Studies on
Congestion Control Mechanisms Realizing
Various End-to-End Comunication Qualities

Kazumi Igarashi

Preface

The Internet has become an important infrastructure and continues to expand. With the ubiquitousness of the Internet in our daily lives, the amount of data, the number of flows, and the types of applications that coexist on the Internet have been increasing. Originally, people used the Internet to realize connectivity between senders and receivers. Currently, however, connectivity that meets a diverse range of requirements for various applications is desired. In a well-provisioned network in which the number of users is limited, it is easy to realize connectivity that meets various requirements with no dedicated controls. On the Internet, however, flows that have various requirements coexist and limited network resources are shared among those flows. Thus, in order to realize connectivity under these environments, congestion controls are important in the network. The present research, therefore, focuses on the congestion control mechanisms in order to realize end-to-end communication qualities that are adequate and suitable for the diversity of the network. Three diversities in the network are considered, i.e., network environments, application types, and the quality of services required by users. In addition, the problems in the current congestion control mechanisms are clarified in order to achieve various levels of end-to-end quality of service in the network and schemes are proposed to solve the problems. The congestion controls in the network can be classified into two categories from an architectural viewpoint: controls conducted between end hosts and controls conducted at all nodes along the path, including intermediate nodes in addition to end hosts. In the following discussion, these two categories of congestion control, working between end-to-end hosts and working at all nodes along the

path, are described.

Chapter 1 describes the background of the present research and outlines the present approach to target issues.

In Chapter 2, the congestion control conducted between end hosts is introduced. Transmission Control Protocol (TCP) is a representative protocol working between end hosts and has been adopted as a transport protocol in the network, which can provide a highly reliable networking environment for non-real time application flows. However, it is well known that TCP cannot achieve efficient data transfer in fast long-distance networks. Therefore, various high-speed transport protocols have been proposed to solve this problem, and these protocols will also be introduced in Chapter 2.

In Chapter 3, the congestion control conducted at all nodes along the path is introduced.

In Chapter 4, the basic characteristics of a number of existing high-speed transport protocols are presented, which are obtained in testbed network. In this chapter, I primarily observe the throughput characteristics of a single high-speed transport protocol and discuss its efficiency and fairness for a Standard TCP flow.

In Chapter 5, a number of experimental results are discussed for various scenarios considering the future high-speed Internet. High-speed transport protocols were originally developed for realizing efficient data transfer in fast long-distance networks. Therefore, in the case of coexisting high-speed transport protocol flows and standard TCP flows, the performance of the standard TCP flow is affected by high-speed transport protocol flows. On the other hand, on the future Internet, the end-to-end network will be faster. Under these circumstances, I believe that users may be interested in transferring their data using high-speed transport protocol instead of the current Standard TCP. Therefore, in Chapter 5, an environment in which high-speed transport protocols are adopted to transfer data by users, and experimental scenarios are considered.

In Chapters 6 and 7, I describe the congestion control mechanisms in which intermediate nodes work in conjunction with end hosts. In Chapter 6, I evaluated the performance

of end-to-end non-real time flows that pass through multiple DiffServ domains. Research on the quality of service of end-to-end flows achieved by DiffServ technologies focuses primarily on a single DiffServ domain. However, the actual network environment is a network of networks, in which multiple DiffServ domains are connected. Therefore, the end-to-end throughput characteristics of a minimum bandwidth guarantee service flow (AF (Assured Forwarding) service flow) that passes through multiple DiffServ domains in the DiffServ framework are investigated. In the AF service, the packets are marked according to service class in their headers based on the measurement at the ingress edge routers, and are then forwarded to the intermediate nodes. At the border router to another DiffServ domain, the packet arrival rate is measured again and the service classes are re-marked if necessary. I investigate the impact of packet remarking that occurs at edge router on the end-to-end throughput characteristics of AF flow. In Chapter 7, early packet discarding schemes are proposed in order to improve the delay characteristics of real-time application flows. Some real-time applications set limits for acceptable network delay. For example, VoIP defines service classes based on the end-to-end packet delay limit. In these applications, packets delayed longer than an acceptable limit are invalidated by their applications when they reach their destinations, even though they have successfully arrived at the receiver. These packets are considered to be useless by the applications and thus impose an excess load on the network. Therefore, an early packet discarding scheme is proposed as a kind of active queue management scheme, in which packets that do not contribute to the quality of real-time applications are discarded in advance at intermediate nodes. I evaluate the effectiveness of the proposed schemes via network simulation in Chapter 7.

Finally, concluding remarks are presented in Chapter 8.

Mar. 2007

Kazumi Igarashi

Acknowledgements

First of all, I wish to express my gratitude to Professor Yuji Oie, my adviser, for his valuable comments, time and help completing the research. His steady support has greatly helped my study. I would also like to express my sincere appreciation to Professor Masato Tsuru. His constant encouragement, guidance through this research, invaluable discussions and advice have greatly helped in accomplishing the research. I thank them for their careful reading of all papers on the research.

I would also like to express my gratitude to Mr. Tadanori Hongo of Toho Electrical Construction Company for his kindly supports.

I would like to express my gratitude to Associate Professor Kenji Kawahara, Associate Professor Takeshi Ikenaga at Kyushu Institute of Technology and Associate Professor Yoshiaki Hori of Kyushu University for their valuable comments, time, and help in completing the research. Their steady support has greatly helped my study.

I would likewise thank my colleagues in NICT Kyushu Research Center, for their detailed, valuable instructions, fellowship and underpinning. I particularly thank Mr. Katsushi Kouyama and Dr. Hiroyuki Koga for their expert suggestions and helpful comments.

And, I extend thanks to Ms. Kazuko Tanaka and Ms. Mika Tsuji for their constant supports.

Finally, my greatest appreciation goes to my family, Kazuyoshi and Yukiyo Kumazoe, Hiroshi, Keiko and Masahiro Igarashi for their understandings and supports for me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

The Internet is a well-established form of infrastructure and has come to be used by not only industry but also as local infrastructure. Initially, the Internet was developed to provide best-effort service, that is, the Internet was designed to guarantee network connectivity. Currently, however, Internet service is required to provide best-effort connectivity while satisfying various diversities observed in the network. There are three elements of diversity on the network: the network environment, the user application type, and the level of quality of service required by users.

The network environment is becoming increasingly heterogeneous, for example, rapid progress is being made in the status of coexisting wired and wireless environments. In addition, several types of application flow coexist in the network, each of which requests different end-to-end service classes. For example, some applications give top priority to reliability and others applications assign the greatest importance to the real-time property. In the future Internet, the number of different application flow types will increase. Thus, the diversity of the requirements of the service classes will be greater. In addition, different diversities observed in the network are the level of service classes provided for users. The range of services

provided to the users in the network will become widespread with the rapid progress of network technology, for example, including the minimum bandwidth guaranteed service based on the contract or the priority service, such as a leased line. To provide connectivity while considering variousness observed in the network, the congestion control algorithm plays an important role in the network. Thus, in the present thesis, various congestion controls that consider the variousness of network elements were investigated.

The congestion control is conducted to reduce the degree of congestion in the network. For example, in TCP congestion mechanisms, the sender resends packets that are detected as discarded in the network, by reducing the original sending rate of the sender by half. TCP congestion control is conducted between end hosts. Therefore, if the sending rate is not reduced based on feedback information from the network, the congestion status is not improved, and as a result, most of the packets on the path become invalid. Congestion controls are designed to resolve the causes of network congestion and deterioration in communication quality in order to make efficient use of network resources.

In the network, the various congestion controls are used to achieve different goals. Table 1.1 lists the congestion controls described in the present thesis, including the purpose for the inclusion of each congestion control and the location at which the congestion control works.

Table 1.1: Targeted congestion control

#	Research theme	Goal	Place of implementation
1	Performance evaluation for high-speed transport protocols over a fast-long distance network	Variousness of network environment	End-to-end
2	Evaluation of minimum bandwidth guaranteed service in multiple DiffServ domains	Variousness of end-to-end service	End-to-end & intermediate node
3	Evaluation of proposed adaptive early packet discarding scheme	Variousness of application flows	End-to-end & intermediate node

As shown in Table 1.1, from the viewpoint of the framework of the control, the congestion controls are classified into the following two categories: those conducted only between end-to-end hosts and those in which the intermediate node works in conjunction with the end equipments. The congestion control conducted between end hosts has scalability compared to the schemes working with the intermediate nodes. However, the latter case has possibility

of conducting precise controls depending on network status.

In sections 1.2 and 1.3, the problems of the congestion controls adopted in the current network are described. In section 1.2, the congestion control conducted between end hosts is targeted, and in section 1.3, the controls conducted at an intermediate node are described. In section 1.4, an outline of the proposed mechanisms with which to address these issues is presented.

1.2 Problems of congestion control conducted between end hosts

The representative congestion control mechanism conducted between end hosts in the network is that in Standard TCP protocol. Many non-real-time applications on the Internet, including the World Wide Web and e-mail, adopt Standard TCP as their transport protocol. Standard TCP executes the error control, the flow control and the congestion control to realize high-reliability data transfer.

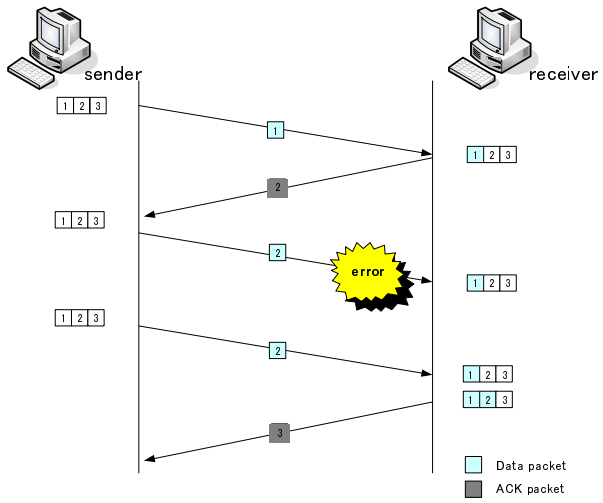


Figure 1.1: Error Control

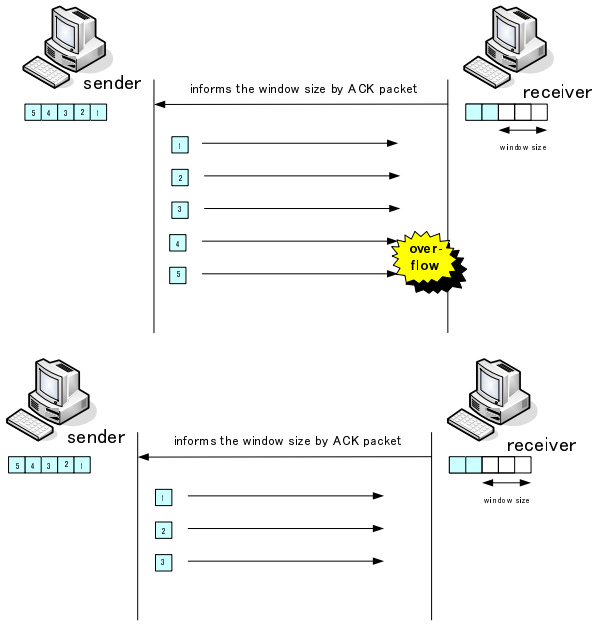


Figure 1.2: Flow Control

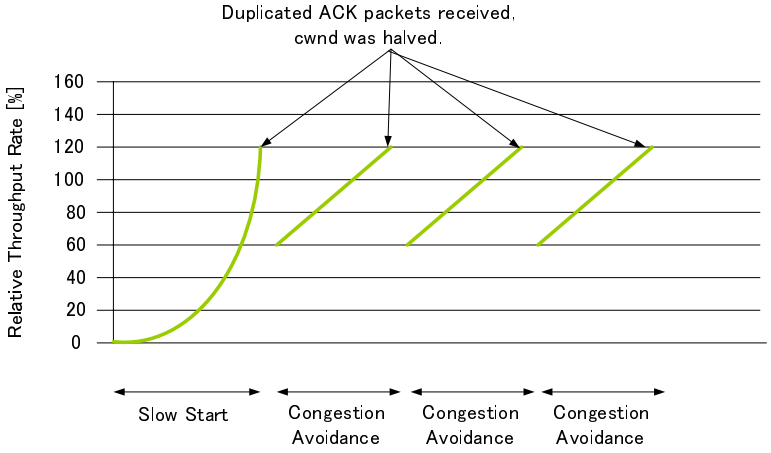


Figure 1.3: Congestion Control

Figures 1.1, 1.2, and 1.3 show the mechanisms of each control. In an error control, the receiver sends acknowledgement packets back to the sender after receiving the data packets. Standard TCP conducts flow control, in which the transmission rate is adjusted based on a sliding window mechanism. In addition, Standard TCP executes congestion control also based on the window control to adjust its transmission rate according to the degree of network congestion. The congestion control of Standard TCP consists of two phases, a slow start phase and a congestion avoidance phase, as shown in Fig. 1.3. In the slow start phase, the congestion window size (cwnd) is doubled every time the sender receives the acknowledgement packet. In the congestion avoidance phase, the cwnd is managed by the Additive Increase and Multiplicative Decrease (AIMD) algorithm as shown below:

$$ACK : cwnd = cwnd + 1$$

$$DROP : cwnd = 1/2(cwnd)$$

Both flow control and congestion control in Standard TCP are conducted between sender and receiver hosts. Thus, particularly in the congestion avoidance phase, cwnd is increased very slowly , as shown in Fig. 1.4

That is, the longer the distance between end hosts and the wider the bandwidth, the more difficult it is to realize efficient data transfer. Therefore, various alternative techniques have been proposed and developed to meet the requirement of highly reliable and high-speed transfer on fast long-distance networks, which can be grouped into the following four major activities: (1) current TCP parameter tuning, (2) modification of the congestion control in the current TCP, (3) proposals for new high-speed transport protocols based on the UDP, and (4) creation of entirely new frameworks on transport protocols suitable for applications in fast long-distance networks working with the support of routers. As examples of (1), dynamic system buffer tuning techniques[FGE03] and the GridFTP protocol, based on multiple parallel TCP streams[ABB+02], are proposed. In addition, in the Internet2 Land Speed Record[lsr], it is reported that high performance is achieved by adopting the jumbo frame

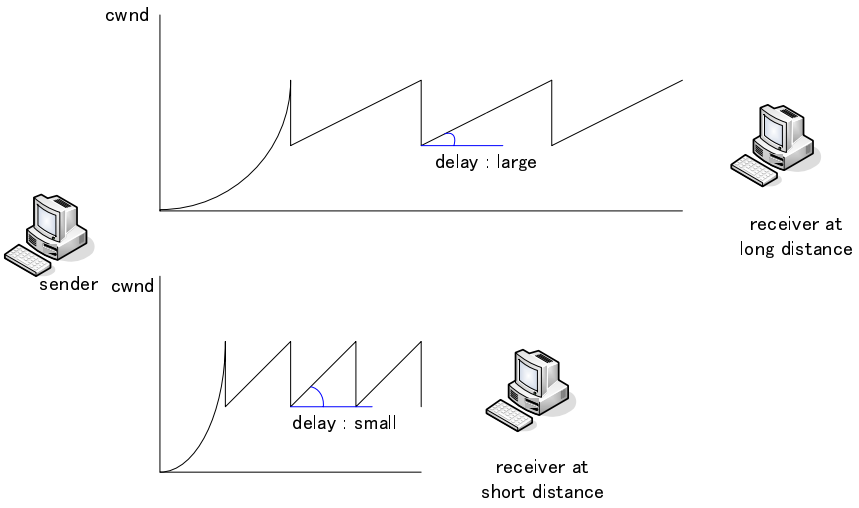


Figure 1.4: Problems in Long Fat Pipe

technique in Standard TCP. In addition, unlike in the case of (4), protocols of (2) and (3) are based upon only end host processing, so that their performances have been reported through various experiments on worldwide testbeds, using a network emulator[RX05][SL04] and a network simulator. However, the evaluation of high-speed transport protocols is not sufficient. In particular, their performances in a realistic network environment are needed in order to determine the performance of each high-speed transport protocol on real networks such as the Internet.

1.3 Problems of congestion controls conducted at both end hosts and intermediate nodes

All nodes along the path must be replaced in order to perform congestion control, in which the intermediate nodes work in conjunction with endpoint equipment. Therefore, this framework was not a realistic solution. However, considering the availability of high-performance CPU technologies and the recent reduction in the price of memory, another framework of

congestion control, in which the intermediate nodes work in conjunction with endpoint equipment, may offer an alternative solution. The present thesis considers two technologies, DiffServ-based schemes and adaptive early packet discarding schemes.

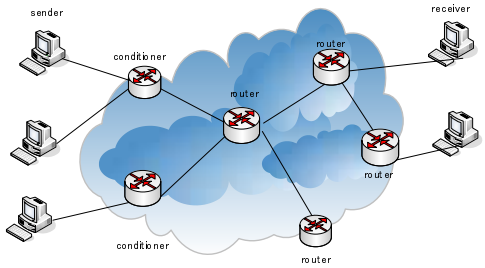


Figure 1.5: Framework of Diffserv Technology

Figure6.1 shows an DiffServ domain model. In the DiffServ domain, the service class of each packet is marked based on the packets contract at the ingress edge node at the DiffServ domain. In addition, each packet is operated in the intermediate node based on the mark in the header. Namely, in DiffServ technology, the intermediate nodes do not just forward the arrival packets, but they also provide suitable operation depending on the service class indicated in the header of the node in order to achieve the relative quality of service in the network. DiffServ technology classifies each packet into a finite service class by a marking strategy and then allocates the appropriate QoS for each class. Thus far, two different types of Per Hop Behavior (PHB) have been proposed, Expedited Forwarding (EF) and Assured Forwarding (AF) PHB. EF PHB provides services such as Virtual Wire, and AF PHB offers a minimum bandwidth allocation service based on Service Level Agreement (SLA). In the present thesis, we examined AF service in order to guarantee the end-to-end minimum bandwidth. In the AF service framework, the packets are discarded probabilistically at the intermediate nodes according to the degree of congestion at each node in order to achieve end-to-end quality of services. A survey of the research activity of DiffServ revealed to be the only study to examine the QoS characteristics of flows over multiple DiffServ domains. Most reports investigated the properties over a single DiffServ domain, whereas the network

is composed of a large number of domains. Therefore, in the present thesis, the throughput characteristics of AF flows over multiple DiffServ domains will be redefined.

