protocol in the Internet Protocol suite, is widely used in popular applications like Telnet, FTP, and HTTP in the current Internet. TCP has been tuned for wired networks where bit error rates are very low and packet losses occur mostly because of congestion in the networks. Therefore, TCP performs well over such networks by adapting its transmission rate to end-to-end delays and packet losses caused by congestion [JK88]. However, when a TCP sender transmits packets over wireless links characterized by high bit error rates, the performance of TCP can be drastically degraded. This is because TCP recognizes packet loss as a result of congestion even if it happens due to bit error over wireless links, and hence reduce its transmission rate [CLM99]. W-CDMA system thus employs FEC (Forward Error Correction) coding and ARQ (Automatic Repeat reQuest) mechanism as error recovery schemes in Layer 2 of wireless networks in order to reduce the damage due to bit error on TCP connections.

Nevertheless, ARQ cannot guarantee that packet arrives in sequence at the mobile stations, so that some waiting timer is used to keep sequence integrity of packet delivery. The larger timer allows multiple retransmissions so that it can attain a larger throughput, whereas it increases the packet queue length at the base station as well as RTO of TCP senders. In fact, the buffer size of the base station is finite, so that the buffer overflow can occur if the queue length becomes too long, and large RTO will keep TCP senders waiting for ACK (Acknowledgment) for long duration if packets really get lost in the buffer of routers on wired networks or the base station. Therefore, it is very likely that TCP cannot execute flow control effectively in wireless networks.

Our major goal of this study is to develop a TCP flow control achieving stable and good throughput performance in this context, and requiring no modification on TCP senders, which will be likely connected with wired networks. For this end, we focus a way of employing information on a data link, i.e., wireless links, at the mobile station for TCP flow control, and propose the receiver-based TCP flow control using the infor-

mation. The TCP receiver can keep the congestion window within a limited range in a way of sending ACK with advertised window size based on the information from Layer 2, instead of usual available buffer size.

A-TCP (Adaptive-TCP) [SJK01], which is a TCP-aware link-layer solution like Snoop protocol, has been proposed based upon similar concept, whereas it needs per-connection state of TCP accommodated at the base station because it uses the available buffer size at the base station instead of that of the mobile station for TCP flow control. However, the mechanism of TCP flow control proposed here does not require any additional schemes in the intermediate nodes including the base station except for a little modification to the mobile station. We will investigate the performance of proposed TCP in wireless networks by means of simulations and show that the receiver-based TCP flow control can moderate the performance degradation under the condition that FER (Frame Error Rate) on Layer 2 is high.

5.2 TCP over wireless environments

In this section, we will introduce the wireless environments and TCP flow control we will focus in this research. We explain W-CDMA system as one of what IMT-2000 system employs, and both of original TCP flow control and proposed one.

5.2.1 W-CDMA

In W-CDMA environment, RLC (Radio Link Control) protocol [3GPc] is standardized by 3GPP as Layer 2 protocol. The RLC protocol divides a SDU (Service Data Unit) corresponding to IP datagram received from the upper layer into several PDUs (Protocol Data Unit) of 42 bytes and transmits them as show in Fig. 5.1. It employs Selective-Repeat ARQ using Polling/Status Report for recovering lost PDUs. The four alternative schemes

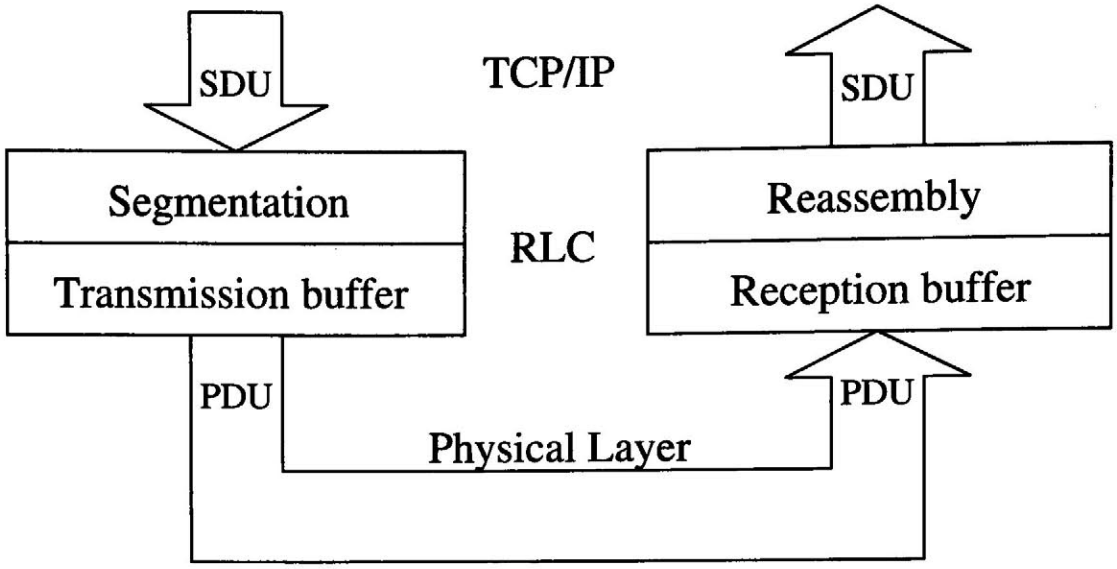


Figure 5.1: SDU/PDU in RLC protocol

of the SDU discard to prevent the delay required for recovering SDU from diverging to infinity are defined [3GPc]. They are the schemes of timer-based discard with explicit signaling, timer-based discard without explicit signaling, discard after maximum number of transmissions, and no discard. We have introduced the schemes of the SDU discard after maximum number of an initial transmission and retransmissions caused by bit error on Layer 2 into this research.

In addition, 3GPP specifies a way of preserving the order of packet delivery by keeping some PDUs waiting in Layer 2 of the mobile station if their preceding PDUs suffer bit error and thus are being transmitted [3GPb]. Furthermore, W-CDMA can provide a 384 Kb/s maximum transmission rate, whereas in fact the transmission rate may change frequently due to air conditions.

5.2.2 TCP flow control

A TCP sender executes flow control to utilize as much available bandwidth as possible. TCP has two windows for its flow control; one is the sender's congestion window (*cwnd*) and the other is the receiver's advertised window (*awnd*). The *cwnd* is increased or decreased according to the conditions of congestion in the networks. The *awnd* represents the capacity of receiver's buffer and is dynamically informed by the receiver. The sender's transmission rate is the smaller one of two windows. In general, it is determined by the *cwnd* for the reason that the receiver usually has the buffer of a larger capacity than network capacity, i.e., an available bandwidth. The *cwnd* is controlled dynamically to adapt to the current conditions of congestion in the networks on the basis of the spacing of the ACKs corresponded to the bandwidth of the bottleneck link. However, in the wireless networks, the spacing of the ACKs and the bandwidth of wireless link change frequently. Accordingly, there is the possibility that the *cwnd* is not controlled suitably, in other words, the sender-based TCP flow control is not executed effectively in the wireless networks.