Various kinds of application flows have been emerged on the Internet and they expect to achieve their requirements of each quality of service. For example, TCP protocol realizes highly reliable communication for non-real-time application flows. On the other hand, for real-time application flows, real-time communication is more important than reliable communication. Although active queue management (AQM) has the advantage of possibly reducing queuing delays [BCC+98], no schemes have focused on methods to reduce the number of worthless packets in real-time flows, which needlessly consume network resources. Therefore, an early packet discarding scheme in the intermediate nodes is proposed herein in order to improve the delay characteristics of real-time flows.

1.4 Outline of the Thesis

In Chapter 2, an outline of existing high-speed transport protocols is presented. In Chapter 3, the DiffServ framework and the proposed early packet discarding schemes are presented and categorized in conjunction with end equipment.

In Chapter 4, the experimental results, which focus mainly on a single high-speed transport protocol on a testbed network, are presented. In response to emerging requirements for high-throughput data transfer on fast long-distance networks, a variety of high-performance transport protocols have been proposed recently, and preliminary experiments on their performance have been reported. However, these have mainly focused on the throughput characteristics of a single connection or under a stable condition. Therefore, this chapter provides a further investigation of the throughput characteristics of these high-speed transport protocols from various aspects for practical use. The throughput characteristics of multiple connections of different protocols that share a link or with dynamically varying competitive UDP traffic, are examined through experiments on the Japan Gigabit Network (JGN), an open testbed in Japan, for four typical protocols: HighSpeed TCP (HSTCP), Scalable TCP, FAST,

and Simple Available Bandwidth Utilization Library (SABUL). For TCP-based protocols, the influence of the receiver-side OS on throughout performance is also investigated, which is of practical importance from a deployment viewpoint.

In Chapter 5, the experimental results, focusing mainly on multiple high-speed transport protocol flows on a testbed network, are presented. A variety of high-speed transport protocols for high-throughput data transfer over fast long-distance networks have been proposed, but insufficient attention has been paid to the problems involved when these protocols are deployed in shared and heterogeneous network environments such as the global Internet. A variety of high-speed transport protocols — HighSpeed TCP, Scalable TCP, FAST, CUBIC, HTCP, and UDT — is investigated extensively in experiments using several testbed environments, such as the Japan Gigabit Network (JGN)II and the TransPAC2, both of which are open 10-Gbps-class high-speed networks between the United States and Japan/Asia-Pacific. Here, the question as to what will happen if these protocols are run on the Internet is considered. The preliminary results of experiments evaluating the characteristics of these transport protocols in cases with realistic conditions (e.g., a variety of receiver-side OSs, coexisting short-lived Standard TCP flows, and the coexistence of constant bit-rate UDP flows) indicated that none of these protocols are effective or efficient in terms of network resource sharing in various situations, although each protocol behaves very differently. These results provide useful insights to realize high-speed data transfer that can co-exist with such Internet applications as short-term web-browsing and long-term streaming video on heterogeneous global networks.

In Chapter 6, we analyzed the QoS performance in a model consisting of multiple DiffServ domains, and focused in particular on the quality of service provided by Assured Forwarding Service (AF) to achieve statistical bandwidth allocation with AF-PHB. Differentiated Service (DiffServ) is a technology designed to provide Quality of Service (QoS) in the Internet, and is superior to Integrated Service (IntServ) technology with respect to the simplicity of its architecture and scalability of networks. Although various simulation studies

and estimations over testbeds have investigated the QoS offered by the DiffServ framework, most of these focused on the characteristics in a single DiffServ domain. However, the Internet is actually composed of a large number of AS domains, and thus packets are very likely to arrive at their destinations after passing through many different domains. Therefore, we have analyzed the QoS performance in a model consisting of multiple DiffServ domains and have focused particularly on the quality of service provided by Assured Forwarding Service (AF) in order to achieve statistical bandwidth allocation with AF-PHB. Our simulation results show some throughput characteristics of flows over multiple DiffServ domains, which clarifies the impact of network configurations on the QoS over multiple DiffServ domains.

In Chapter 7, we propose two active queue management mechanisms in which packets that experience too much delay are discarded at intermediate nodes based on the delay limit for the application and the delay experienced by each packet in order to improve the delay characteristics of real-time flow. The quality of real-time networked applications is significantly affected by the delay in packets traversing a network. Some real-time applications set limits for acceptable network delay, and thus, a packet that is delayed longer than the limit before arriving at its destination is not only useless to the flow to which the packet belongs but is also harmful to the quality of the coexisting application flows in the network because it may increase the queuing delay in other packets. Therefore, we proposed an adaptive scheme involving two mechanisms in which packets that experience too much delay are discarded at intermediate nodes based on the delay limit for the application and the delay experienced by each packet. Such early discarding of packets is expected to improve the overall delay performance of real-time flows competing for network resources when the network is congested. The results of an extensive simulation show that the proposed scheme has a great potential to improve the delay performance of the real-time traffic not only in homogeneous scenarios but also in heterogeneous scenarios in terms of traffic intensity and the delay requirements of applications.

Concluding remarks are presented in Chapter 8.

Chapter 2

End-to-end congestion control

In section 2.1, the congestion control mechanism of TCP is introduced, and the reason why the Standard TCP cannot use the network resources efficiently in fast long-distance networks is explained. Section 2.2 describes the survey of the targeted high-speed transport protocols in the present experiments.

2.1 Congestion control of Standard TCP

TCP has following functions to realize reliable end-to-end data transfer in IP networks, where the packet reachability is not guaranteed: (1) error control: when packet loss, an error, or packet misordering is detected, the packet is resent and sorted correctly, (2) end-to-end flow control: the sending rate is adjusted adaptively in order to avoid receiver buffer overflow, (3) congestion control: the available bandwidth and the status of the path are estimated and the transmission rate is adaptively adjusted.

The faster the network link becomes, the more important it is for the end hosts to respond quickly to packet losses. In fast long-distance network feedback from the end hosts is delayed because the distance between end hosts is long. Therefore, under these circumstances, it is essentially difficult for the transport protocols, which work based on end-to-end control, to

achieve efficient data transfer.

First, the window control implemented by TCP protocol, in which the sending rate is adjusted based on the combination of the error control, the flow control and the congestion control, is described. In the following, terms and their abbreviations are defined.

- Round Trip Time (RTT): Time required for an IP packet to pass back and forth between end hosts. The RTT is used as a measure of time granularity for the end-to-end controlled protocol. Therefore, the TCP must measure the RTT value continuously in order to determine the timeout value and other various control sequences. The minimum value of the RTT depends on the distance between hosts (propagation delay), while the instantaneous RTT value fluctuates depending on the degree of network congestion along the paths.
- Maximum Segment Size (MSS): The maximum length of a data segment in the TCP packet that the IP packet can convey. The MSS is equal to the value obtained by subtracting both the IP and TCP headers from the MTU.
- Bandwidth-Delay Product (BDP): Product of bandwidth and delay. The maximum amount of data that senders can send back to back until the ACK packet is returned.

In the remainder of this subsection, we assume that the characteristics of targeted Standard TCP are as follow: TCP protocol definition (RFC793), Slow start based on RFC2581 (TCP Reno), congestion control, fast retransmit, fast recovery, and SACK function implementation based on RFC2018 and RFC3517.

1. Each TCP packet that is sent and received has a sequence number (SN) in bytes. If the packet is received successfully, the receiver sends an ACK packet to the sender, in which the SN that is next expected to be sent from the sender to the receiver is set as an acknowledged number (AN). At the sender side, the data packets that are sent are stored until the corresponding ACK packets for the sent data are received. For simplicity, in the following, an ACK packet is supposed to be sent for each transmitted

IP packet. (Actually, the delayed ACK scheme, in which one ACK packet is sent for more than two IP packets, is employed in most cases.)

2. The sender may send the next MSS data on the next IP packet without waiting for the arrival of the ACK packets for the previously sent data. The window size at each moment is set to the smaller of the congestion window (cwnd) and the advertised window.

   For simplicity in the following discussion, the advertised window is assumed to be sufficiently large, thus we can consider the instantaneous window size to be identical to cwnd. Accordingly, adjusting the sending rate is equivalent to adjusting cwnd, and the average throughput is defined as the average cwnd divided by the RTT.

3. cwnd is adjusted by the Additive Increase and Multiplicative Decrease algorithm.

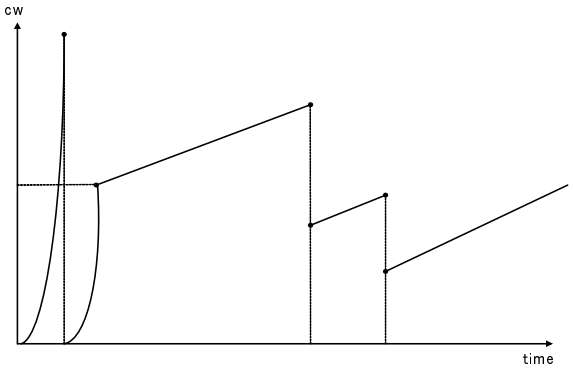


Figure 2.1: Congestion window

As shown in Fig. 2.1, the change of cwnd occurs in two phases: (1) the slow start phase and (2) the congestion avoidance phase.

(1) In the slow start phase, cwnd starts at some initial value. Each time an ACK packet arrives at the sender, cwnd is added by MSS [byte]. Therefore, cwnd increases exponentially. As a result, bursty packet loss occurs due to congestion and a timeout occurs. In this case, the slowstart threshold (ss) is set to half of cwnd when the congestion was detected. Then,

cwnd is again set to the initial value and increases exponentially thereafter. When the cwnd reaches the value of ss, the phase is switched to the congestion avoidance phase.

(2) In the congestion avoidance phase, each time an ACK packet arrives at the sender, the sender increases its cwnd by MSS/cwnd bytes, that is, the packet increases MSS [byte] during RTT [ms]. Although the cwnd is increased slowly compared to that in slow start, intermittent packet losses occur and the next three duplicated ACK packets are sent back from the receiver to the sender. In this case, the packet is retransmitted based on the three duplicated ACK packets (fast restransmit), and both cwnd and ss are set to half of cwnd when the packet loss was detected (fast recovery). On the other hand, for the case in which a timeout occurred due to successive packet losses, the phase is switched to the slow start phase. The average throughput S in the congestion avoidance phase is given by the following equation: S = 1.2 MSS/(p RTT), where the p is packet loss rate. Thus, the problems encountered when TCP is adopted as a transport protocol in a fast long-distance path are as follows: [1] The throughput is given as cw/RTT. Therefore, cwnd must grow to BDP in order to use up the resources on the path. [2] In the slow start phase, although cwnd increases exponentially, it takes a long time to grows to a large value, when BDP is large. If packet loss occurs after the cwnd has increased to a certain extent, a massive number of packets must be retransmitted, and as a result excessive congestion will occur. [3] In the congestion avoidance phase, cwnd is increased by MSS per RTT. Therefore, the rate of increase is limited if the RTT is large. On the other hand, intermittent packet losses occur even if the network is not so congested. The cwnd is then half of the value when the loss is detected, and the cwnd is again increased. However, the rate of increase is very small, so it takes a very long time for cwnd to recover to the original level. Namely, it is absolutely essential that the packet loss rate is very low in order to keep the average cwnd large. For example, we consider the situation in which an average throughput of 10 [Gbps] must be achieved in the stable congestion avoidance phase, that is packets are not dropped successively. MSS is set to 1,500 [byte] and RTT is 100 [ms]. From the average throughput equation, we can obtain the random packet loss rate,

p, as 2e-10. This means that one in every 4,822,500,000 packets is dropped, which is not realistic.

Issues in above [1] are related to the framework of error control in TCP, where control information is exchanged between end hosts. Therefore, it is difficult to solve the problem caused by [1]. First, it is important to prepare the configuration in which cwnd can grow sufficiently. Then, the key to solving the issues in [2] and [3] must be found in order to rapidly increase cwnd. In order to sufficiently increase cwnd, the following settings are basically required in end hosts: (i) a large socket buffer size at the sender, (ii) a large socket buffer size at the receiver, and (iii) to expand the window size, the window scale option and time stamp option must be adjustable.

Consequently, the issues mentioned in [2] and [3] must be solved. One solution for [2] is to set the initial cwnd to be large. This solution can improve the performance of data transfer of short-lived flow. However, limited use is expected for data transfer on long-lived flows. For example, adopting a large MTU and establishing multiple TCP connections is effective for improving the performance to some extent. However, in order to improve the performance significantly, it may be necessary to redesign the slow-start or AIMD algorithm.

2.2 Targeted high-speed transport protocols

In our experiments, we examine six TCP-based high-speed transport protocols (HSTCP, Scalable TCP, FAST, BIC, CUBIC and HTCP) and one UDP-based high-speed transport protocol(UDT). The TCP-based protocols adopt flow control techniques based on the sliding window and can be classified into two categories according to whether they change their congestion window (cwnd) based on the packet loss event or delay information.

- updating cwnd based on the packet loss event: HSTCP, Scalable TCP, BIC, CUBIC, and HTCP
- updating cwnd based on the delay and loss information: FAST

In the following subsections, the mechanisms of congestion control of each high-speed transport protocol are described.

2.2.1 HSTCP

Sally et al. analyzed the performance of Standard TCP in a fast long-distance network and reported a problem of Standard TCP when it is adopted as a transport protocol in a fast long-distance network[Flo03]. The cwnd in Standard TCP flow is updated by the following equation and thus is determined by the packet loss rate, p.

$$cwnd = 1.2/\sqrt{(p)}$$

From the response function of Standard TCP, the packet loss rate has to be smaller than $3 * 10^{-}-8$ in order to achieve a throughput of 1 [Gbps], when the MSS is 1,500 [byte] and the RTT is 100 [ms]. Namely, the packet loss rate must be very low in order to achieve stable 1 [Gbps] performance. The most frequently cited example of the performance achieved by Standard TCP flow in the fast long-distance network is that a Standard TCP connection can achieve a steady state throughput of 10 [Gbps] when the average congestion window grows to 83,333 segments, with 1,500 [byte] packets and a round trip time of 100 [ms]. This means that the packet drop rate is at most one congestion event every 1 2/3 hours. The average packet drop rate of at most $2 * 2^{-10}$ needed for full link utilization in this environment corresponds to a bit error rate of at most $2 * 10^{-14}$. This is an unrealistic requirement for today's network environment.

To solve this problem, Sally et al. proposed the HSTCP protocol. The behavior of the response function of HSTCP flow is defined using three parameters: Low_{window}, $High_{window}$, and $High_P$. In order to realize compatibility with Standard TCP flow, the response functions of Standard TCP and HSTCP are identical in area where the cwnd is smaller than the Low_{window}. For the case in which cwnd is larger than the Low_{window}, it is set to 38-MSS, the HSTCP response function is adopted. $High_{window}$ is set to 83,000 segments, and this value is

equal to the cwnd needed to achieve 10 [Gbps] in the environment where the MSS is 1,500 [byte] and RTT is 100 [ms]. The packet loss rate in this case is $High_P$, 10^{-7}. The response function of the HSTCP flow when the cwnd is larger than the Low_{window} (that is 38-MSS) is defined as follows:

$$cwnd = (p/Low{-}P)S\,Low{-}Window, S = (logHigh{-}Window{-}logLow{-}Window)/(logHigh{-}P{-}logLow{-}P)$$

The cwnd is given as 0.12/p0.835 when the Low_{window} is 38-MSS, Low_P is 10^{-3}, $High_{window}$ is 83,000, and $High_P$ is 10^{-7}. Based on the response function of the HSTCP, the HSTCP flow can reach 1 [Gbps] when the packet loss rate is $8 * 10^{-5}$ in the environment where the MSS is 1,500 [byte] and the RTT is 100 [ms]. This packet loss rate is realistic in comparison to the adoption of Standard TCP, in which case the packet loss rate is $8 * 10^{-7}$.

In case of an equation-based protocol like TFRC, the response function is adopted as an equation directory, however, the response function of HSTCP should be translated to the additive increase and multiple decrease parameters of the AIMD algorithm. HSTCP flows determine their cwnd size based on the AIMD algorithm, in the same manner as the Standard TCP flow. These parameters are constant in the Standard TCP, while the parameters in the HSTCP flow are determined as a function of the size of the instantaneous cwnd in the congestion avoidance phase.

$$ACK : cwnd = cwnd + a(cwnd)/cwnd$$

$$DROP : cwnd = cwnd - b(cwnd)xcwnd$$

2.2.2 Scalable TCP

Scalable TCP takes a similar approach to that of HSTCP in controlling its cwnd size. In the HSTCP flow, the increase and decrease parameters of the AIMD algorithm are determined as a function of the current cwnd, while in Scalable TCP these parameters are constant, as shown below:

$$ACK : cwnd < -cwnd + a$$

$$Drop : cwnd < -cwnd - b * cwnd$$

where a is set to 0.01 and b is set to 0.125. When the cwnd is smaller than 16-MSS, it is determined in the same manner as the Standard TCP flow. In the Standard TCP flow, the packet loss reduces the cwnd by half, while 87.5 percent of the cwnd is adopted after the packet loss detected in the Scalable TCP flow. In the congestion avoidance phase, the cwnd is increased by one segment per RTT in the Standard TCP flow, while the Scalable TCP flow increases its cwnd by $0.01 * cwnd$ every time an ACK packet arrives at the sender. As described above, Scalable TCP used a multiplicative increase multiplicative decrease algorithm for updating the cwnd size.