Therefore, we will propose the receiver-based TCP flow control using some information on wireless link conditions which can be obtained from Layer 2 at the mobile station. The TCP receiver informs the sender of the current available bandwidth of a wireless link in a way of sending ACK with the *awnd* based on the information from Layer 2, instead of usual available capacity of its buffer. Consequently, the TCP receiver can keep a transmission rate which is the smaller one of the *cwnd* and the *awnd* within a limited range, i.e., an available bandwidth of the wireless link. This scheme can make use of the IPsec which encodes the payload of IP datagram due to keeping the end-to-end model of the TCP connection. However, the split connection schemes and the snoop schemes such as the Indirect-TCP (I-TCP) [BB95] and A-TCP [SJK01] cannot use it because they are in need of snooping into the header of TCP segments at the base station.

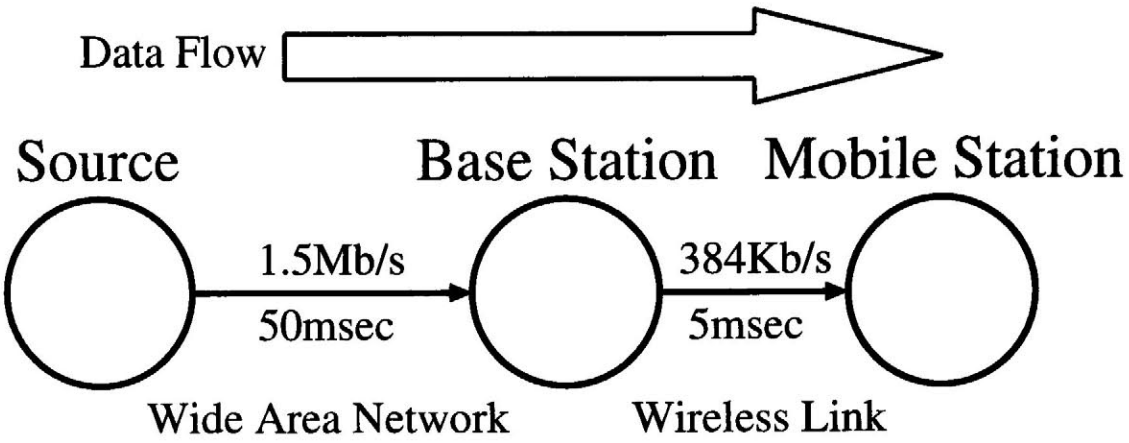


Figure 5.2: Simulation model

5.3 Simulation model

In this section, we describe the simulation model used here. We use the VINT Network Simulator NS Version 2 [NS] after we added some modifications for Layer 2 ARQ schemes over wireless links for our research.

In our simulation, one TCP source transmits packets from a wired link to a wireless link via base station as shown in Fig. 5.2. The wired link is 1.5 Mb/s in bandwidth and 50 msec in propagation delay like in WAN (Wide Area Network), and the wireless link is 384 Kb/s in bandwidth and 5 msec in propagation delay including the processing delay in base station and mobile station. We assume that the wireless link suffers a burst error caused by Rayleigh Fading [ASS98]. More specifically, users are walking with the mobile station, which resulting in the two-path fading channel with Doppler frequency of 5 Hz. A power control is also performed. The average error rates of Layer 2 considered here are 1.2, 2.6, and 5.9 percent in FER (Frame Error Rate) with FEC. We also assume that the ACK and NACK (Negative ACK) are error-free.

In this simulation, it is assumed that TCP traffic is used for greedy file transfer, which keeps the transmitting data of infinity at the sender. The proposed scheme can apply to

TCP variants used in the current Internet since it does not require any modifications to the existing flow control of the TCP variants. The TCP variants employed here are thus Tahoe, Reno, NewReno [FH99], and SACK [MMFR96]. Tahoe TCP is the oldest implementation and has the most simple algorithm. It uses a fast retransmit mechanism to recover lost packets. Reno TCP has more effective flow control by making use of a fast recovery mechanism. NewReno TCP has been proposed to moderate Reno's problem that the throughput performance is degraded due to multiple packet losses in one window, and it is used mostly in the current Internet. SACK TCP employs a more efficient retransmission mechanism, i.e., Selective-Repeat ARQ different from Go-Back-N ARQ used by other TCP variants.

The TCP segment size is set to 1500 bytes, and the Layer 2 PDU size is set to 42 bytes; the header is 2 bytes and the payload is 40 bytes. In Layer 2, the Selective-Repeat ARQ is employed, which the maximum number of allowable retransmissions is set to five. The base station keeps some PDUs waiting for the Waiting Timeout value (WT) to preserve sequence integrity of packet delivery, and it changes from 0 msec to 1000 msec. The base station is equipped with a buffer whose capacity is 15 packets, and packet loss thus occurs there due to congestion. We carry out simulation experiments for 80 seconds, and discuss the average throughput performance of TCP.

5.4 Simulation results

In this section, we will show the characteristics of TCP performance in wireless networks by means of simulation results. We will first discuss the performance of proposed TCP in wireless networks where a link bandwidth is constantly 384 Kb/s. Next, we will examine whether the proposed TCP is effective from viewpoints of the throughput performance when the bandwidth of a wireless link changes dynamically.

5.4.1 The performance of proposed TCP

First, we consider an ideal *awnd* value. To use effectively an available bandwidth, TCP needs to set up the *awnd* larger than the capacity of the bottleneck link, i.e., the bandwidth-delay product of the wireless link. On Layer 2, the retransmission of the lost PDUs is executed due to bit error. The delay thus equals the worst time T required to transmit one TCP segment to the mobile station over a wireless link. It can be expressed as Eq. (5.1) if the transmission delay time of the ACK (NACK) on Layer 2 is ignored.

$$T = \left(\frac{\text{TCP segment size}}{\text{link bandwidth}} + 2 * \text{propagation delay} \right) * \text{the maximum number of allowable retransmissions.} \quad (5.1)$$

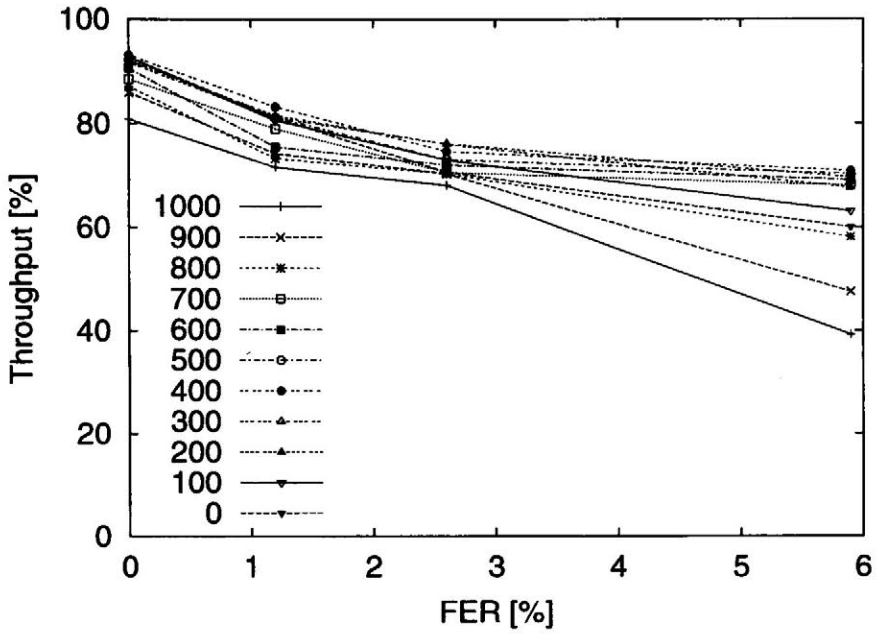
Using this T, the ideal *awnd* value can be given by

$$\text{ideal } awnd = \frac{\text{link bandwidth} * T}{\text{TCP segment size}}. \quad (5.2)$$

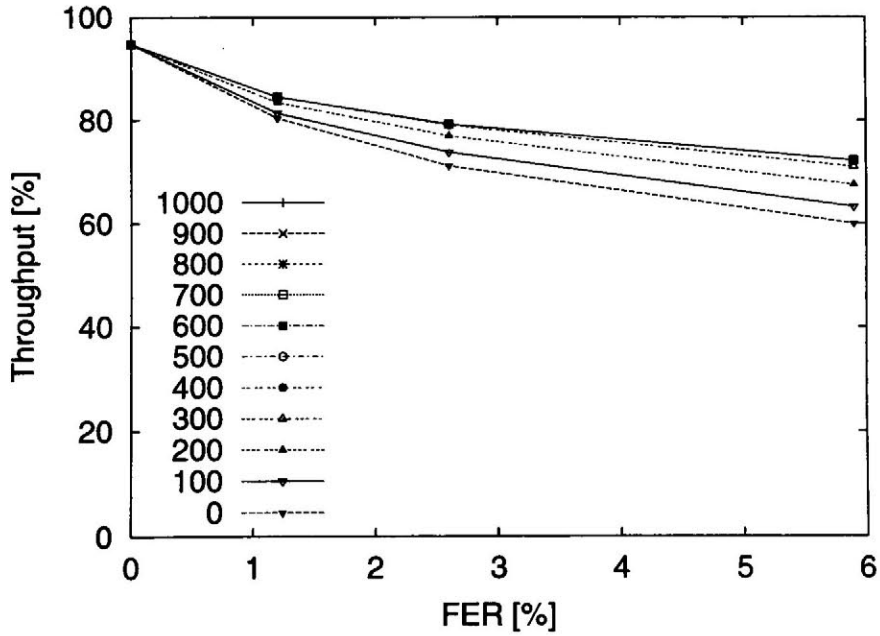
If the link bandwidth is 384 Kb/s and the TCP segment size is 1500 bytes, the ideal *awnd* value is 9.6 packets.

Now, we examine the average throughput performance of TCP when WT on Layer 2 varies from 0 msec to 1000 msec. Fig. 5.3 (a) and (b) show the throughput characteristics of original and proposed (*awnd* = 10 packets) TCP respectively. We consider the case that the bandwidth of the wireless link is constantly 384 Kb/s and NewReno TCP is employed. The throughput of the y-axis is normalized to the bandwidth of the wireless link. From these figures, the larger FER decreases the throughput performance of TCP. The throughput of TCP is out of proportion to the WT value, and thus it is necessary to set up WT value to adapt the FER to achieve good performance when the original TCP is employed. However, the proposed TCP can attain excellent throughput performance with a larger WT value.

To investigate the optimum value of WT that maximizes the throughput of TCP, the throughput performance of TCP is shown in Fig. 5.4 when FER is 2.6 percent. The TCP