For example, the cwnd size is 100 when the packet loss is detected, and the cwnd is updated by every ACK packet received from the receiver as 50, 51, 52, and 53 in Standard TCP, and the cwnd is updated as 87.5, 88, 89, and 90, each time an ACK packet is received, that is, the cwnd is multiplexed by 1.01.

In Standard TCP flow, it takes 28 [minutes] to recover the sending rate from 500 [Mbps] to 1 [Gbps] in the environment where the MSS is 1,500 [byte] and the RTT is 200 [ms], whereas in the case of a Scalable TCP flow, it takes 2.7 [s] to recover the original sending rate, with a = 0.01 and b = 0.125.

2.2.3 FAST

Most high-speed transport protocols, including HSTCP and Scalable TCP, determine their cwnd based on the feedback information of packet losses during the congestion avoidance phase, while the cwnd of the FAST flow is updated by the delay information. The cwnd of FAST flows is controlled by following equation:

$$cwnd < -min2 * cwnd, (1 - gamma)cwnd + gamma(baseRTT/RTT)cwnd + \alpha(w, qdelay)$$

In this subsection, information on the FAST protocol is summarized based on [CL03],

[PWSJ03], [JWL04], [ml], and e-mails from Dr. Cheng Jin of FastSoft.

As Dr. S. H. Low noted in [ml], the ”algorithm there can be thought of as a high speed version of Vegas”, and thus FAST might have its origin in Vegas. The FAST might start a “Stabilized Vegas” in [CL03]. A short summary of [CL03] is given below.

1. Vegas has discrete properties and therefore oscillates around equilibrium. To keep the status of Vegas stable, the bandwidth delay product (BDP) must be small.
2. To stabilize Vegas in the environment in which the BDP is large, a sequential model should be adopted instead of a discrete model.
3. The proportional differential (PD) controller, the dual algorithm, and the addition of slow timescale dynamics are introduced to the source in order to stabilize Vegas.
4. As a result, the status of Vegas is stabilized, which can be confirmed based on a Nyquist diagram.
5. Note, however, that “The stabilized Vegas proposes a solution to fix the existing Vegas protocol for stability. It is still no clear whether the stability presented in the paper is significant enough in real networks.”
6. Therefore, the Caltech group based the first FAST implementation, called FAST version 1, on Stabilized Vegas.

[PWSJ03] also tried to stabilize Vegas by using a dual algorithm and adding slow timescale dynamics without adopting the fluid flow model. In [PWSJ03], [CL03] was introduced as an another approach to stabilize Vegas, thus [PWSJ03] is a different approach from [CL03] for stabilizing Vegas. In conclusion, they stated that “Based on our preliminary success in simulations, we are currently pursuing experimental deployment of these kinds of protocols.” in reference to the FAST protocol. Therefore, FAST is thought to have been implemented based on [PWSJ03], but it was not accurate. Dr. Cheng Jin reported that Dr. Paganini’s work was mostly theoretical and different from the present implementation. For the actual

FAST implementation, the reader should refer to [JWL04], and other studies are basically theoretical. Therefore, it seems that the origin of [JWL04] is not [PWSJ03].

The background of FAST is conjectured to be as follows:

- TCP Vegas has a problem in that it can become unstable if the delay is large.
- However, a simple modification enables its stabilization. In addition, [CL03] and [PWSJ03] were introduced to solving the utility maximization problem (as a dual problem) and find an equilibrium point for delay(p) and cwnd(w).
- $\alpha logx$ is the utility function of the FAST flow. Here, p and w can be found by solving the optimization problem, or the utility maximization problem, as shown below: $max \sum \alpha logx R_t <= c$
- The above equation can be rearranged to obtain the following dual problem: $min \sum cp - \sum \alpha log \sum R_p$, which can be solved by adopting a scaled gradient projection algorithm. Here, (w,q), which is the achieve equilibrium point of the equation, can be found upon updating cwnd.

According to Dr. Cheng Jin, the biggest differences between FAST TCP and Vegas lies in how the delay is used. Vegas actually uses delay to avoid packet loss, whereas FAST uses delay for an entirely different purpose, i.e., to measure how far the connection is from its equilibrium state. Knowing the actual distance enables FAST to take the appropriate actions with respect to increasing/decreasing the window size. Namely, Vegas estimates the current throughput based on the measured RTT and determines the cwnd based on this information, whereas FAST decides the cwnd based on the information on the distance between current status and the equilibrium point. There are some additional difference points between Vegas and FAST. For example, the maximum increase rate of the cwnd per RTT is limited to 1 in TCP Vegas, whereas FAST has no such limit. In addition, FAST will use ECN bit when it becomes available. These protocols have the following similarities. They update their cwnd value based on the delay information, and the methods for measuring the RTT adopted in

both protocols are identical. In [CL03], the proportional differential (PD) controller was adopted to stabilize Vegas. However, FAST is an equation-based approach and is not based on a special control theory. Dr. Cheng Jin stated that "Control theory is used as the principle in fast design, and will be used to show the stability of FAST. However, it is not used in terms of implementation." FAST is an equation-based approach. Based on the measured RTT information, FAST can determine how far the current status is from the equilibrium point. If the current point is far from equilibrium, then the cwnd is increased/decreased drastically. When the current point is close to equilibrium, the cwnd is adjusted on a narrow range. The method for measuring the distance to equilibrium is described below. In equilibrium, we want to put alpha packets into the buffer for each TCP stream, and the number of packets we are currently putting into the buffer is cwnd/rtt * q, where cwnd is the window, rtt is the average rtt, and q is the queuing delay. The difference between alpha and cwnd/rtt * q tells us how far we are away from equilibrium. We then go halfway between where we are now and where we want to be (keeping alpha packets in the buffer).

From an implementation standpoint, a FAST flow can only reach the equilibrium point when the sum of the buffer size is equal to the number of flows x α packets. Otherwise, packet losses occur before reaching the equilibrium point and the cwnd is halved, as is the case with a NewReno flow. Therefore, when running FAST TCP, we must consider the buffer size along the path.

2.2.4 BIC

BIC is a high-speed transport protocol in which cwnd is updated by the information of packet loss proposed in 2004.

BIC regards the congestion control as a search problem for an optimal cwnd size, and cwnd is updated in a binary search algorithm. Linux implements BIC protocol as a congestion control algorithm in its kernel standard from kernel 2.6.7.

Figure 2.2 shows the behavior of cwnd observed in a BIC protocol flow.

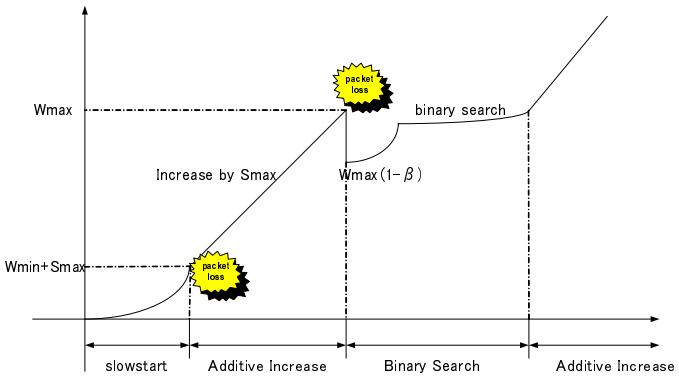


Figure 2.2: congestion window on a single BIC flow

- At the beginning of the flow, the cwnd is updated by the original slow-start algorithm. In order to maintain compatibility with the performance achieved by a Standard TCP flow, the algorithm for updating cwnd is identical for Standard TCP and BIC TCP when cwnd is smaller than 14 MSS. The cwnd is updated to $W_{max} + S_{max}$ by the slow start algorithm (where S_{max} is originally set to 32 MSS).
- Then, BIC enters the additive increase mode in which it increases the cwnd by S_{max} until a packet loss is detected. When the packet loss is detected, the instantaneous cwnd is set to W_{max}.
- The cwnd is set to $W_{max}(1 - \beta)$, where β is set to 0.125. The cwnd updating mode enters the binary search mode.
- In the binary search mode, when the updated cwnd is larger than W_{max}, BIC re-enters the additive increase mode and finds updated W_{max}.

As described above, the BIC flow increases its sending rate aggressively in the additive increase mode. In addition, it detects that its sending rate approaches the target rate, which is the point at which the packet loss was detected previously and updates the rate by the binary search algorithm. Comparing the behavior of the cwnd update phase observed in HSTCP, Scalable TCP, the cwnd approaches the optimal value smoothly in BIC flow.

The achievable maximum throughputs in HSTCP and Scalable TCP flow are limited by the socket buffer size of each flow, whereas that observed in BIC flow (and CUBIC flow) is not limited by the socket buffer size. This is a unique characteristic observed only in BIC and CUBIC flows, among the high-speed transport protocols examined herein.

2.2.5 CUBIC

CUBIC was proposed as a successor to the BIC protocol by the research group that developed the BIC protocol. BIC achieves stable performance by increasing cwnd slowly around the point where the packet loss was detected. BIC achieves its effectiveness by increasing the rate linearly in the additive increase max proving phase. However, the degree of increase of the cwnd for a BIC flow is too aggressive compared to that of the Standard TCP flow. In order to improve this point, the CUBIC protocol, which extends the growth function of the BIC protocol, was proposed and developed. In CUBIC flow, the degree of increasing cwnd is approximately zero around the point at which the packet loss was previously observed. Dr. Injong Rhee noted that ”The essential difference between the BIC and CUBIC algorithms is evident in that the CUBIC algorithm attempts to reduce the amount of change in the window size when near the value where packet drop was previously encountered.” According to the developers of CUBIC, the CUBIC protocol is triggered by the HTCP, ”Our work was partially inspired by HTCP whose window growth function is also based on real time.’ This means that the growth function of CUBIC flow is governed by the cubic function, where the cwnd is updated at the time elapsed since the last packet loss was detected:

$$W_{cubic} = C(t - K)3 + W_{max}$$

where C is the scaling factor, t is the elapsed time from the last window reduction, $W_{max}$is the cwnd when the last packet loss is observed, and K is the 3 root SW_{max} β/C

The difference in the behavior in increasing cwnd between BIC and CUBIC protocol is shown in Fig.2.3.

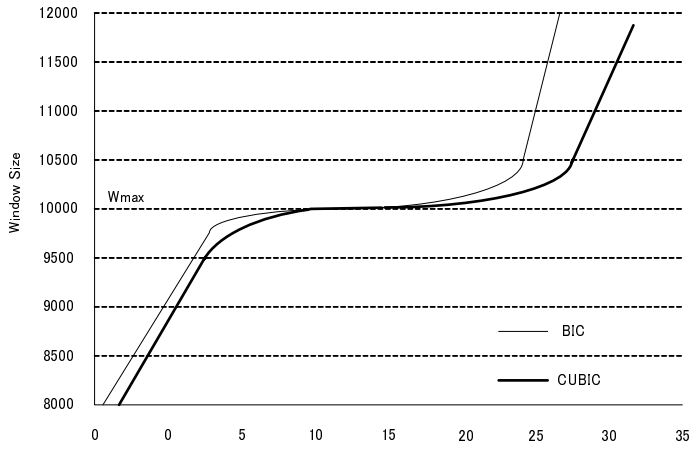


Figure 2.3: Window Adjustment for BIC and CUBIC

Both are similar but the degree of increase of the cwnd is slower in the CUBIC flow than in the BIC flow.

2.2.6 HTCP

HTCP is also a high-speed transport protocol that modifies the congestion control algorithm of the Standard TCP. HTCP updates its cwnd size by the AIMD algorithm and changes the increasing parameter α as a function of the time elapsed since the last packet drop experienced by its source. That is, HTCP changes its increase parameter as a function of the elapsed time since the last congestion event occurred, as follows:

Δ: time in seconds that has elapsed since the last congestion event experienced by a flow
$\alpha(\Delta)$: AIMD increases the parameter according to some function, denoted as $\alpha(\Delta)$

$$\alpha(\Delta) = 1(\Delta < \Delta L) = \alpha H(\Delta)(\Delta > \Delta L)$$

ΔL: threshold for switching from standard/legacy operation to the new increase function.

As noted in the previous subsection, the CUBIC protocol was partially inspired by HTCP. Both CUBIC and HTCP manage the cwnd based on the time elapsed since the last packet was dropped, whereas the HSTCP and Scalable TCP control the cwnd based on the information

of the packet loss event. To control the cwnd smoothly according to the network status, it might be useful to use the historical information on packet loss and delay.

2.2.7 UDT

Most high-speed transport protocols are based on the Standard TCP and modify its congestion control algorithm of Standard TCP. In the present experiment, we also investigated high-speed transport protocols based on the UDP protocol, UDT(UDP-based Data Transfer Protocol). UDT has some differences from other the high-speed transport protocols investigated in the present experiments. First, UDT is a high-speed transport protocol that is based on the UDP protocol. Second, the UDT is an application level solution, and thus can work on various kinds of OS, whereas most high-speed transport protocols can only work on Linux.

The UDT congestion control algorithm combines rate control and window control (flow control), where the rate control adjusts the packet-sending period and the flow control limits the number of unacknowledged packets. The UDT measures the available bandwidth along the path periodically using the receiver based packet pair method and adjusts its sending rate (rate control).

Originally, UDT was not intended to replace TCP in the Internet, where the bottleneck bandwidth is relatively small and there are a large number of multiplexed short life flows. The UDT was expected to be used in situations in which a small number of bulk sources share abundant bandwidth. In other words, the UDT was developed for a Grid environment, not for the internet. However, users can obtain the source code for the UDT from the web and install it to their machines to enjoy high-speed data transfer. Therefore, we examined the UDT protocol as a high-speed transport protocols in the present experiments.

2.3 Activities on evaluation of high-speed transport protocols

The activities on evaluation of high-speed transport protocols is divided into three cotegories: simulation based, network emulator (working on PC routers) based and testbed network based.

There are several evaluations of high-speed transport protocols via the simulation. The performance of HSTCP and the impact of its use on the present implementation of TCP is analyzed in different network conditions, including different degrees of congestion, different levels of loss rate, different degrees of bursty traffic and two router queue management policies (RED and DropTail) were presented[AD]. And detailed results of evaluation via simulation targeted various high-speed transport protocol variants in [RX05]. These results indicate that each high-speed transport protocol can achieve effectiveness on high-speed network but they all have difference on achieving the fairness between coexisting different protocol flows.

A number of research groups have also been conducting a variety of experiments for high-throughput data transfer on fast long-distance networks. For example, the Hamilton Institute group developed a common benchmarking environment using dummynet, where different protocols were implemented with common network stacks and were evaluate in terms of a common performance measure[LLS]. They show detailed performance of each targeted high-speed transport protocols in both where no background traffic exists and background traffic coexist. The web traffic is used as background traffic.

The BIC Lab team tried to realize realistic performance evaluations by creating a realistic network environment in which high-speed transport protocols are likely to be used with background traffic[HKL+06], [inj]. They targeted to evaluate the protocols in the environment with more realistic background traffic. Therefore, they plan to use some of the existing traffic generators which rely on real network traces as seeds for generating synthetic network traffic.

The Work in [CAK+05] were the first evaluations in testbed targeted various high-speed transport protocols. Various basic characteristics of each high-speed trasport protocols were shown. For benchmarking in 10Gbps class environment, the Caltech group is constructing the Wan-in-Lab, which is a 2,400 [km] long-haul fiberoptic testbed in the lab as a benchmark testing platform for the research community and for evaluating different protocols in a more realistic environment[wan].

Chapter 3

Controls on intermediate nodes

3.1 DiffServ

The specifications of Assured Forwarding (AF) Per Hop Behavior (PHB) in the DiffServ framework are provided in [HBWW99]. The architecture to realize AF PHB is presented by D. D Clark and W. Feng in [CF98]. They proposed the TSW tagger as a marker, which is suitable for TCP flow, and RIO queue management based on RED queue management implemented at intermediate nodes to assure the throughput of the TCP flow to the contracted value. They described the contract between users and ISP achieved by this framework as an expected capacity, rather than a strict guarantee. They evaluated the performance of AF flows, where the number of flows is 50, the sum of bandwidth contract is 50 [Mbps], and bottleneck bandwidth is 50 [Mbps], and found that AF service based on [CF98] cannot provide contracted service quality for each flow. Therefore, it is difficult to apply DiffServ AF architecture on the Internet. However, the targeted model in [CF98] has some questionable areas regarding the evaluation of the performance of AF service. First, the sum of the target rate of each flow is set equal to the value of the bottleneck bandwidth. Second, the maximum value of the RTT was set to be very large, 160 [ms]. Therefore, we will evaluate its performance in a more realistic model. In [GDJL00], it is shown that the service class-based

packet marking is useless in the circumstance where the sum of the target rates of each flow is equal to the bottleneck bandwidth, therefore the sum of the contract of each flow should be smaller than the bottleneck bandwidth.

N. Seddigh, B. Nandy, and P. Pieda reported that the throughput characteristics achieved by AF service affect the RTT, contract, packet size, and parameters adopted at RIO active queue management[SNP99]. It is also reported that the quality of AF service is affected by the RTT and the number of active flows, which indicates that the parameter of RIO must change according the number of active flows[NSP99]. In addition to the interaction between coexisting UDP flow and TCP flow, the contract rate between users and the ISP, the number of aggregate flow, the type of TCP, and the buffer size, the parameter of RIO is very important for the performance of end-to-end AF service.

RIO (RED with IN/OUT), which is adopted as the queue management algorithm in the DiffServ AF framework, has two packet drop classes for IN and OUT packet, and thus can provide users with two different service classes. The Time Sliding Window Three Color Marker (TSWTCM), a conditioner that can treat three service classes, has been proposed. In our research, however, we adopted a two color marker as a conditioner because the Three Color Marker is suitable for environments in which the UDP and TCP flow coexist[GDJL00]. Figures 3.1 and 3.2 show the conditioner and RIO queue, respectively.