(a) original TCP



(b) proposed TCP

Figure 5.3: The effect of FER

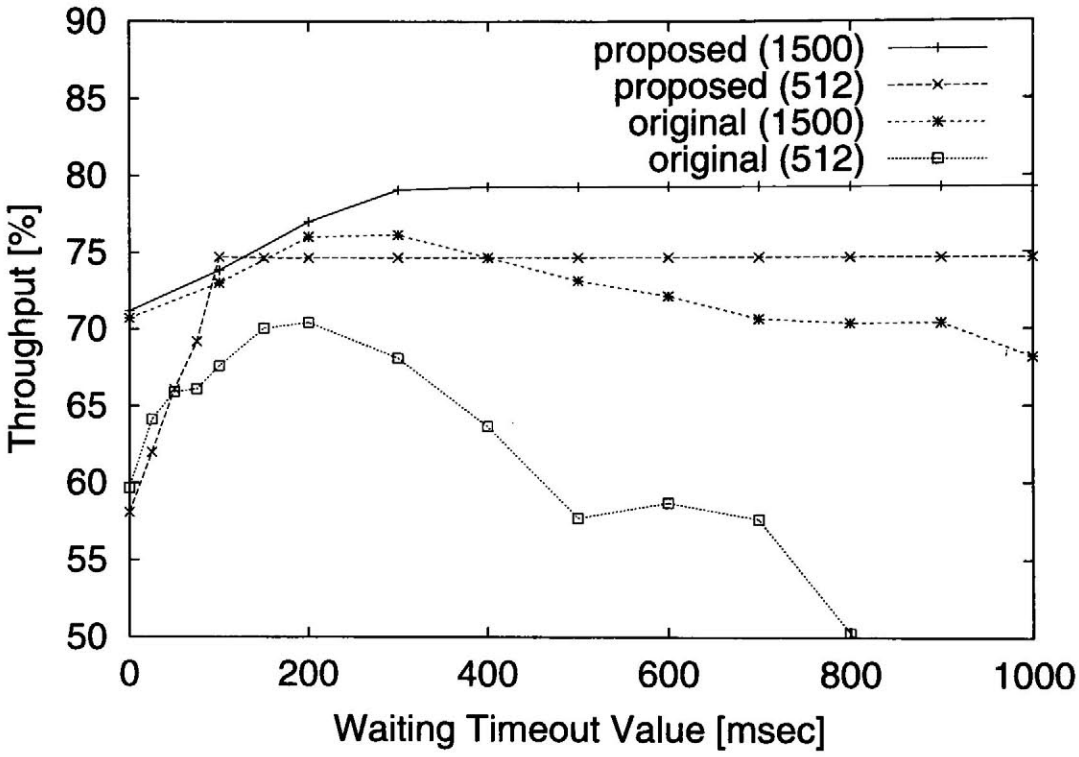


Figure 5.4: The effect of L.2 Waiting Timeout value: FER=2.6%

segment size is set to 512 and 1500 bytes. From this figure, the optimum values of *WT* are different for each TCP segment size; it is 100 msec for TCP segment size of 512 bytes and 300 msec for that of 1500 bytes. These values equal approximately the worst time required to transmit one TCP segment to the mobile station over a wireless link, which is given by Eq. (5.1). Using Eq. (5.1), the optimum values of *WT* are approximately 210 msec for the TCP segment size of 1500 bytes and 100 msec for that of 512 bytes. A larger *WT* than an optimum value can cause the throughput degradation of TCP when original TCP is employed. This is because the mobile station wastes a long time detecting the occurrence of packet discards due to the overflow of the base station's buffer as *WT* increases. This leads the timeout in TCP flow control. From Fig. 5.4, it can be seen that it is necessary to use the *WT* of the above value to improve the throughput performance of the original TCP. However, this value usually changes due to air conditions. On the other hand, the proposed TCP does not depend on the *WT* value and achieves better throughput performance than that of the original TCP in a wide range of *WT* values. In the following, we will deal with the case that the *WT* values are suitably set to 300 msec for the link bandwidth of 384 Kb/s and 1000 msec for that of 64 Kb/s.

Next, Fig. 5.5 shows the average throughput performance of TCP for different TCP variants employed. We can see that the proposed mechanism applied to every TCP variant improves the throughput performance and there is no difference of throughput performance among any TCP algorithms. This is because it can restrain the buffer overflow of the base station by preventing the transmission rate from overgrowing to the available bandwidth of wireless link using *awnd*. In the following, we will focus on NewReno TCP, which are mainly used on the current Internet.

Fig. 5.6 shows the average throughput performance of TCP as a function of *awnd* when the *WT* value is set to the optimum value for the link bandwidth of 64 Kb/s and 384 Kb/s. For reference, it also shows the results when the *WT* value is set to not optimum value (i.e. *WT* = 1000 msec). The TCP segment size is set to 1500 bytes. From

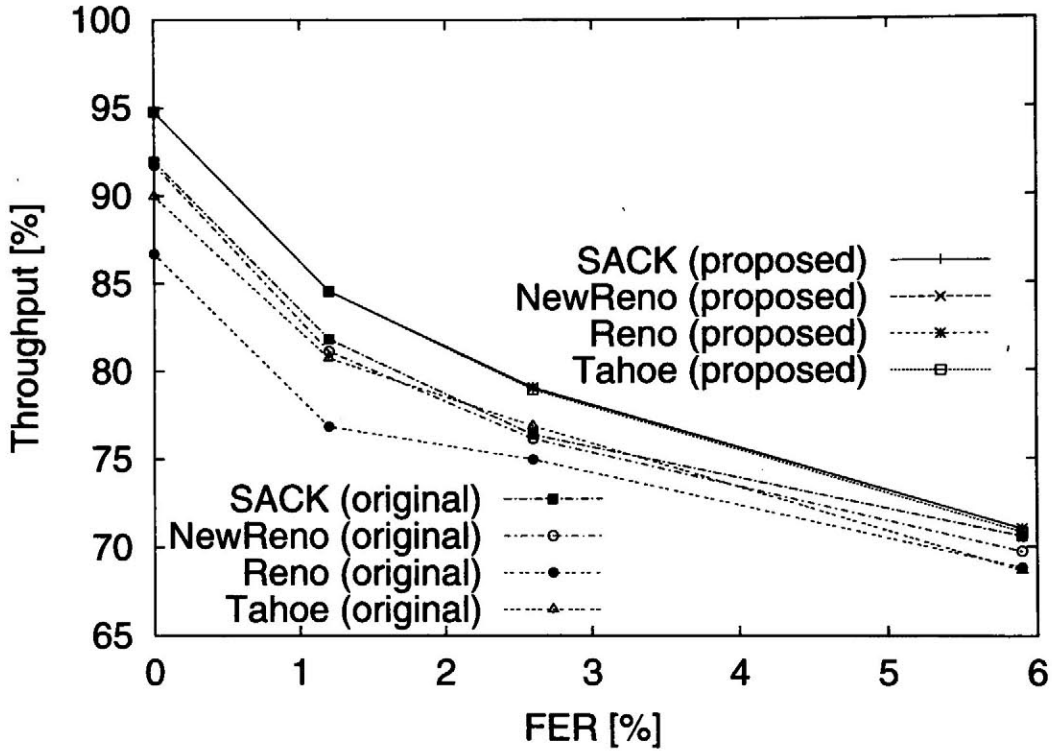


Figure 5.5: The effect of TCP algorithms

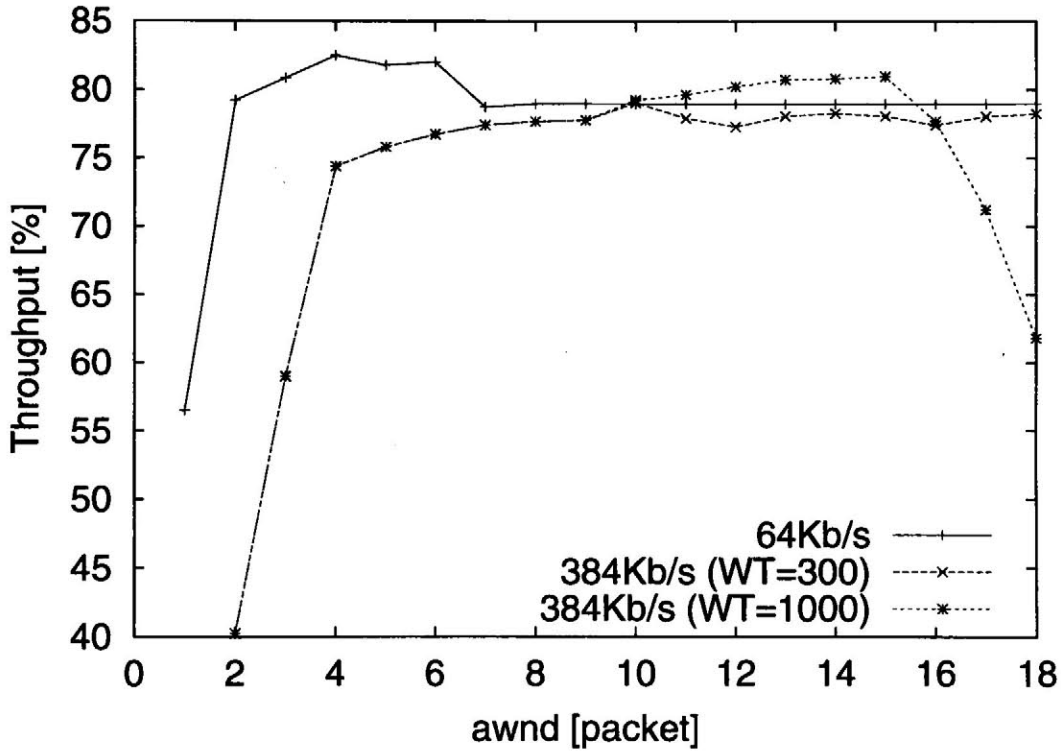


Figure 5.6: The effect of *awnd*

Table 5.1: The number of TCP retransmissions

awnd	64Kb/s		384Kb/s	
[packet]	Timeout	Dup. ACK	Timeout	Dup. ACK
3	0	0	0	0
4	**0**	**0**	0	0
5	1	2	0	0
6	1	4	0	0
7	1	12	0	0
8	1	13	0	0
9	1	13	0	0
10	1	13	**0**	**0**
11	1	13	0	19
12	1	13	0	27
13	1	13	0	27

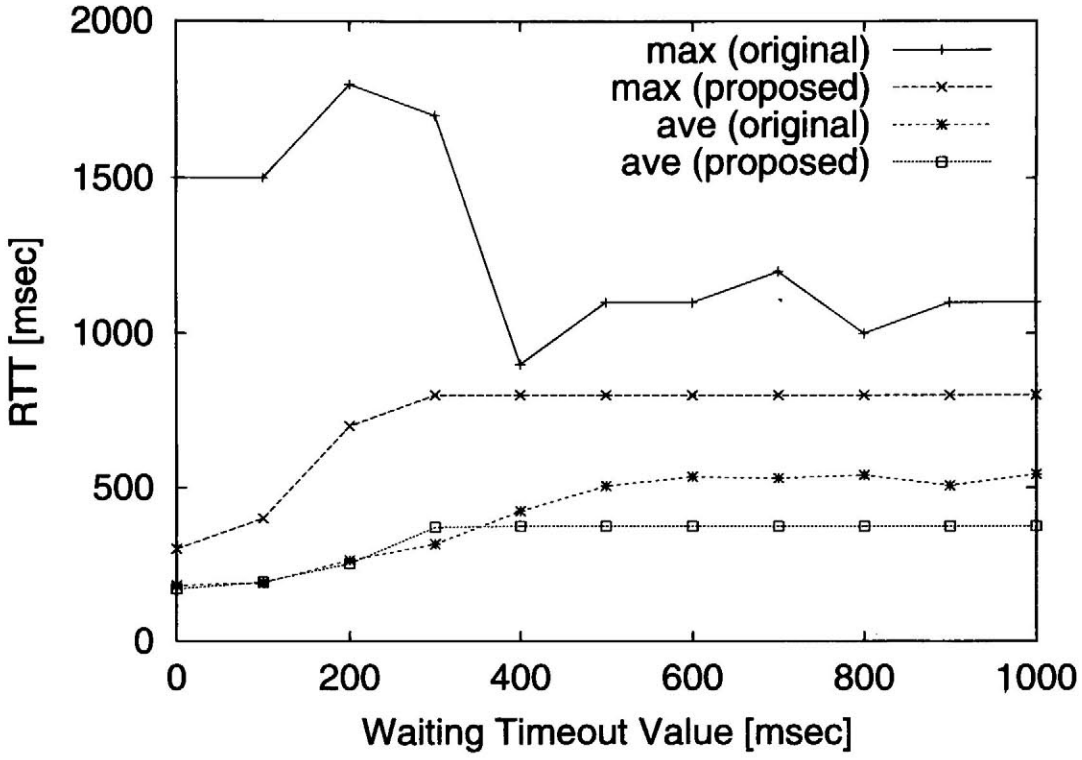


Figure 5.7: The effect of RTT

this figure, the optimum values of *awnd* are 4 packets for the link bandwidth of 64 Kb/s and 10 packets for that of 384 Kb/s (WT = 300 msec). Table 5.1 shows the number of TCP retransmission occurrences by timeout and duplicate ACKs in that case. From the figure and the table, it can be seen that the optimum value of *awnd* is determined by the maximum value that the retransmission of TCP does not happen, i.e., the buffer overflow does not occur at the base station. If the *awnd* is set to larger than the optimum value, the throughput performance of TCP degrades due to TCP retransmission occurrence. Furthermore, for a wide range of the *awnd*, the throughput performance of TCP does not change. It is helpful to easy operation.

Lastly, Fig. 5.7 shows the average and maximum value of the RTT (Round Trip Time) observed by TCP. From this figure, the proposed TCP can keep both of the average and