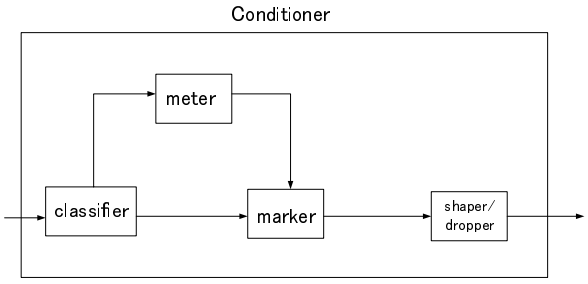


Figure 3.1: Conditioner

W. Fang et al. selected the Time Sliding Window (TSW) tagger as a meter and marker as a result of adopting several algorithms[CF98]. On the other hand, the Token bucket algorithm

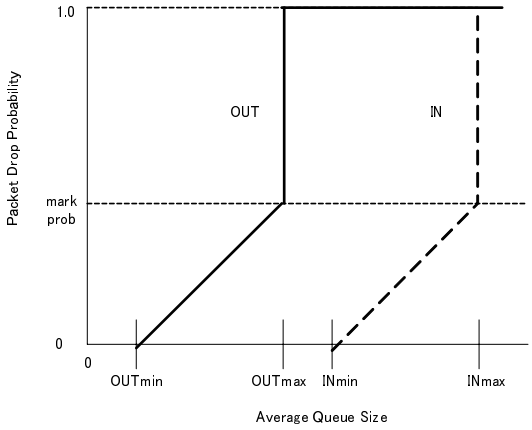


Figure 3.2: RIO (RED with IN/OUT)

was selected as the meter and marker algorithm in [JN98], because the TSW tagger marks a class on each packet probabilistically so that the marking probability is not strictly related to the circumstances.

Based on our results, we can observed a number of points on the characteristics of Tagger, as follows:

- Compared to packet marking using the TSW marker, The Token bucket mechanism marks packets in a bursty manner. Therefore, it appears desirable to adopt the TSW Tagger as a meter and marker.
- In the case of adopting the token bucket algorithm, the parameters should be selected based on the burst length.
- In a previous study, it was recommended to set for window size in the TSW Tagger, avg-interval, to 1 [s]. We adopted an avg-interval of 1 [s] for our simulation based on the results acquired in the previous works.
- No difference in the throughput characteristics of flow over a single domain DiffServ network achieved by Token Bucket and TSW tagger was observed when the token bucket parameter is set to appropriate value.

- W. Feng et al. reported that the difference in the meter and marker architecture does not affect the performance by rebuilding the TCP architecture to suit the DiffServ framework[FPes].

Although various simulation studies and estimations over testbeds have investigated the performance offered by the DiffServ framework, almost all of these studies have focused on the characteristics in a single DiffServ domain. However, the Internet is actually composed of a large number of domains. The characteristics of TCP flows over multiple DiffServ domains were investigated, and the end-to-end throughput characteristics of TCP flow were not affected by the packet re-marking that occurred at the cascaded tagger[Fan99]. In the present research, we investigate the throughput characteristics of flows that pass through multiple DiffServ domains, and, in particular, the effect of the cascading tagger on the throughput characteristics.

3.2 Active Queue Management

Real-time applications such as VoIP and video conferencing adopt UDP as a transport protocol. For example, real-time applications such as VoIP define service classes based on an end-to-end packet delay limit for a flow in a network. In such applications, a packet that is delayed longer than this limit before arriving at its destination is not only worthless, but is also harmful to the quality of the application. In the present research, we propose the adoption of active queue management at intermediate nodes in order to improve the quality of real-time traffic flows[BCC+98]. In [BCC+98], it is reported that an active queue management mechanism may have some advantages, including providing lower-delay interactive service. That is, by keeping the average queue size small, queue management will reduce the delays experienced by flows. This is particularly important for interactive applications such as real-time traffic. In the present research, we propose a type of active queue management in which the packets are discarded according to the status of an intermediate node to improve

the delay characteristics of real-time flows.

3.3 Related work

From the viewpoint of deployment, it is easier to implement controls only between end-to-end equipment in a network. However, networks can obviously provide more finely grained services when intermediate nodes work in conjunction with endpoint equipment. Various schemes using intermediate nodes have been proposed to realize better network performance.

Early packet discarding, in which some packets might be dropped even though the queuing buffer is not full, is not an entirely new concept, and was introduced in the context of Active Queue Management (AQM) [BCC+98]. For example, in Random Early Detection (RED) [FJ93] and its variants, in order to mitigate instability and unfairness in the throughput of TCP flows traversing an intermediate node, the node probabilistically discards incoming packets based on its queue length by introducing random losses instead of burst packet losses. In contrast, the proposed scheme attempts to reduce the delay in real-time application flows. Although AQM has the advantage of possibly reducing queuing delays [BCC+98], to the best of our knowledge no schemes have focused on methods to reduce the number of worthless packets in real-time flows, which needlessly consume network resources. In a recent study, a packet discarding technique was introduced to achieve fairness between wired and wireless TCP sessions [MMY04], and another deterministic AQM was proposed to improve TCP throughput over 3G wireless links [AC06]. In the context of TCP over ATM (in which one large packet in the upper-layer is split into several small cells in the lower-layer), if one cell is dropped then all remaining cells belonging to the same packet are useless and degrade the upper-layer performance, even though they have not yet been dropped. Early Packet Discard (EPD) [RF95] was proposed to improve the performance of TCP with respect to this fragmentation problem. The proposed scheme is similar to EPD in terms of the basic idea of discarding packets in advance if they are likely to be useless to the overall performance of the application. To reduce delays in real-time flows, a variety of packet

scheduling policies, such as Earliest Deadline First (EDF) [GGP97], have been developed along with various resource reservation schemes to optimally reorder the sequence of packets belonging to various flows. On the other hand, the proposed scheme does not require such scheduling policies, but can be combined with them. Note that while complex packet scheduling schemes are not generally scalable, the proposed scheme is so lightweight that it can even be applied to heavily loaded nodes, such as core routers. The expected queuing delay of a packet in an intermediate node using MTQ and QTL mechanisms can easily be obtained from the queue length upon arrival of the packet, which can readily be managed at the node. The delay is then simply compared with an MTQ/QTL value in the packet header or subtracted from the QTL value there. In QTL, which is similar to the TTL mechanism, the cost of updating the QTL value by subtraction is negligible. Another way to reduce delays in real-time flows is by over-provisioning, in which a network has sufficient bandwidth to ensure that all flows are within their delay limits [FTD03]. However, the appropriateness of this solution depends on the time and circumstances of the network, because traffic demands change faster than network provisioning in many situations. A scheme to fully and efficiently utilize the existing network resources is still required, and the abovementioned approaches are complementary.

3.4 Adaptive early packet discarding at intermediate nodes

The MTQ and QTL mechanisms were originally proposed in a lightweight practical framework for active networks[TKFO03], and their effectiveness for time synchronization over a network was shown [KMT+05]. However, in the present paper, we apply these mechanisms to more general delay-sensitive applications.

The proposed target model, in which real-time application flows compete for resources in a single domain network, is outlined in Fig. 3.3. The MTQ and/or QTL parameters are set in the header of each packet entering the domain at the edge nodes, and these packets are forwarded to intermediate nodes. Every time a packet arrives at the intermediate nodes, it is

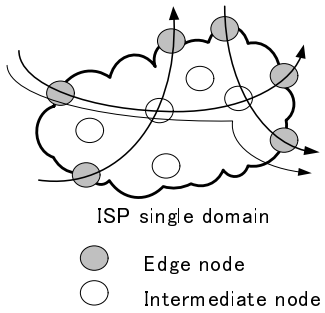


Figure 3.3: Target network model

queued or discarded according to the MTQ and/or QTL mechanisms described below and in Fig. 3.4.

Packets using the MTQ mechanism are managed based on the local queuing delay limit at each intermediate node. The processing procedure for each packet at edge and intermediate nodes is as follows:

1. A value for MTQ (i.e., the maximum queuing delay in one node) is set in the header of each packet at an edge node.
2. When a packet arrives at an intermediate node and if the queuing buffer bound for the next node to which the packet will be forwarded is not full, then (1) the local queuing delay for the packet is calculated based on the queue length and output link bandwidth, and (2) if the value for MTQ in the packet header is larger than the calculated queuing delay, the packet will be queued. On the other hand, (3) if the calculated queuing delay is larger than the MTQ value, the packet is discarded.

Because the MTQ mechanism limits the local queuing delay at each intermediate node, setting the MTQ parameter is equivalent to limiting the size of the queuing buffer at every node through which the packet passes.

Packets using the QTL mechanism, on the other hand, are managed based on the global (total) queuing delay limit throughout the network (from an ingress edge node to an egress edge node):

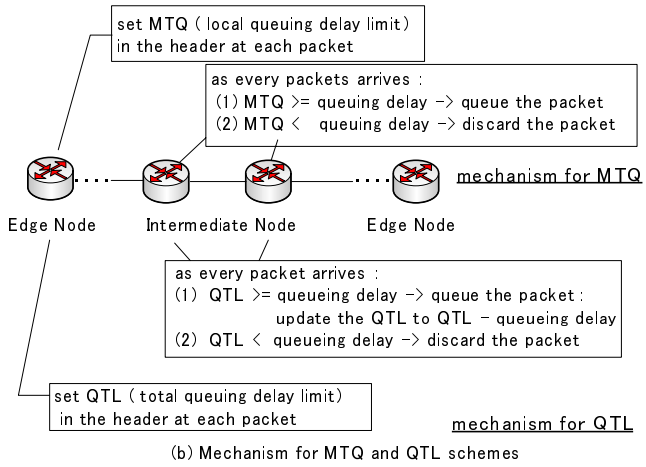


Figure 3.4: MTQ and QTL mechanisms

1. A value for QTL (i.e., the maximum queuing delay in the network) is set in the header of each packet at an edge node.
2. When a packet arrives at an intermediate node and if the queuing buffer bound for the next node to which the packet will be forwarded is not full, then (1) the local queuing delay is calculated based on the queue length and the output link bandwidth. (2) If the value for QTL in the packet header is larger than the calculated queuing delay, then the value for QTL in the packet header is updated to be “QTL minus the calculated queuing delay” and the packet will be queued. On the other hand, (3) if the calculated queuing delay is larger than the QTL value, the packet is discarded.

Because the packet is discarded if the updated QTL value is not positive, the initial value of the QTL parameter corresponds to the global queuing delay limit.

Chapter 4

Experiments for High-Speed Transport Protocol of a Single Flow

4.1 Introduction

The majority of network applications adopt the Transmission Control Protocol (TCP) rather than the User Datagram Protocol (UDP) as the transport layer protocol on IP networks. This is because the TCP has important two functions; one is the error control function providing a reliable, error free data transmission and another is the congestion control mechanism realizing a modest sharing of network resources.

However, the current TCP is not always suitable for applications which require highly reliable and high speed transfer of a huge amount of data over long haul networks, such as in the Grid Computing environment [FP00]. This results from its slow start algorithm and the Additive Increase Multiplicative Decrease (AIMD) algorithm in congestion control; the former limits initial throughput to a small value, and the latter further causes slow and inefficient adjustment of the transmission rate due to a large bandwidth-delay product. Hence, the flow of the current TCP can not run out of vast bandwidth from the beginning of the flow and maintain a high stable throughput, even though abundant network resources are available[Flo03].

Therefore, various alternative techniques have been proposed and developed to meet the requirement of highly reliable and high speed transfer on fast long-distance networks, which can be grouped into the following four major activities : (1) current TCP parameter tuning, (2) modifying the congestion control in the current TCP, (3) proposals for new high-speed transport protocols based on the UDP and (4) entire new frameworks on transport protocols suitable for applications in the fast long-distance networks working with the support of routers. As examples of (1), the dynamic system buffer tuning techniques[FGE03] and the GridFTP protocol based on the multiple parallel TCP streams[ABB+02] are proposed. And in the Internet2 Land Speed Record[lsr], it is reported that the high performance is achieved by adopting the jumbo frame technique in Standard TCP. In addition, unlike in the case of (4), protocols of (2) and (3) are based upon only end hosts processing, so that their performance have been already reported through various experiments on worldwide testbeds[yan].

However, those experiments on various testbeds have mainly focused on the throughput characteristics under a stable condition, particularly for a single flow, except for recent experiment[BCRJ04]. Therefore, a wide variety of throughput characteristics should be investigated in more actual network environment for aiming at practical use. In this chapter, we will show some results on the throughput characteristics of high-speed transport protocols through extensive experiments on the Japan Gigabit Network (JGN), an open testbed in Japan. This paper extends the results in [KHTO04]. In particular, we treat some practical cases; multiple connections of different protocols share a link in one case and the amount of UDP traffic sharing some link with them changes drastically in another case. We will deal with some practical high-speed transport protocols of type (2) based on the TCP (High-Speed TCP(HSTCP), Scalable TCP and FAST), and of type (3) based on the UDP (Simple Available Bandwidth Utilization Library(SABUL)), using implementations contributed by several researchers. For TCP-based protocols, the influence of the receiver-side OS (TCP implementation) on the throughput performance, which is of practical importance from the deployment viewpoint, is also investigated.

This chapter is organized as follows. In Section 4.2, we explain new transport protocols adopted in our experiment. In Section 4.3 we present the experimental configuration. Experimental results are presented in Section 4.4. Finally, in Section 4.5, we conclude with some consideration on transport protocols for fast long-distance networks.

4.2 Targeted High Speed Transport Protocols

In our experiments, we targeted three TCP based protocols(HSTCP[Flo03], Scalable TCP[Kel03], FAST[JWL04]) and one UDP based protocol (SABUL[GG03]) and investigated their throughput performance on the JGN.

HSTCP and Scalable TCP change their congestion window size(*cwnd*) according to the AIMD algorithm with the following equations, same as the current TCP (which we will refer to as the Standard TCP(RFC2581:TCP Congestion Control)[APS99] from now on) during the congestion avoidance phase, where a is the increase parameter and b is the decrease parameter.For the Standard TCP, the values of a and b are 1/*cwnd* and 0.5, respectively.

$$ACK : cwnd = cwnd + a * MSS$$

$$DROP : cwnd = cwnd * (1 - b)$$

The HSTCP behaves identically to the Standard TCP for a small *cwnd*, but in the area of window size larger than a threshold (for example, 38 maximum segment size(MSS)), *cwnd* is governed by a modified AIMD algorithm, where a and b vary in a complex way depending on the current value of the *cwnd*. Scalable TCP also modifies the characteristic AIMD behavior of the Standard TCP and has a threshold window size (the default size is 16 MSS). When the *cwnd* exceeds the threshold, it will be updated using a of 0.01 and b of 0.125. Table 4.1 summarizes the values for a and b adopted in each protocol.

FAST is a high-speed transport protocol which modifies the TCP Vegas[Veg]. FAST uses the same acknowledgement and window control mechanism as the Standard TCP, but

Table 4.1: Parameters in the AIMD algorithm of TCP-based Protocols

	Standard TCP	HSTCP	Scalable TCP
a	1/*cwnd*	a(*cwnd*)	0.01
b	0.5	b(*cwnd*)	0.125

it does not use Standard TCP functions like slow start and the AIMD algorithm. It updates its *cwnd* based on the measured RTT information in the same manner as TCP Vegas. This updating manner is different from the other TCP-based transport protocols such as HSTCP and Scalable TCP in that they decide their *cwnd* based on the packet loss event information.

Several new UDP-based protocols have been proposed like TSUNAMI and RBUDP. We treat SABUL as an example of UDP-based protocol among them, because it has received an award at the Supercomputing Bandwidth Challenge [bc].

SABUL establishes two connections simultaneously between the sender and the receiver; one is a UDP connection and the other a TCP connection. SABUL uses a UDP connection to transfer data and a TCP connection to transfer ACK and control packets, which provide error information and the total number of received packets during the constant period from the receiver to the sender. Based on this information, lost packets are retransmitted on the UDP connection and in addition the sending rate is periodically updated.

Table 4.2 lists the implemented protocols adopted in our experiment.

Table 4.2: Targeted Protocols

Protocol	Proposer	Contributer for Code	Version
HSTCP	Sally Floyd	G.Fairey	patch for Linux 2.4.19
Scalable TCP	Tom Kelly	Tom Kelly	patch for Linux 2.4.19
FAST	CALTECH	CALTECH	running on Linux 2.4.20
SABUL	Yunhong Gu	Yunhong Gu	version 2.2

4.3 Experimental Setup

We conducted experiments on the JGN, which was a public testbed for research and provided users with ATM paths of up to 600[Mbps]. In our experiment, we use two types of UBR PVC paths, nicknamed *JAPAN* and *EARTH2*, as introduced in Table 4.4. (These nicknames derive from the length of the each path, i.e. *JAPAN* path was set its length to around Japan, and the length of *EARTH2* path is equal to the distance of round the earth two times.)

The network topology and equipment specifications are illustrated in Fig. 4.1 and Table 4.3, respectively.

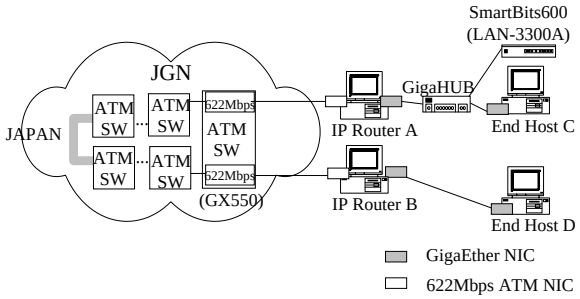


Figure 4.1: Network Topology

Table 4.3: Equipment Specifications

	IP Router A,B	End Host C	End HOST D
OS	Red Hat Linux 8.0 Kernel 2.4.18	Red Hat Linux 9.0 Kernel 2.4.20	Red Hat Linux 9.0 Kernel 2.4.20 Windows 2000 SP3 Solaris9 12/02 FreeBSD 4.7 Release
CPU	PentiumIV 2.4[GHz]		
Memory	512[Mbyte]	1[GByte]	
PCI BUS	32[bit]		
EtherCard	Intel Pro		
ATM NIC	Fore(Marconi) HE 622SMF	-	

Table 4.4: Path Characteristics in Japan Gigabit Network

Path Nickname	Distance[km]	*1[Mbps]	*2[Mbps]	*3[Mbps]	Measured RTT[ms]	Bandwidth-Delay Product
JAPAN	8,000	600	522	305	100	3.88[Mbyte]
EARTH2	40,000	120	105	98	460	5.75[Mbyte]

1: ATM Cell Rate, 2: Theoretical Maximum Bandwidth, 3: Measured Maximum Bandwidth

In Fig. 4.1, we establish TCP or UDP connections from end hosts connected with router A to those connected with router B. The performance of TCP-based protocols is monitored by Iperf[ipe], while that of SABUL is measured by itself. We will show the throughput characteristics for every two seconds, unless otherwise noted.

Bottleneck bandwidth and round-trip time(RTT) in a load test measured by Smartbit on *JAPAN* and *EARTH2* paths are listed in Table 4.4. In the load test over the *JAPAN* path, no UDP packet losses are observed in a range of throughput up to 310[Mbps], which corresponds to 305[Mbps] of TCP throughput. However, with reference to Table 4.4, bottleneck bandwidth of the *JAPAN* path is 522[Mbps], which indicates that the actual bottleneck bandwidth is limited to 305[Mbps] in our configuration. This is due to the PCI bus performance restriction in IP router machines.

On the other hand, to avoid ATM cell losses caused by traffic policing on the ATM PVC path, we insert a 100[Mbps] hub in the access line to the *EARTH2* path, which is expected to be a traffic shaper. In this configuration, no packet loss events occurred on the *EARTH2* path, in a range of throughput up to a 100[Mbps] UDP packet load with Smartbit, which corresponds to 98[Mbps] of TCP throughput.

Hereafter, we use a configuration in which the buffer size at the sender, s-buf, is set at 8[Mbyte] or autotuning of the buffer size is turned on[aut], and the buffer size at receiver, r-buf, is set at 8[Mbyte] and 16[Mbyte] in the *JAPAN* and *EARTH2* path test environments, respectively, referring to the value of Bandwidth-Delay Product in Table 4.4.

We adopted Linux as the sender-side OS in our experiments because, except for SABUL, the function codes are provided as patches for source code(HSTCP and Scalable TCP) or the

kernel(FAST) in the Linux.

Before showing our experimental results, we note the following. The JGN paths used in our experiment are ATM UBR PVC connections which share ATM network resources with other constant or temporary traffic on the JGN. To avoid the influence of such unexpected and uncontrollable concurrent background traffic, we conduct five or ten trials in each experiment with the same configuration, and then discard irregular cases. Unless otherwise noted, the results we show hereafter are for the cases exhibiting good throughput performance, which is expected to be obtained when the influence of background traffic is weak enough to be negligible.

4.4 Experimental Results

In this section, we investigate various performance characteristics of several high-speed transport protocols from practical viewpoints.

First, as fundamental characteristics, throughput performance of a single connection is examined in Section 4.4.1, and that of multiple connections of the same transport protocol in Section 4.4.2. Moreover, we treat cases of more practical interest in the following subsections. In actual networks, traffic from the high-speed transport protocols treated here is very likely to share a link with UDP traffic as well as with traffic of different high speed transport protocols, which are investigated in Sections 4.4.3 and 4.4.4. Finally, there are actually various TCP implementations with many operating systems. We examine behavior of some of them depending upon their TCP implementations. Section 4.4.5 will show the effect through experiment results.

4.4.1 Throughput Performance Comparison on a Single Connection

In this section, we report throughput of a single connection adopting each targeted protocol. Figure 4.2 presents throughput characteristics of a single flow on the *JAPAN* path, for each

target high-speed transport protocol shown in Table 4.2. The autotuning buffer function at the sender is turned on. We conduct five trials and show the results for the largest throughput characteristics obtained among them.

We also show some properties of each protocol flow, including time to achieve 100[Mbps] throughput, maximum throughput, average throughput over 300 [s] and packet loss rate in Table 4.5.

And we also trace the size of *cwnd*(i.e., the burst size for transmission) in the TCP-based protocol flows via *tcpdump*[tcp] in Fig. 4.3.

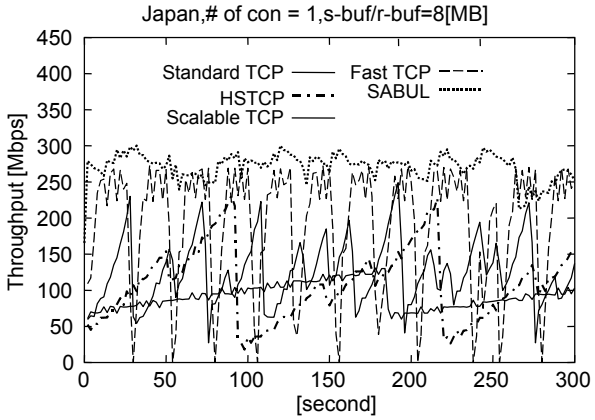


Figure 4.2: Throughput Characteristics on a Single Flow on *JAPAN* Path

As Fig. 4.2 and Table 4.5 indicate, there are distinct differences in the behavior of high-speed transport protocols treated here. With a high-speed transport protocol, throughput increases rapidly and the achievable maximum throughput is also larger than that of the Standard TCP. Especially with SABUL, the flow could achieve about 280[Mbps] throughput constantly. The bottleneck bandwidth in our environment is 305[Mbps], hence on adopting SABUL as the transport protocol, 92% of the available bandwidth could be used.

Table 4.5: Characteristics of Various Transport Protocol Flows

	time to achieve 100Mbps[s]	maximum throughput[Mbps]	average throughput[Mbps]	packet loss rate[%]
Standard	120	120	92	0.0082
HSTCP	35	230	106	0.023
Scalable TCP	10	250	125	0.058
FAST TCP	<< 1	272	186	0.028
SABUL	<< 1	300	271	0.12

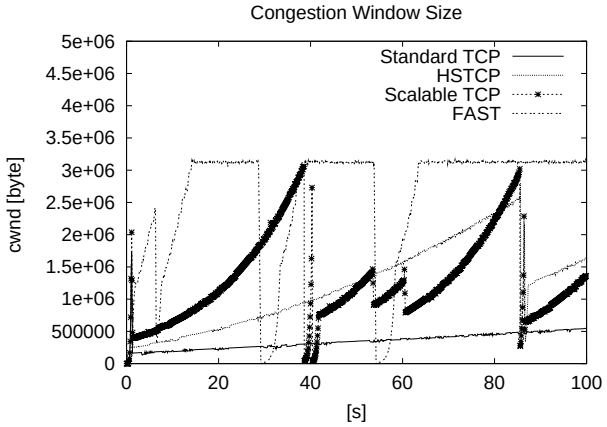


Figure 4.3: Congestion Window Size

We find that the FAST flow achieves good performance, but at the same time we observe that the throughput deteriorates periodically. We assume that this may come from the fact that we cannot set an appropriate value for the TCP Vegas parameters which must be set in FAST and it seems to have a large impact on the throughput characteristics.

In Fig. 4.2, we find that the throughput changes dynamically in HSTCP and Scalable TCP flow. This is because the *cwnd* becomes too large very quickly, which causes congestion in the networks, shrinks its size, and drastically deteriorates the throughput. A sequence of these events occurs repeatedly, as shown in Fig. 4.3.

In addition, Fig. 4.3 tells us that the changes of *cwnd* of the FAST flow is considerably different from other protocols. This is likely to come from the fact that the FAST controls its *cwnd* based on the measured RTT information instead of the packet loss information.

Comparing packet loss rates of protocols in Table 4.5, the larger throughput the protocol achieves, the greater loss rate it exhibits, except for FAST.

Next we compared the throughput characteristics of the multiplexing the Standard TCP flow (# of flows is set to eight.) with that of the high-speed transport protocol flows. In Fig. 4.4, it is observed that multiplexing Standard TCP flows can achieve similar levels in throughput of a single HSTCP flow or a Scalable TCP flow.

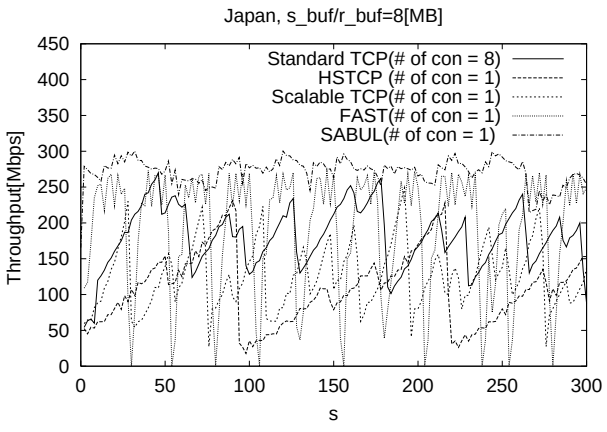


Figure 4.4: Multiple Standard TCP Flows - Single High-Speed Transport Protocol Flow

We also investigate the throughput characteristics for a single flow in the *EARTH2* environment, which has a longer RTT and a smaller bandwidth. From Fig. 4.5 and Table 4.6, we observe tendencies similar to those in the *JAPAN* environment on the throughput performance.

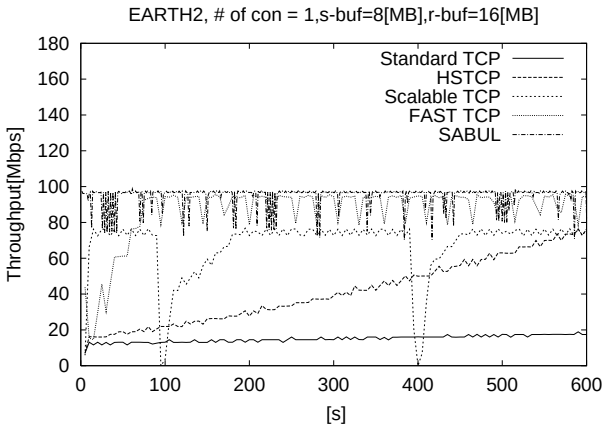


Figure 4.5: Throughput Characteristics on *EARTH2* path

Table 4.6: Maximum and Average Throughput on *EARTH2* path

Protocols	Maximum Throughput[Mbps]	Average Throughput[Mbps]
Standard TCP	20	9.8
HSTCP	78	42.1
Scalable TCP	78	55.8
FAST	100	90.1
SABUL	100	98

These results indicate that the flows adopting the new high-speed transport protocols can achieve higher throughput than that of the Standard TCP. In particular, SABUL and FAST flows maintain high throughput characteristics in our environment.

4.4.2 Multiplexing Characteristics of Flows from Homogeneous Protocol

Next, we investigate multiplexing characteristics of flows from same protocol; characteristics of two flows will be examined as an example below. We have found that the HSTCP, Scalable TCP, FAST and SABUL protocol could ensure fairness in throughput among homogeneous protocol flows, like Standard TCP. Additionally, total throughput of two flows is larger than that of a single flow, and all the protocols investigated effectively improve their throughput performance by setting up multiple connections.

Figure 4.6 provides an example of throughput characteristics, where two Scalable TCP flows were established simultaneously.

In addition, we show the average throughput over 300[s] of each of the transport protocol in Fig. 4.7. It tells us that all the protocols can ensure the fairness in the throughput characteristics among homogeneous protocol flows.

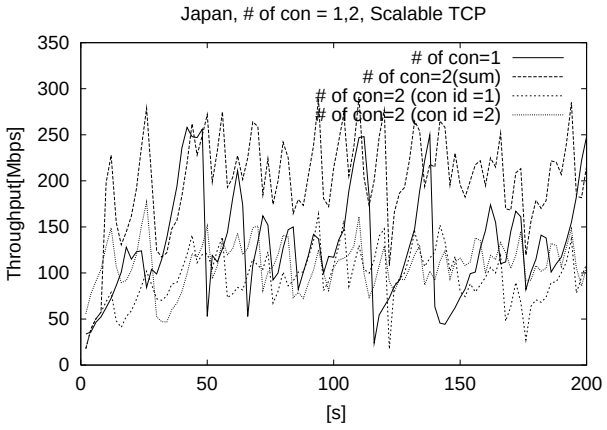


Figure 4.6: Throughput Characteristics in Multiple Scalable TCP Connections

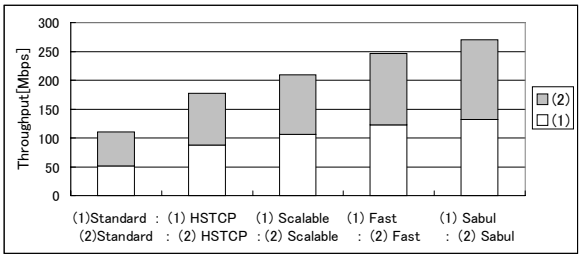


Figure 4.7: Average Throughput in Intra-Protocol Flows

4.4.3 Ability to Adapt to Changing Available Bandwidth

In this section, we investigate the behavior of flows of each transport protocol when the network congestion situation changes. Figure 4.8 illustrates the throughput characteristics on a single connection between end hosts C and D in Fig. 4.1, with an interruption by a UDP flow of 200[Mbps] starting at 200[s] and lasting 100[s], which causes congestion.

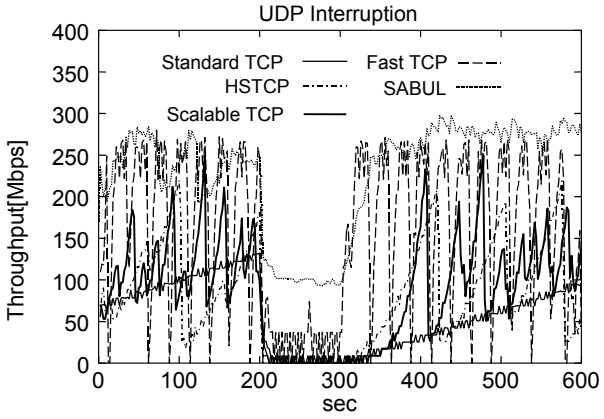


Figure 4.8: Influence of Interrupting UDP Flow

As shown in this figure, when there is no co-existing UDP flow on the path, TCP-based protocols repeatedly increase and decrease their throughput, while the SABUL flow could sustain high-level throughput from the beginning of the flow. After a UDP flow begins data transfer at a rate of 200[Mbps], the Standard TCP, HSTCP and Scalable TCP flows rapidly reduce their throughput to nearly zero, and the throughput remains at the same level until the UDP flow stops its transfer. The SABUL and the FAST flows also reduce their throughput due to the influence of the UDP flow, but they can achieve certain level of throughput even in this case. Especially the SABUL flow can sustain a throughput of approximately 100[Mbps] when sharing the path with the UDP flow. In our experimental environment, the achievable throughput is 305[Mbps] so that we can see that the SABUL flow can almost fully utilize the bandwidth that is unused by the UDP flow.

After the UDP flow stops transmitting, Standard TCP, HSTCP and Scalable TCP take a large amount of time until recovering their throughput. On the contrary, the throughput of SABUL and FAST flows can recover rapidly to the level attained previously.

Table 4.7 shows the average packet loss rate observed in the UDP flow. In this scenario, although the observed packet loss rate in the UDP flow co-existing with FAST or SABUL flow is larger than that with other protocol flows, we find that FAST and SABUL flow can adjust their sending rate in response to the changing network condition. This must be due to the flexible and the explicit rate-control of each of those protocols.

Table 4.7: Packet Loss Rate in the UDP Flows

Protocol	Packet Loss Rate[%]
Standard TCP	0.87
HSTCP	0.84
Scalable TCP	0.91
FAST	1.9
SABUL	1.9

In addtion, Fig. 4.9 illustrates the throughput characteristics of establishing two SABUL flows simultaneously under the same scenario in Fig. 4.8. The throughput performance of

a single SABUL flow is shown there just for comparison. We can observe that the SABUL flows can perform well, i.e., they can make good use of the network resources which is unused by the UDP flow. But in the recovery phase after the UDP flow stops transmitting, the single flow outperforms the multiple flows in their throughput characteristics.

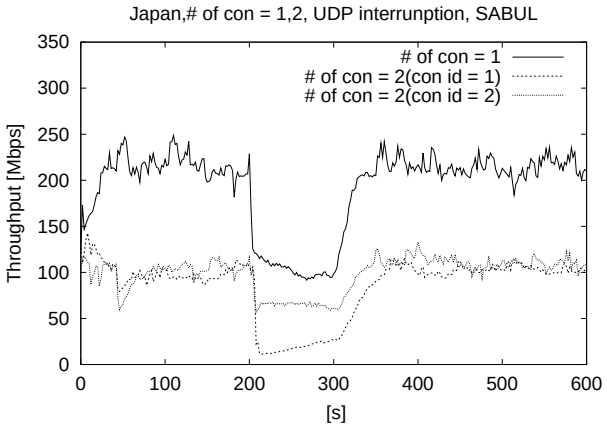


Figure 4.9: Throughput Characteristics on Multiple SABUL Flows with UDP Interruption

4.4.4 Inter-Protocol Throughput Characteristics

Currently, diverse TCP variants and their different implementations are used on the Internet, and traffic of the upcoming high-speed transport protocols will thus encounter that of other transport protocols on the Internet.

In this section, we investigate the throughput characteristics of different protocols sharing the network resources: inter-protocol throughput characteristics.

Figure 4.10 shows the average throughput over 600[s] of two different flows on the link; one is a Standard TCP flow and another is one of the high-speed transport protocols. In Fig. 4.10, one of the flows is that of Standard TCP. Thus, the figure illustrates how the Standard TCP is affected by the high-speed transport protocols. In all the cases, the share of Standard TCP with different protocols deteriorates its performance. In particular, Scalable TCP and FAST heavily deteriorated the performance of Standard TCP in our experiments.