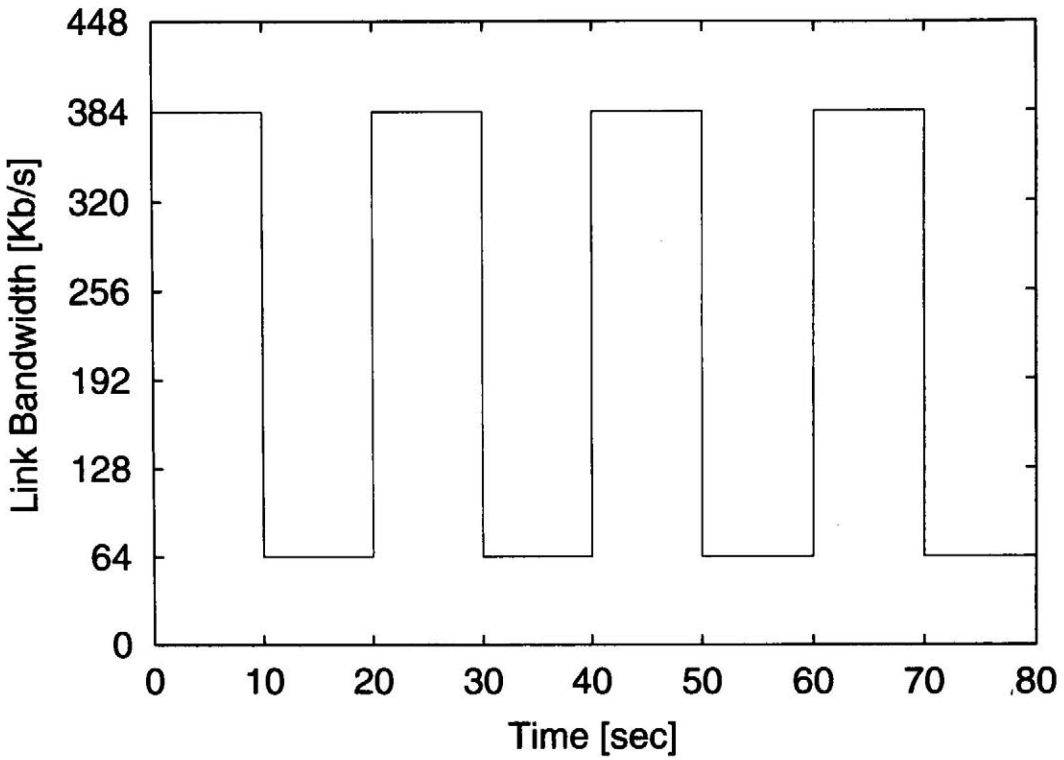


Figure 5.8: The situation of the link bandwidth fluctuated

maximum values smaller than that of the original TCP. Consequently, the proposed TCP can decrease the RTO (Retransmission Timeout) of TCP calculated based on the RTT.

5.4.2 The adaptability to the fluctuation of the link bandwidth

Next, in this subsection, we examine whether proposed TCP effectively adapts to the environment where the bandwidth of the wireless link changes dynamically. We consider the case that the bandwidth of the wireless link fluctuates between 384 and 64 Kb/s at intervals of 10 seconds as shown in Fig. 5.8.

First, Fig. 5.9 shows the average throughput performance of TCP when WT value of Layer 2 varies from 0 msec to 1000 msec. NewReno TCP is employed as the previous

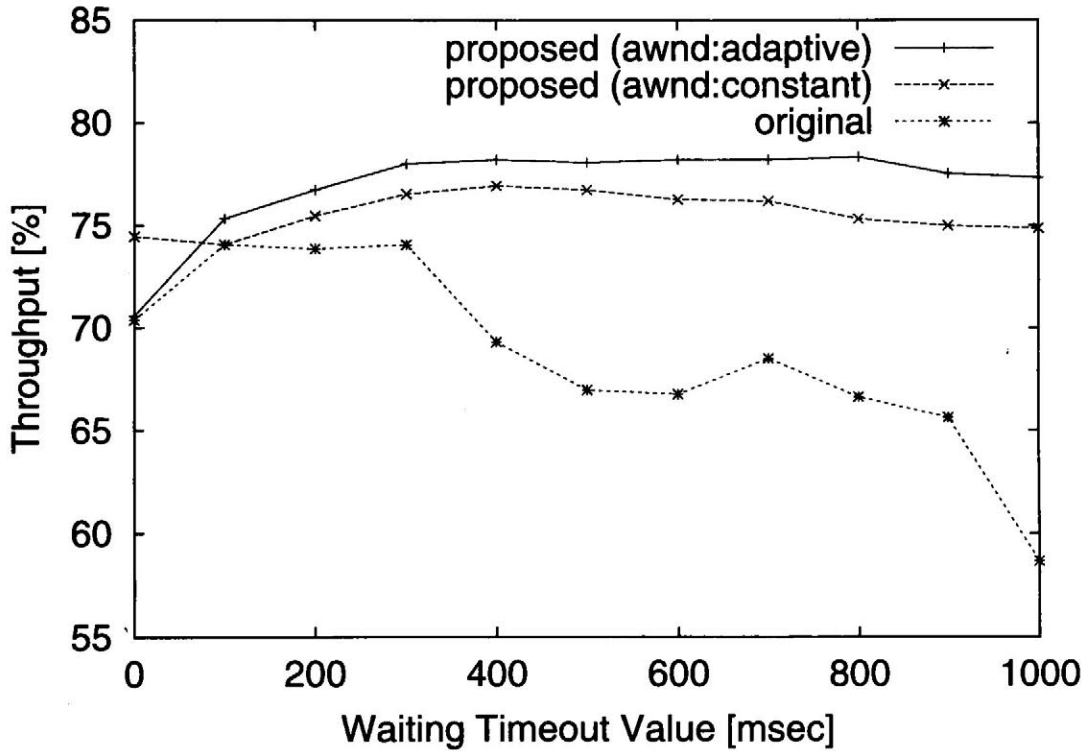


Figure 5.9: The effect of L.2 Waiting Timeout value: FER=2.6%

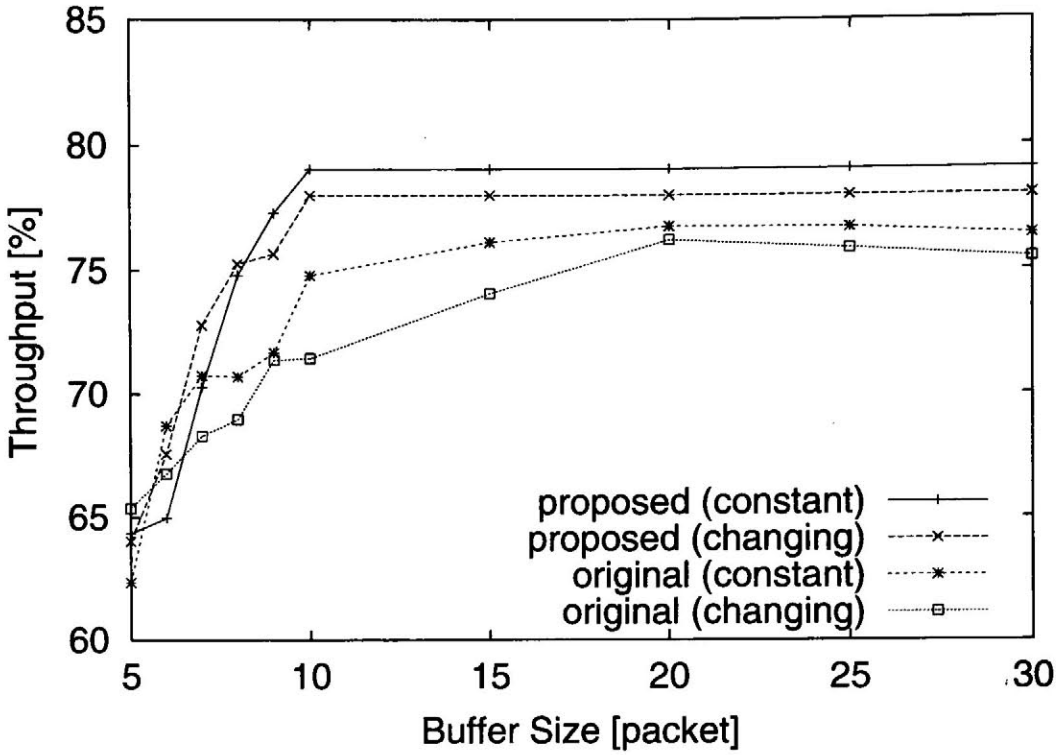


Figure 5.10: The effect of the capacity of the buffer at the base station: FER=2.6%