In Fig. 4.11, we also show the average throughput over 600[s] when two of high-speed transport protocols share the network.

From these figures, we can see that Inter-protocol unfairness occurs when different protocols share the network, unlike in Fig. 4.7.

Furthermore, the Standard TCP and HSTCP are particularly affected by other protocols. SABUL is superior to others in terms of achievable total throughput of two flows, and can also attain its own excellent throughput performance in all the cases. The realized fairness issue should be further studied.

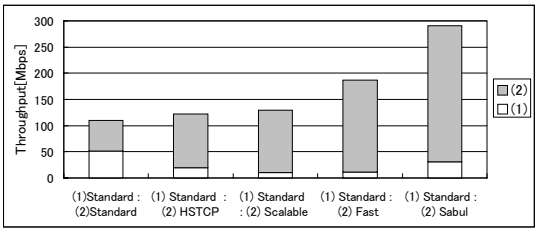


Figure 4.10: Inter-Protocol Average Throughput (Standard TCP - High-Speed Protocol)

4.4.5 Impact of OS at the Receiver on the Performance of TCP-based Protocols

Among the protocols in our experiment, the SABUL protocol must be installed at both sender and receiver sides, and works independently of the OS stack. On the contrary, the TCP-based protocols, i.e., the HSTCP, Scalable TCP and FAST, are installed only at the sender, and they can establish connections with receivers using various TCP stacks implemented on OS. In this section, we discuss the throughput performance of flows on which TCP-based protocols

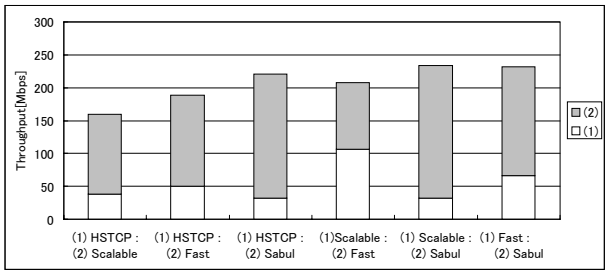


Figure 4.11: Inter-Protocol Average Throughput (High-Speed Protocol - High-Speed Protocol)

send data to receivers with various OSs, e.g., Linux, FreeBSD, Solaris and Windows 2000. First, we carry out preliminary experiments of monitoring TCP SYN segments by *tcpdump* in order to examine the use of TCP options on all the treated OSs. According to the results, Linux, Windows 2000, and Solaris use the following options: (1) notification of MSS, (2) available for TimeStamp Option, (3) SACK Option, and (4) Window Scale Option. We have found that the SACK Option is not used by FreeBSD (ver.4.7 adopted in our environment).

Next we investigate the throughput of flows whose receivers are on various OSs. Table 4.8 shows the average throughput over 200[s] on the flows with receivers working on various kinds of OSs. Also, Fig. 4.12 presents throughput characteristics on flows adopting Standard TCP as a transport protocol. Both of the results clearly indicate that the throughput performance heavily depends on which OS is used at the receiver.

In order to reveal what happened to the data sequence in the flow with the receiver on each OS, we monitored the sequence number of data and ACK packets in the flow via *tcpdump*, when the Standard TCP is adopted as a transport protocol for example.

Figure 4.13 shows the cumulative distribution of interval of ACK packets returned by the receiver with various OSs. From this figure, the interval of generated ACK packets varies

Table 4.8: Average Throughput

receiver side OS	Standard TCP	HSTCP	Scalable TCP	FAST TCP
Linux	62	121	111	186
FreeBSD	151	62	46	50
Solaris	99	136	169	157
Windows2000	115	122	107	136

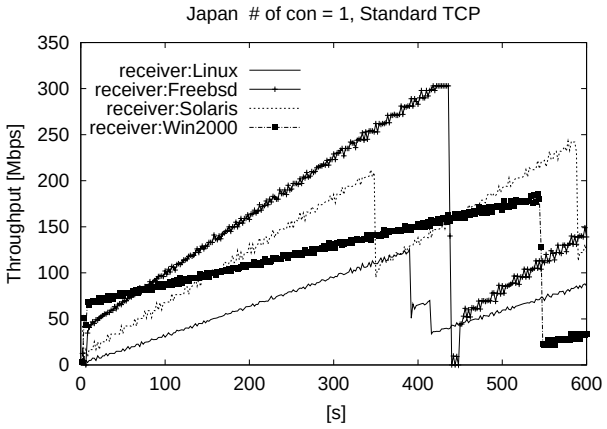


Figure 4.12: Throughput Characteristics in the Standard TCP Flow (Linux to Various OSs)

depending on the kind of OS used at the receiver; i.e., Windows 2000 and Linux send back one ACK packet per a chunk of MSSs (up to 14 MSSs), whereas Solaris and FreeBSD send one ACK packet per one or two MSSs.

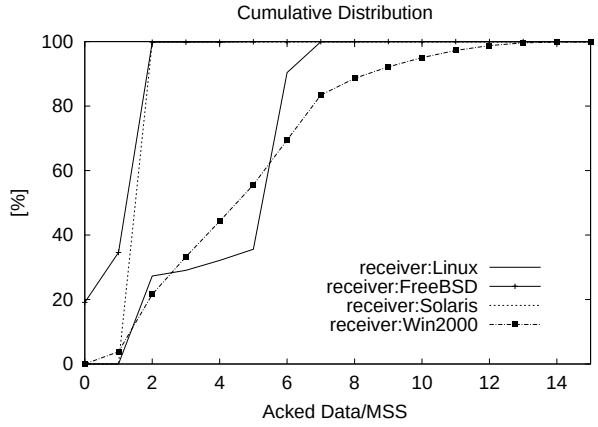


Figure 4.13: Cumulative Distribution of ACK Packet Interval

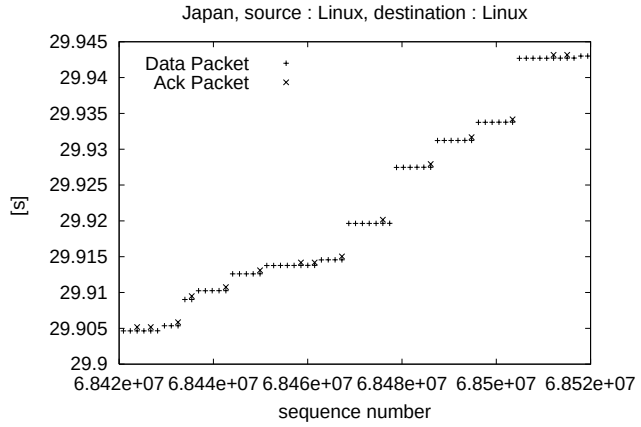


Figure 4.14: Interval of the ACK Packets (Linux to Linux)

Figure 4.14 and 4.15 show when data packets arrive and their ACK packets are sent out at the receivers on Linux and FreeBSD during congestion avoidance phase, respectively. In Fig. 4.14, the receiver on Linux OS sends out one ACK packet just after receiving a chunk of data packets. However, the length of the chunk is not constant and may vary up to seven

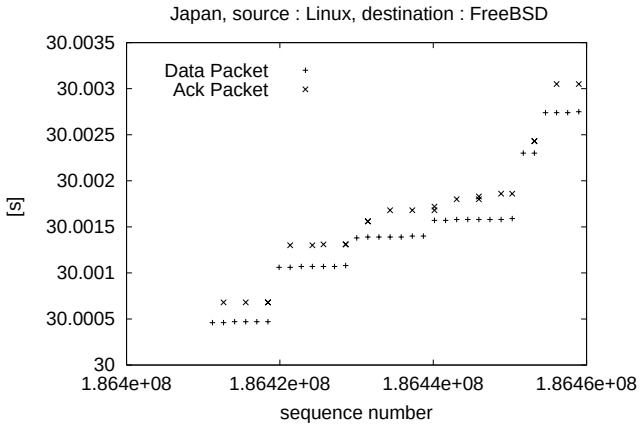


Figure 4.15: Interval of the ACK Packets (Linux to FreeBSD)

MSSs. In Fig. 4.15, the receiver on FreeBSD OS basically sends back one ACK packet per two data packets, though it sometimes sends out two ACK packets simultaneously. This event is counted as an event of ”Acked data is zero” in Fig. 4.13, which happens in 20 percent of all ACK packets.

Standard TCP, which uses ACK packets for both congestion control and error control, increases its *cwnd* every time an ACK packet arrives. The increase of the throughput thus depends on the receiving rate of the ACK packets in number. Therefore, the flow with the receiver on FreeBSD OS, which sometimes sends an ACK packet with no new data packet arrival, increases its *cwnd* size rapidly. As a consequence, the flows with the receiver on FreeBSD OS can increase their throughput quickly as observed in Fig. 4.12.

On the other hand, the receiver on Linux or Windows 2000 sending out an ACK packet for a chunk of data packets limits the chance of increase of *cwnd* at the sender-side during the congestion avoidance phase, and thus, prevents the throughput from being increased quickly, as seen in Fig. 4.12.

In [All03], the modification of *cwnd* increment based on the amount of acknowledged data, rather than the number of ACK packets, has been proposed in order to improve throughput performance. The above-mentioned results indicate, however, the sender-side TCP on

Linux does not implement that modification.

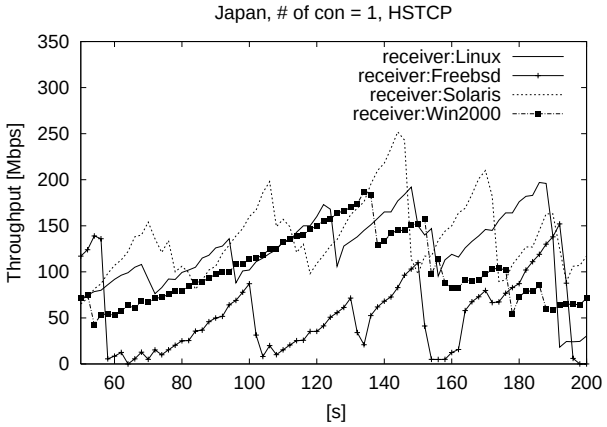


Figure 4.16: Throughput Characteristics in the HSTCP Flow (Linux to Various OSs)

Figure 4.16 presents the throughput characteristics on flows adopting HSTCP with the receivers on various OSs. Similarly to the case of Standard TCP (Fig. 4.12), while the flow with the receiver on Solaris exhibits a rapid increase of the throughput, the flow with the receiver on Linux or Windows 2000 exhibits a relatively slow increase of the throughput. Thus, the case of Solaris achieves the highest throughput. On the other hand, the throughput of the flow with the receiver on FreeBSD (no SACK option) rapidly increases but often falls down to zero, and thus is considerably low on an average compared with other protocols mainly due to the lack of the SACK option. The experiment on Scalable TCP also showed a similar tendency.

Contrarily, the flow of FAST exhibits different throughput characteristics from those of HSTCP and Scalable TCP. There is no significant difference in the degree of throughput increment among the receiver OSs, because FAST updates *cwnd* based on the measured RTT that does not depend on the strategy of sending ACK packets at the receiver-side. In details, however, the case of the receiver on Windows 2000 achieves the highest peak throughput, while the case of receiver on Linux achieves the highest averaged throughput. The case of the receiver on FreeBSD cannot get high throughput.

From these results, it can be concluded that the throughput characteristics of TCP-based high performance protocols depend on the TCP implementation on the receiver-side OS in general. In particular, protocols employing ACK-based congestion control (e.g., HSTCP and Scalable TCP) are strongly affected by the strategy of sending ACK packets at the receiver-side. Note that if those protocols adopt the modification of *cwnd* increment by [All03], this dependency is expected to be reduced. The strong need of the SACK option in the high performance protocols is also indicated.

4.5 Concluding Remarks

In this chapter, we investigated the throughput characteristics of a variety of high-speed transport protocols (HSTCP, Scalable TCP, FAST, and SABUL) recently proposed for transmitting a huge amount of data on fast long-distance networks, through experiments on the Japan Gigabit Network.

All treated protocols in our experiments exhibited good performance in throughput compared with the Standard TCP. They were listed in order of their achieving throughput as follows : SABUL > FAST > Scalable TCP > HSTCP, while the packet losses were more likely to occur in the following order : SABUL > Scalable TCP > FAST > HSTCP. SABUL and FAST also showed that they could adjust their sending rate rapidly and appropriately in response to the dynamic changes of competitive UDP traffic. For TCP-based protocols, the performance was considerably affected by the kind of receiver-side OS due to the difference in strategy of sending back ACK packets and ability of the SACK option.

The fact that SABUL and FAST outperform implies that the conventional congestion avoidance mechanism of TCP based on receiving ACK packets may not be suitable for fast long-distance networks. For high-speed transport protocols, it seems desirable to separate the congestion control from the error control, and further to make the rate control smooth and rapid.

However, all treated protocols suffered from the problem of unfairness or undesirable

interference when multiple connections of different protocols share a common link. Further progress in high-speed transport protocols is needed to solve this problem.

Chapter 5

Experiments for High-Speed Transport Protocol of Multiple Flows

5.1 Introduction

In response to the emerging requirements for high-throughput data transfer on fast long-distance networks in distributed data processing and data sharing environments such as Grid, a variety of high-speed transport protocols have been proposed. Among them are two practical end-to-end approaches: modification of the congestion control mechanism in TCP at the sender-side, and implementation of new transport protocols over UDP. Their performances (throughput characteristics) in dedicated and/or homogeneous network environments have been studied in various experiments on worldwide testbeds (e.g., [CAK+05]). The bandwidth of the Internet, on the other hand, has been increasing in both access networks and core networks. In Japan, for example, reasonably priced 1-Gbps broadband access (FTTH) services have recently become available and 10-Gbps services might be available soon. Internet users, including large application service providers (ASPs), may thus want to use high-speed transport protocols on the Internet to transfer a larger amount of data, regardless of the intentions of the original developers of those protocols. Little attention, however, has

been paid to the problems involved when those transport protocols are deployed on shared and heterogeneous network environments like the global Internet.

We therefore started investigating several promising high-speed transport protocols — HighSpeed TCP (HSTCP) [Flo03], Scalable TCP [Kel03], FAST [JDS04], CUBIC [RX05], HTCP [SL04], and UDT [GG05] — in experiments using several testbed environments such as the Japan Gigabit Network II (JGNII) [1] and the TransPAC2 [2], both of which are open 10Gbps-class Ethernet-based network between US and Japan/Asia-Pacific. We focus on examining what happens under realistic conditions, such as those with change of network routes, i.e. propagation delay, with a variety of receiver-side OSs (Linux, FreeBSD, and Windows XP), coexisting flows with different RTTs and different protocols, coexisting short-lived TCP flows (e.g., web browsing flows), and coexisting constant bit-rate UDP flows (e.g., video stream flows)

This chapter is organized as follows. Section 5.2 explains the configuration of the experimental environments, Sections 5.3 and 5.4 present the experimental results, and Section 5.5 makes some closing remarks.

5.2 Configuration of our experiments

Figure 5.1 summarizes our view of the locations of bottleneck links for TCP flows in the Internet. Cases 1 and 2 show the typical cases in which the bottleneck link is in an edge router on the sender side, while Case 3 shows an example of a bottleneck link in an edge router on the receiver side. This paper reports experimental results obtained in the Case 1 configuration: a 1-Gbps bottleneck link in an edge router on the sender side.

In our experiment we used the JGNII and TransPAC2 as the backbone network shown in Fig. 5.1. Both the JGNII and TransPAC2 provide a maximum bandwidth of 10 Gbps. The information of traffic load on both networks are continuously available at the website of the

[1] http://www.jgn.nict.go.jp/e/index.html

[2] http://www.transpac.org

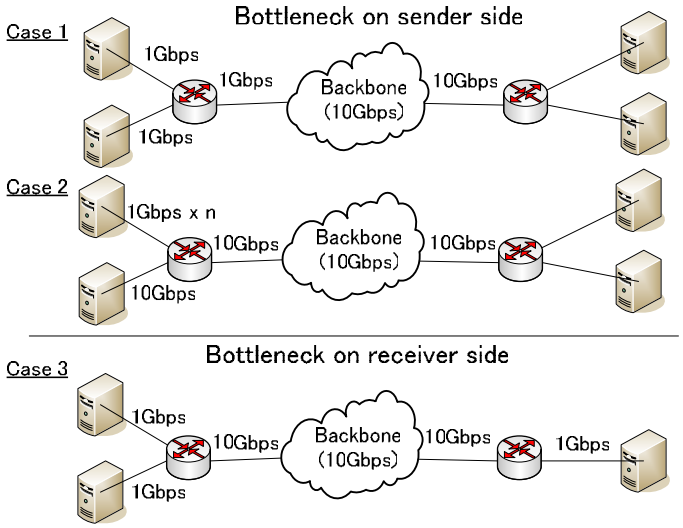


Figure 5.1: Locations of bottleneck links for TCP flows in the Internet.

APAN Operations Center[3]. The end-to-end packet loss rates over these international lines are very low in usual.

Figure 5.2 shows the network configurations in our experiments: (a) the network emulator path (emulated by a Hurricane II from PacketStorm), (b) the network emulator and the JGNII domestic loopback path (Kitakyushu-Tokyo-Kitakyushu), (c) the network emulator or the JGNII domestic loopback path, and (d) the JGNII/ TransPAC2 path between the US and Japan (Chicago - Tokyo - Kitakyushu/ Chicago - Los Angeles - Tokyo - Kitakyushu). The measured RTT of JGNII domestic line, the JGNII international line and the TransPAC2 are 39[ms], 180[ms] and 189[ms], respectively. The number of intermediate nodes (routers) on the JGNII domestic line, JGNII international line and TransPAC2 are 3, 4 and 11, respectively. Using the network emulator, we can configure RTT at values ranging from 0.1 to 10000[ms] and change the bandwidth at values ranging from 0 to 1000 [Mbps].

[3]http://www.jp.apan.net/NOC/

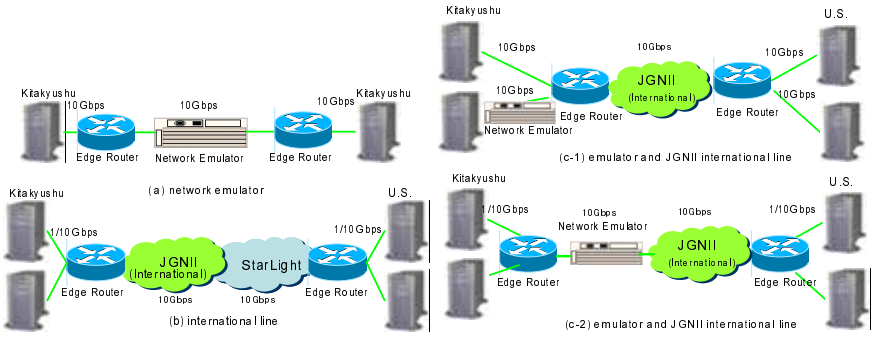


Figure 5.2: Network configurations.

The sender and receiver equipment specifications are listed in Table 5.1. We used Linux (RedHat 9.0 Kernel 2.4.20) as the sender-side OS in our experiments mainly because the function codes for the protocols we used, except for UDT, were provided as patches for the source code of Linux. We tuned various parameters (e.g., Linux txqueuelen, system memory, and RxDescriptors of the NIC(e1000) driver) according to the technical information provided in the web pages of each of the targeted protocols.

We usually used the default output buffer size of the edge routers but also used a larger buffer size in order to investigate the effect of the buffer size of the edge router at the sender side, that is a bottleneck in Case 1 shown in Fig. 5.1. It is known that the performance of high-speed transport protocol was considerably affected by the buffer size of the bottleneck link (router). The edge router we used has a distributed packet-buffering architecture, which has several internal output buffers, and the size of one of them is configurable. We therefore show the experimental results obtained when the buffer size was the default size (indicated as Buf=Small) and when the size was as large as possible (indicated as Buf=Large). Note that, a recent study ([Hir06]) reported a single TCP flow did not work well with the small buffer sizes less than 200 packets in their configuration where the bottleneck bandwidth is limited to 800[Mbps] using dummynet. In our experiments, the small (default) buffer size is less then 200 packets and the large buffer size is more than 200 packets.

In the following sections, we targeted six high-speed transport protocols and used the im-

plementations listed in Table 5.2. We used Iperf to measure the throughput for TCP flows and the packet loss and delay jitter for UDP flows. We usually showed the throughput characteristics as time series of the one-second-averaged throughput of individual flows. In addition, if we observed similar tendencies on two or more protocols (e.g., HSTCP and Scalable TCP), we reported the result for only one of these protocols.

Note that the international and domestic testbed networks we used generally accommodate other constant or temporary traffic. To avoid the influence of unexpected large background traffic, we monitored the traffic load of the network while conducted several trials of each experiment with the same configuration and then discarded irregular cases. Unless otherwise noted, the results reported here were for cases exhibiting relatively good throughput performance.

Table 5.1: Equipment specifications

	End host in Chicago	End host in Kitakyushu
OS	Debian Linux	Red Hat Linux 9.0 Kernel 2.4.20
CPU	Xeons 2.4 GHz/opteron	Xeons 3.2 GHz
Memory	1 GByte	2 GBytes
PCI bus	64 bits	
NIC (1 Gbps)	Intel Pro(e1000)	

Table 5.2: Targeted protocols

Protocol	Version	Protocol	Version
HSTCP	patch for Linux 2.4.19	CUBIC	patch for Linux 2.4.25
Scalable TCP	patch for Linux 2.4.19	HTCP	patch for Linux 2.4.20
FAST	patch for Linux 2.4.20	UDT	version 2.0

5.3 Experimental results for a single data flow

This section shows the fundamental characteristics of each of the high-speed transport protocols revealed by examining the throughput performance of a single connection under various conditions that is likely to be met in the global Internet. In fact, in the daily use of the Internet, an optimal tune of the socket buffer size at an end-host for a specific situation is not easy; the buffer size of a bottleneck router cannot be changed; the receiver-side OS is not always Linux; and the end-to-end route likely changes by some reasons.

5.3.1 Buffer sizes in end-hosts and in bottleneck routers

First, we investigate the effect of the socket buffer size in end-hosts, which is directly related with the upper-limit of the congestion window size (cwnd) of TCP, on the throughput characteristics of the targeted high-speed transport protocol flows. In the international line, the maximum cwnd required to theoretically achieve 1 [Gbps] (actually, payload bandwidth is 941[Mbps]) throughput with RTT of 189 [msec] is approximate 22 [MB], that is the bandwidth delay product (BDP) of the end-to-end path. Figure 5.3(a) shows the long-term averaged throughput of a single flow over the international line configured as illustrated in Fig. 5.2(d) with the default (small) buffer in the edge routers, which indicates that all TCP-based high-speed protocols except for CUBIC are sensitive to the socket buffer size around the BDP and damaged by an inadequately large socket buffer. Figure 5.4(a) shows the throughput characteristics of a FAST flow with different socket buffer sizes from 80% to 106% of BDP using the international line with the small buffer in the edge routers, and Fig. 5.4(b) shows the throughput characteristics of HTCP and Scalable TCP flows with socket buffer sizes set to 95% and 100% of BDP in the same configuration.

From these figures, it is clearly observed that the maximum throughput is limited by the socket buffer size but seems stable when the socket buffer size is smaller than the value of BDP. On the other hand, when the socket buffer size is more than or equal to the BDP, the

throughput characteristics may become unstable depending on the protocols.

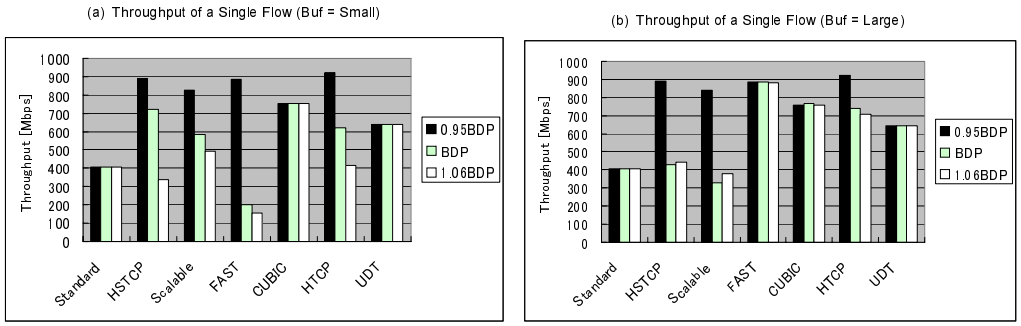


Figure 5.3: Influence of the socket buffer size on the throughput characteristics

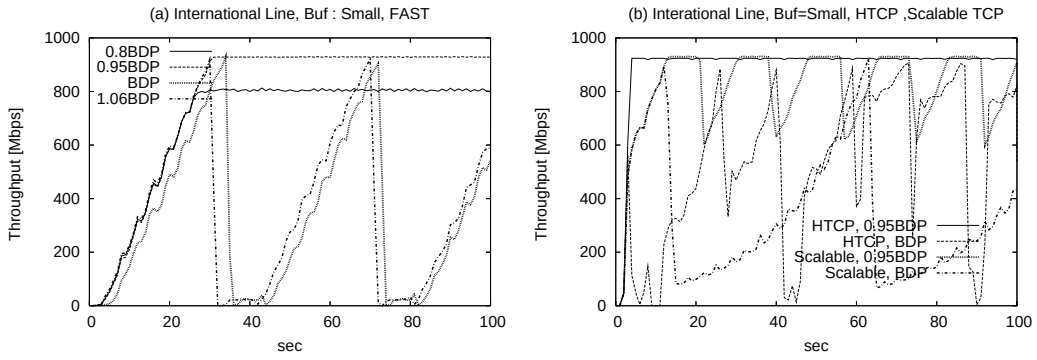


Figure 5.4: Effect of the buffer size at the end hosts, international line (a) FAST and (b) HTCP and Scalable TCP.

Therefore, in our experiments, we investigated the throughput characteristics of each of the high-speed transport protocols adopting various socket buffer sizes, e.g., 0.95 BDP, BDP, and 1.06 BDP.

Next, we investigate the effect of the output buffer size in bottleneck edge routers on the throughput characteristics of the targeted flows. Figure 5.3(b) shows the long-term averaged throughput of a single flow over the international line configured as illustrated in Fig. 5.2(d) with the large buffer in the edge routers. The adverse effects of too large socket buffers (in the end-hosts) are considerably mitigated by using a large edge router buffer for most of the

targeted protocols, although HSTCP and Scalable TCP are still damaged by the too large socket buffers. Figures 5.5 (a) and (b) show the throughput characteristics of a single flow of some targeted protocols (Scalable TCP, FAST, and CUBIC) with 1.06BDP socket buffer over the international line using the small buffer and the large buffer at the edge routers, respectively.

From these figures, when using a large socket buffer (1.06 BDP) in end-hosts, it can be seen that the throughput of the FAST flow is greatly improved by setting the output buffer size of the edge router large, while that of the Scalable TCP flow is compromised by setting so. In addition, the throughput of the CUBIC flow is not so affected by the edge router buffer size. For Scalable TCP (and HSTCP), the above preference for a smaller buffer in the edge routers in case with a large socket buffer may comes from its nature of rapid growth of cwnd. That is, a large socket buffer in combined with a large edge router buffer may cause too many successive packet losses in intermediate routers.

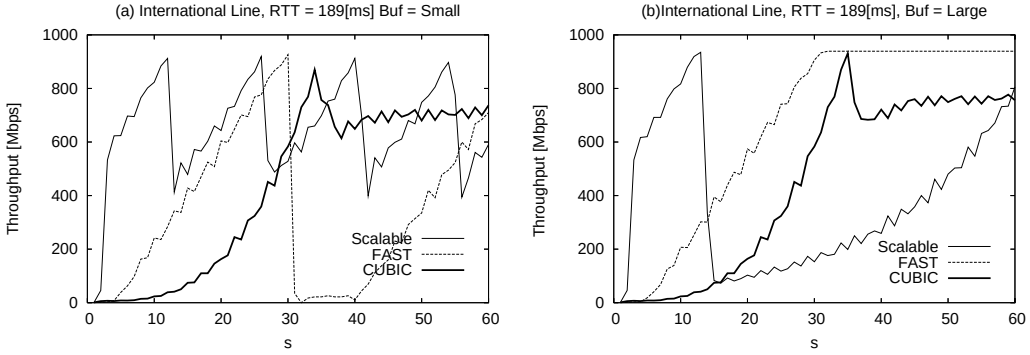


Figure 5.5: International Line single-flow throughputs obtained with (a) small and (b) large buffers at an ingress edge router,socket buffer size = 1.06BDP.

Before ending this subsection, we address the use of real testbed networks and the network emulator by comparing the experimental results of the throughput characteristics of a single flow in those different network environments. Figures 5.6 (a) and (b) show the throughput characteristics of a single flow of some targeted protocols with 1.06BDP socket

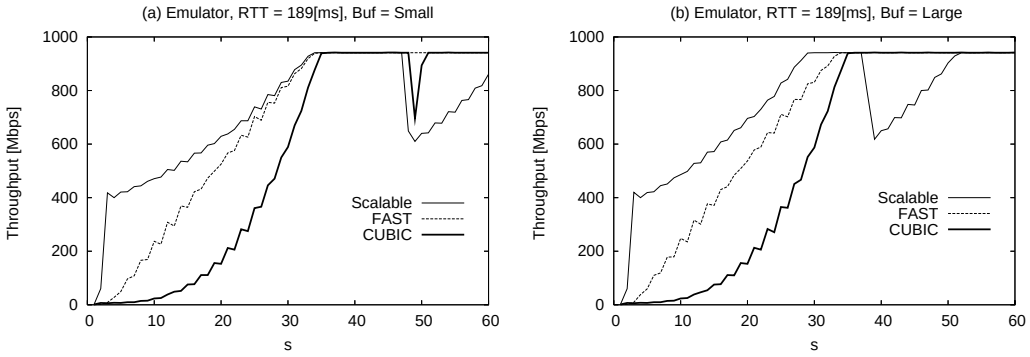


Figure 5.6: Emulator single-flow throughputs obtained with (a) small and (b) large buffers at an ingress edge router,socket buffer size = 1.06BDP.

buffer over the network emulator configured as illustrated in Fig. 5.2 (a) using the small buffer and the large buffer at the edge routers, respectively. The RTT of this configuration is approximate to that of the real testbed over international line. Comparing these figures with Fig. 5.5 (the international line scenario), it can be observed that the throughput characteristics for all protocols in the network emulator were somewhat similar to, but much more stable (and much less sensitive to the edge router buffer size) than, those in the real testbed networks. This is because packet losses do not occur in the network emulator unless a packet loss rate is set in the emulator. Thus, we then examine the throughput characteristics using the network emulator with setting the packet loss rate to several values.

Since buffer-size-dependent differences were not significantly observed in their throughput characteristics of CUBIC and HTCP flows in our experiment, in Fig. 5.8 we show their throughput characteristics using the network emulator with setting the packet loss rates to some values. In Fig. 5.8(a), we see that the behaviors of CUBIC flows observed in the international line and in the network emulator with a small loss rate are very similar during an initial increase of the throughputs but their behaviors slightly differ in ways depending on the packet loss rate once the maximum throughput has been reached. On the other hand, the HTCP flow throughput characteristics observed in the network emulator and in the interna-

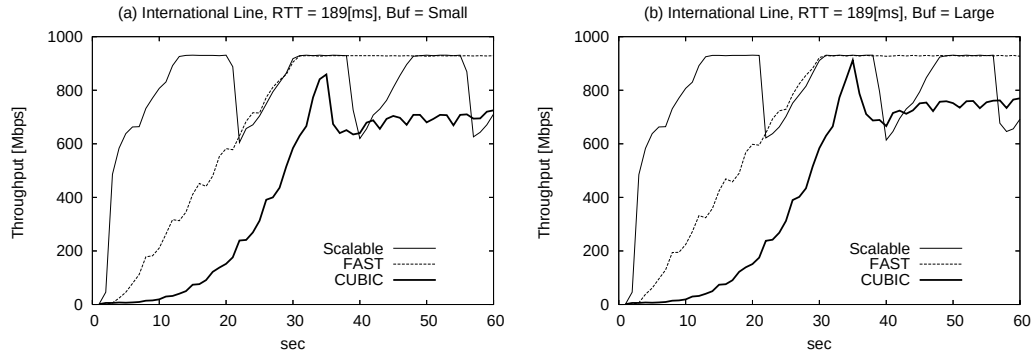


Figure 5.7: International line single-flow throughputs obtained with (a) small and (b) large buffers at an ingress edge router , socket buffer size = 0.95BDP.

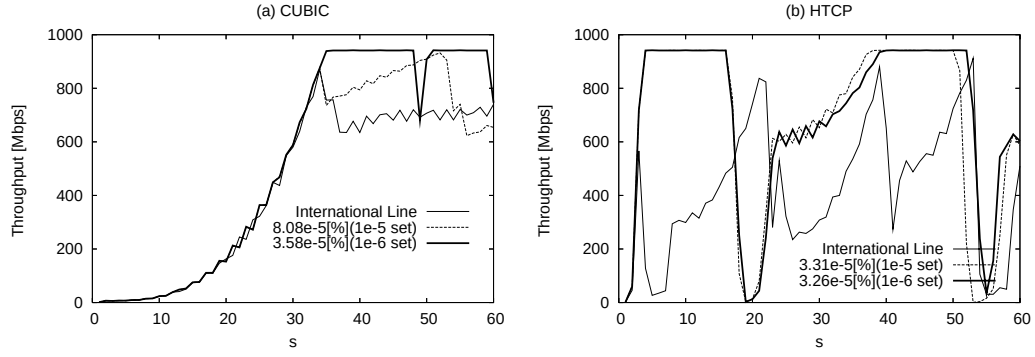


Figure 5.8: Emulator single-flow throughputs for (a) CUBIC and (b) HTCP protocols (packet loss rate set by emulator).

tional line are very different as shown in Fig. 5.8(b).

From these results, we ran our experiments in both environments when that was possible, and used the network emulator to investigate the basic characteristics and the JGNII/TransPAC2 lines to investigate characteristics under more realistic environments, respectively.

5.3.2 Variety of receiver-side OSs

The TCP-based high-speed transport protocols are installed only on the sender side and can established connections with receivers using various TCP stacks implemented on the different OS. This subsection shows the performance measured when the FreeBSD (ver.5.3) and Windows XP (SP2), which are commonly used in the Internet, were used as receiver side OSs in the network configurations shown in Fig. 5.2(c). We first tuned several performance related parameters of each OS according to the technical information in [tun].

We carried out preliminary experiments of monitoring TCP SYN segments by tcpdump in order to examine the use of TCP options on all the treated OSs. And we confirmed that all targeted protocols used the following options: (1) notification of MSS, (2) available for TCP Stamp Options, (3) SACK options, and (4) Window Scale Option.

Figures 5.9 (a) and (b) show the throughput characteristics of a single flow on the network emulator in cases that FreeBSD and Windows XP are used as the receiver-side OS, respectively. In both cases, we set the RTT to 189 ms and the socket buffer size to 0.95BDP. Basically, we observed that all protocol flows could achieve high throughput regardless of the kinds of receiver side OS in the environment where no packet losses occurred, although the detailed throughput characteristics of individual protocols vary depending on the receiver-side OS. For example, the throughput in the case with an XP receiver increases more slowly than do the throughputs in the cases with Linux and FreeBSD receivers.