subsection, and FER considered is 2.6 percent. The proposed TCP has two cases in this figure; one is the case that the *awnd* value is kept the optimum value (10 packets) for the bandwidth of 384 Kb/s even if the link bandwidth lowers to 64 Kb/s, which is shown as *awnd:constant*. The other is the case that the *awnd* value adaptively changes according to the link bandwidth (10 packets for 384 Kb/s, 4 packets for 64 Kb/s), which is shown as *awnd:adaptive*. From this figure, the proposed TCP can attain the good throughput performance even if the link bandwidth changes dynamically. Furthermore, adapting *awnd* to the link bandwidth fluctuated can improve the throughput performance of TCP.

Lastly, we examine the effect of the buffer size at the base station on the throughput performance of TCP when FER is 2.6 percent as shown in Fig. 5.10. In this figure, *constant* means the case that the link bandwidth is constant and *changing* means the case that

one is changing as shown in Fig. 5.8. The proposed TCP employed here is *adaptive* which adapts *awnd* to the link bandwidth fluctuated. From this figure, the proposed TCP can attain the good throughput performance if the base station equips the buffer whose capacity is the *awnd* value (= 10 packets) at least. On the other hand, the original TCP needs the buffer of a larger capacity at the base station to improve the throughput performance.

5.5 Conclusions

In our research, we have proposed receiver-based TCP flow control in which the receiver sends back ACK with *awnd* based on the information of wireless link conditions which can be obtained from Layer 2 at the mobile station, and investigated the effectiveness of the proposed TCP over wireless networks of IMT-2000 by means of simulation. First, from our simulations, the throughput performance of the original TCP extremely depends on the Waiting Timeout value (WT) which is required by Selective-Repeat ARQ at the Layer 2 to keep sequence integrity of the packet delivery. However, the proposed TCP does not depend on the WT value and can achieve better throughput performance than that of the original TCP. Furthermore, the proposed TCP can decrease the average and maximum values of RTT observed by TCP and cause that the difference of the TCP algorithms does not affect the throughput performance. Next, the proposed TCP can attain the excellent throughput performance even if the bandwidth of the wireless link is changing drastically. Furthermore, adapting *awnd* value to the link bandwidth fluctuated can improve the throughput performance of TCP. Lastly, the proposed TCP can attain the good throughput performance if the base station equips the buffer whose capacity is the *awnd* value at least, that is it needs smaller capacity of the buffer at the base station than the original TCP.

Chapter 6

Concluding Remarks

This dissertation has examined transport protocols in the emerging network environment; of "the multimedia Internet" and "the mobile Internet". In particular, TCP (Transmission Control Protocol) has been evaluated, focusing on its overall performance in such networks, by means of simulation experiments.

In Chapter 2, two main protocols used in transport layer, TCP and UDP were described. TCP flow control mechanisms, which use a sliding window algorithm with slow start and congestion avoidance algorithms, and the historical process of the development of TCP variants were explained. The Tahoe TCP, Reno TCP, NewReno TCP, and SACK TCP were introduced as TCP variants used in the current Internet that have improved flow control mechanisms. In particular, the Fast Retransmit and Fast Recovery algorithms used for efficient use of network bandwidth when network congestion occurs were discussed. The issues that TCP encounters when used over wireless networks because it was designed to operate over highly reliable links and with stationary hosts were identified. Schemes for improving TCP over wireless networks by preventing the throughput degradation due to bit errors in TCP connections were proposed. They are classified into split connection schemes, proxy schemes, and end-to-end schemes.

In Chapter 3, the performances of TCP variants in the multimedia Internet were in-

vestigated and compared. In such QoS networks, CBR traffic can have higher priority than TCP traffic, so that TCP will use the unused bandwidth left unused by high priority traffic. Therefore, the amount of bandwidth available for TCP traffic can change time to time. Simulation results have shown that SACK TCP outperforms the other TCP variants in QoS networks as well as in the current Internet. Investigation of the effect of the fluctuation in the interarrival times of CBR packets on the performance of TCP revealed that the total throughput of TCP connections is not affected by it, but it does reduce the fairness of throughput among those connections. Nevertheless, SACK TCP is very robust against that problem.

In Chapter 4, the performance of TCP in the mobile Internet was also examined. In wireless networks, packet loss can be caused by the burst errors that result from Rayleigh fading. Thus, the performance of TCP can be severely affected by the high bit error rate that is characteristic of wireless links, because TCP is tuned to perform well in wired networks where the error rates are very low and packet loss occurs mostly due to congestion. To avoid that degradation, 3GPP employs FEC coding and ARQ mechanisms as error recovery schemes in Layer 2 of wireless networks. This can cause packets to arrive at their destination out of order because of the Selective-Repeat ARQ algorithm employed in W-CDMA. Hence, the effect of the ARQ mechanisms of Layer 2 and an effective way of managing packets for sequence integrity to improve the performance of TCP over such networks were studied. Simulation results showed that, with Layer 2 ARQ, TCP can achieve better throughput performance with packets of larger size. However, Layer 2 ARQ is less effective in improving the throughput of TCP on links of larger bandwidth. Nevertheless, the reordering of packets for maintaining sequence integrity of packet arrivals at the mobile station improves the throughput performance of TCP, even if the link bandwidth is large.

In Chapter 5, A receiver-based TCP flow control scheme for the mobile Internet was proposed. In actual environments, the link bandwidth that is available to TCP flows can

change in response to changing atmospheric conditions for efficient use of wireless links. TCP thus has to adapt its transmission rate according to the changing available bandwidth. Therefore, a receiver-based effective TCP flow control scheme that does not invoke any modification of TCP senders was proposed. Under this scheme, the TCP receiver informs the sender of the current available bandwidth on the wireless link. Consequently, the TCP receiver can maintain the transmission rate within the available bandwidth of the wireless link. Simulation results showed that the proposed TCP can achieve throughput performance that is better than that of the original TCP without dependence on Layer 2 configuration. Furthermore, the proposed TCP can attain excellent throughput performance, even if the bandwidth of the wireless link changes drastically.

In this dissertation, I have considered some issues on transport protocol in the emerging network environment. However, the following issues remain for future work. Although I have presented the case of just one TCP flow with a simple network topology, in actual networks, there are more flows in complex topologies. It is therefore necessary to study the characteristics of multiple TCP flows in a multihop network topology and schemes for providing QoS assurance over wireless networks in future work.

Bibliography

[3GPa] 3rd Generation Partnership Project, *http://www.3gpp.org/.*

[3GPb] 3GPP TS 25.301 V4.2.0 Radio Interface Protocol Architecture, December 2001.

[3GPc] 3GPP TS 25.322 V4.2.0 RLC protocol specification, September 2001.

[APS99] M. Allman, V. Paxson, and W. Stevens, "TCP congestion control", *RFC2581*, April 1999.

[ASS98] F. Adachi, M. Sawahashi, and H. Suda, "Wideband DS-CDMS for next-generation mobile communications systems", *IEEE Communication Magazine*, Vol. 36, No. 9, pp. 56–69, September 1998.

[BB95] A. Bakre and B. R. Badrinath, "I-TCP: Indirect TCP for mobile hosts", In *Proceedings of the 15th International Conference on Distributed Computing Systems (ICDCS)*, May 1995.

[BKVP97] B. S. Bakshi, P. Krishna, N. H. Vaidya, and D. K. Pradhan, "Improving performance of TCP over wireless networks", In *Proceedings of the 17th International Conference on Distributed Computing Systems (ICDCS'97)*, Baltimore, May 1997.

[BPSK96] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz, "A comparison of mechanisms for improving TCP performance over wireless links", In *Proceedings of ACM SIGCOMM'96*, pp. 256–269, August 1996.

[BSAK95] H. Balakrishnan, S. Seshan, E. Amir, and R. H. Katz, "Improving TCP/IP performance over wireless networks", In *Proceedings of 1st ACM International Conference on Mobile Computing and Networking (Mobicom)*, November 1995.

[CB00] A. Chockalingam and G. Bao, "Performance of TCP/RLP protocol stack on correlated fading DS-CDMA wireless links", *IEEE Transactions on Vehicular Technology*, Vol. 49, No. 1, pp. 28–33, January 2000.

[CLM99] H. Chaskar, T. V. Lakshman, and U. Madhow, "TCP over wireless with link level error control: Analysis and design methodology", *IEEE Transactions on Networking*, Vol. 7, No. 5, pp. 605–615, October 1999.

[CZ99] A. Chockalignam and M. Zori, "Wireless TCP performance with link layer FEC/ARQ", In *Proceedings of IEEE ICC'99*, Vancouver, June 1999.

[CZR98] A. Chockalingam, M. Zorzi, and R. R. Rao, "Performance of TCP on wireless fading links with memory", In *Proceedings of IEEE ICC'98*, Vol. 2, pp. 595–600, Atlanta, June 1998.

[FF96] K. Fall and S. Floyd, "Simulation-based comparisons of Tahoe, Reno, and SACK TCP", *ACM Computer Communication Review*, Vol. 26, No. 3, pp. 5–21, July 1996.

[FH99] S. Floyd and T. Henderson, "The NewReno modification to TCP's fast recovery algorithm", *RFC2582*, April 1999.

[FJ95] S. Floyd and V. Jacobson, "Link-sharing and resource management models for packet networks", *IEEE/ACM Transactions on Networking*, Vol. 3, No. 4, pp. 365–386, August 1995.

[HSSO98] Y. Hori, H. Sawashima, H. Sunahara, and Y. Oie, "Performance evaluation of UDP traffic affected by TCP flows", *IEICE Transactions on Communications*, Vol. E81-B, No. 8, pp. 1616–1623, August 1998.

[IMT] IMT–2000, *http://www.itu.int/imt/*.

[Jac90] V. Jacobson, "Modified TCP congestion avoidance algorithm", end2end interest mailing list, April 1990.

[JBB92] V. Jacobson, R. Braden, and D. Borman, "TCP extensions for high performance", *RFC1323*, May 1992.

[JK88] V. Jacobson and M. J. Karels, "Congestion avoidance and control", In *Proceedings of ACM SIGCOMM'88*, pp. 314–329, August 1988.

[KHO01] H. Koga, Y. Hori, and Y. Oie, "Performance comparison of TCP implementations in QoS provisioning networks", *IEICE Transactions on Communications*, Vol. E84-B, No. 6, pp. 1473–1479, June 2001.

[KIHO] H. Koga, T. Ikenaga, Y. Hori, and Y. Oie, "Out-of-sequence in packet arrivals due to layer 2 ARQ and its impact on TCP performance in W–CDMA networks,", to appear in Proc. of SAINT-2003 (The 2003 International Symposium on Applications and the Internet).

[KKO02] H. Koga, K. Kawahara, and Y. Oie, "TCP flow control using link layer information in mobile networks", In *Proccedings of SPIE Conference of Internet Performance and Control of Network Systems III*, Vol. 4865, pp. 305–315, July 2002.

[KP87] P. Karn and C. Partridge, "Improving round-trip time estimates in reliable transport protocols", *ACM Computer Communication Review*, Vol. 17, No. 5, pp. 2–7, August 1987.

[MMFR96] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP selective acknowledgement options", *RFC2018*, October 1996.

[MV97] M. Mehta and N. Vaidya, "Delayed duplicate-acknowledgements: A proposal to improve performance of TCP on wireless links", Technical report, Texas A&M University, December 1997.

[NS] VINT Project, Network Simulator ns–2, *http://www.isi.edu/nsnam/ns/*.

[Pos80] J. Postel, "User datagram protocol", *RFC768*, August 1980.

[SF98] N. K. G. Samaraweera and G. Fairhurst, "Reinforcement of TCP error recovery for wireless communication", *ACM Computer Communication Review*, Vol. 28, No. 2, pp. 30–38, April 1998.

[SJK01] S. Seok, S. Joo, and C. Kang, "A-TCP: A mechanism for improving TCP performance in wireless environments", In *Proceedings of IEEE Broadband Wireless Summit (N+I 2001)*, May 2001.

[Ste94] W. R. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*, Addison-Wesley, Reading, Massachusetts, 1994.

[WS95] G. R. Wright and W. R. Stevens, *TCP/IP Illustrated, Volume 2: The Implementation*, Addison-Wesley, 1995.