To find out what happened to the data sequence in the flow when each OS was used on the receiver side, we monitored the sequence number of data and ACK packets in the

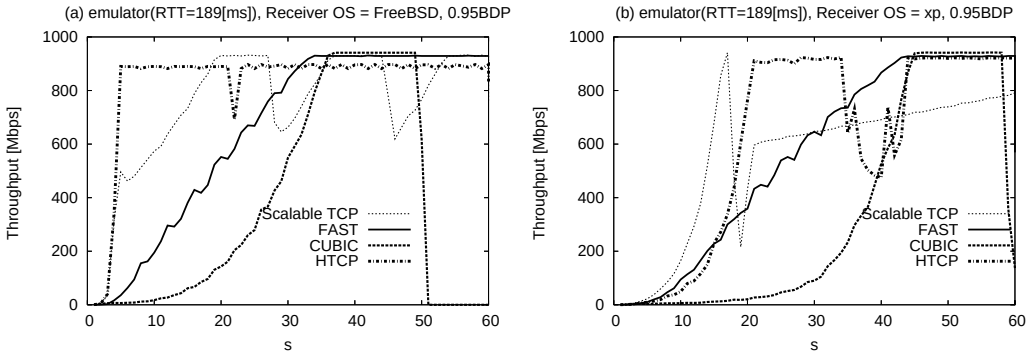


Figure 5.9: Throughput characteristics observed in (a) emulator(RTT=189 ms) with FreeBSD ver.5.3 receiver, socket buffer = 0.95BDP, (b) emulator (RTT=189 ms) with Windows XP receiver, socket buffer = 0.95BDP.

flow via tcpdump when the Scalable TCP was used as the transport protocol. Figure 5.10(a) shows the cumulative distribution of interval of ACK packets returned by the receiver with various OSs, which indicates the interval of generated ACK packets varies depending on the receiver-side OS. Figures 5.10(b), 5.10(c), and 5.10(d) respectively show when data packets arrive and their ACK packets are sent out during congestion avoidance at receivers running on Linux, FreeBSD, and Windows XP.

Figure 5.10(b) indicates that a Linux receiver sends out an ACK packet after receiving a chunk of data packets but the length of the chunk is not constant. Figure 5.10(c) indicates that a FreeBSD receiver basically sends back one ACK packet per two data packets but sometimes sends out two ACK packets simultaneously. Figure 5.10(d) that a XP receiver sends out an ACK packet after receiving a long chunk of data packets.

For example, an OS-dependent difference in CUBIC flows observed by comparing Fig. 5.9(a) (where the receiver-side OS is FreeBSD) and Fig. 5.9(b) (where the receiver-side OS is XP) can be explained as follows. A TCP flow, which uses ACK packets for both congestion control and error control, increases its cwnd every time an ACK packet arrives. Therefore the throughput of the flow communicating to a FreeBSD receiver, which sometimes sends

an ACK packet when no new data packet arrives, increases rapidly. On the other hand, the throughput is not increased so quickly when a receiver is running on Windows XP because a long chunk of data packets corresponding to an ACK packet for a chunk of data packets during the congestion avoidance phase limits the chance of increasing the cwnd on the sender-side.

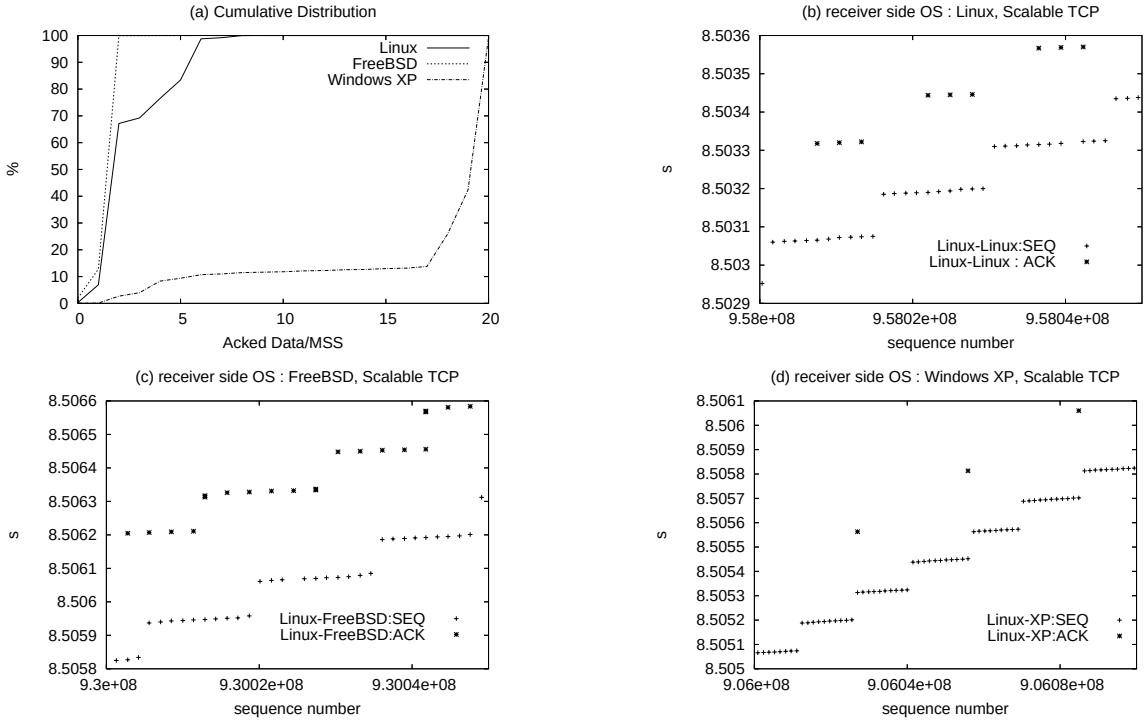


Figure 5.10: Interval of ACK packets: (a) cumulative distribution, (b) Linux to Linux, (c) Linux to FreeBSD, (d) Linux to XP.

Figures 5.11(a) and (b) respectively show the throughput characteristics of a single flow for FreeBSD and Windows XP as the receiver-side OS with setting the socket buffer size to 0.95BDP on the JGNII domestic line in which packet losses likely occur, while Figs. 5.11(c) and (d) are those with setting the socket buffer size to 1.06BDP. When the receiver side OS is FreeBSD, ACK packets were sent out from the receiver to the sender in the bursty, by which the sender could increase its cwnd rapidly generating packets got bursty. As a result,

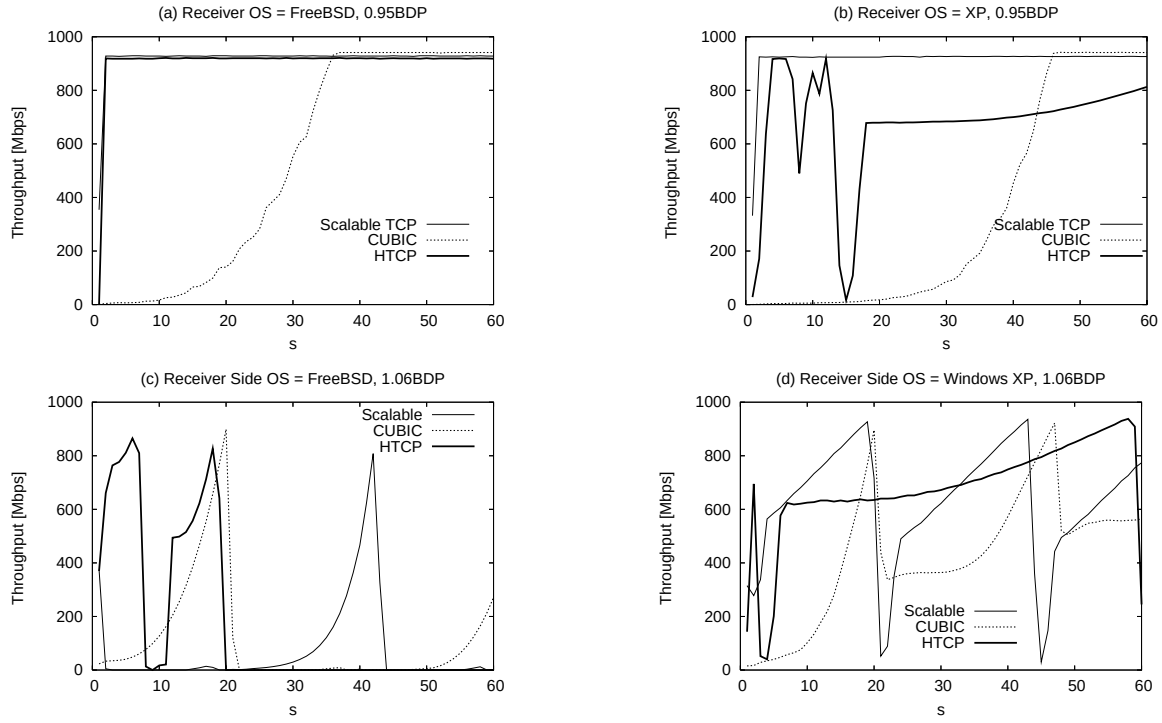


Figure 5.11: Throughput characteristics of JGNII domestic line when (a) FreeBSD or (b) Windows XP(SP2) is the OS on the receiver side with socket buffer size are 0.95BDP, (c) and (d) is that for FreeBSD and XP with socket buffer size are 1.06BDP.

packet losses increased and throughput characteristic was degraded. In case of the receiver side OS is XP, on the other hand, since XP does not send ACK packets so frequently, the degradation observed in case of FreeBSD does not occur. At least, Figs. 5.11(b) and (d) imply that users who install the most popular OS (Windows XP) and tune the performance-related parameters accordingly can enjoy high-throughput reception if the sender uses high-speed transport protocols.

5.3.3 Rapid change of propagation delay

Route changes will be likely in the Internet because of operational errors and the need for global traffic engineering. We therefore investigated how rapid changes of RTT affect the throughput characteristics of each of the high-speed transport protocols by path switching between the emulator-path (very stable) and the JGNII domestic path (somewhat unstable) in the network configuration shown in Fig. 5.2(c). Note that the socket buffer size was set to the maximum bandwidth-delay product – i.e., the longer RTT (80 ms) $\times$ the bandwidth (941 Mbps).

Figure 5.12(a) shows the throughput behavior of a single flow in case that the path was switched from the original path with as 80-ms RTT (the emulator path) to an alternative path with an 39-ms RTT (JGNII domestic path) 30 seconds after the start of the flow and was then switched back to the original one 60 seconds later. At the moment of the change from the original path to the one with a shorter RTT, the throughput of all protocols became unstable. This change likely makes the cwnd the sender too large and also causes packet misordering at the receiver, both of which can lead to bursty packet losses. On the other hand, at the moment of the change back to the original path with the longer RTT, the throughput of each of the TCP-based protocols decreased by nearly 50% (which might be due to a few packet losses), which is followed by gradually recovering its original high level throughput, while that of UDT seemed insensitive to this change. Note that the FAST flow cannot recover its throughput when the RTT changes, because FAST changes its cwnd size on the basis of

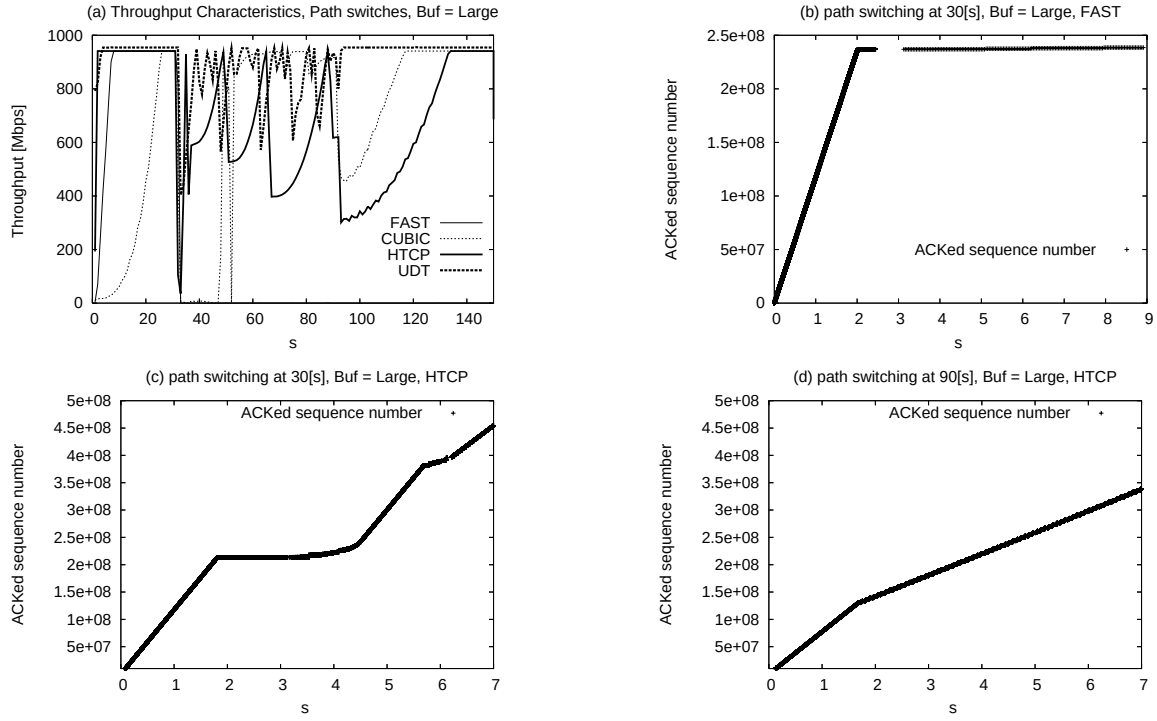


Figure 5.12: Single-flow throughputs in case that the path switches

delay information.

To confirm the above observation, we investigated the acknowledged sequence number in ACK packets (monitored at sender side by using the tcpdump). Figure 5.12(b) shows the ACKed sequence number when a FAST flow was switched from the original path with a longer RTT to the alternative path with a shorter RTT, in which many duplicated ACKs can be seen successively. Figures 5.12(c) and (d) respectively show the ACKed sequence number observed when a HTCP flow was switched at 30 s and at 90 s. Successive duplicated ACKs are observed at the moment of changing from the path with the RTT to the shorter RTT path, while no duplicated ACKs are observed on returning to the path with the shorter RTT.

5.4 Experimental results for coexisting data flows

This section reports our investigation in cases that various kinds of flows coexist on the bottleneck link : long-lived flows with different round trip times (RTTs) or by different transport protocols, short-lived Standard TCP flows, and constant-bit-rate (CBR) UDP flows. In fact, a high-speed transport flow will run on the Internet by sharing resources with such data transfer flows that traverse different locations and have different protocols including Standard TCP. In addition, a high-speed transport flow should efficiently coexist with other types of Internet application traffic such as short-term web browsing flows and long-term video streaming flows.

5.4.1 Coexisting flows with different RTTs

We investigated the throughput characteristics when coexisting flows have different RTTs. In the configuration shown in Fig. 5.2(b), we started two flows by a same high-speed transport protocol (flow 1 and flow 2) simultaneously and set the ratio of the RTT of flow 2 to that of flow 1 to either 1, 2, or 4 by using the network emulator.

Figure 5.13 shows the time-averaged throughput of these two coexisting flows where

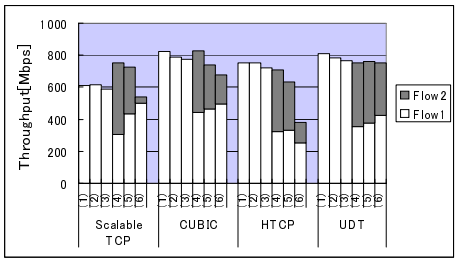


Figure 5.13: Long-term averaged throughputs of flows with different RTTs.

RTT of flow 1 is 39 ms and that of flow 2 is 39(=(4)), 78(=(5)), or 156(=(6)) ms. It also shows for each protocol the throughput of a single flow where the RTT is 39(=(1)), 78(=(2)) and 156(=(3)) ms. Note that similar experiments in which the bottleneck bandwidth was much lower than that in our case have already been reported [LLS].

We can see that each high-speed protocol is only slightly affected by RTT when a only single flow is established. When the two flows of UDT protocol coexist, the performance is insensitive to RTT. When two flows of a TCP-based protocol with the same RTT coexist((4)), the total achievable throughput is almost the same as or larger than that achieved by a single flow. When two flows of a TCP-based protocol with different RTTs coexist((5) and (6)), however, we observe severe inefficiency as well as unfairness in throughput. For example, in Case (6) where the RTT of flow 2 is four times larger than that of flow 1, the achievable throughput of flow 2 significantly decreases and that of flow 1 increases just a little compared with Case (4) where the RTT of two flows are same.

5.4.2 Coexisting flows with different transport protocols

We examined the scenario in which two long-lived data transfer flows with different protocols coexist on a same bottleneck link. When a high-speed transport protocol flow and a Standard TCP flow simultaneously run, we observed the well-known unfairness problem: that is, a high-speed transport protocol flow starved the long-lived Standard TCP flow for bandwidth, while the performance of the high-speed transport protocol nearly unchanged.

We also performed the simultaneous runs of all the combinations of two high-speed transport protocols in the same path. Each of the bar graphs in Fig. 5.14(a) shows the sum of the average throughputs of two flows over 300 seconds: a flow with the protocol indicated on the X-axis and a flow indicated by an indicator coexist. We found that the sum of the throughputs of different kind of two high-speed transport protocol flows was smaller than that for two coexisting flows with the same high-speed transport protocol. That is, the link utilization degrades when different kinds of high-speed transport protocol flows coexist. Figure 5.14(b) shows average throughput of each flow when a HTCP flow coexists with another high-speed transport protocol flow. It is clearly observed that there are unfairnesses in throughput. The unfairness problem was found in all cases of coexisting different kinds of high-speed transport protocol flows. UDT protocol flow, in particular, significantly affected the performance of coexisting TCP-based protocol flows.

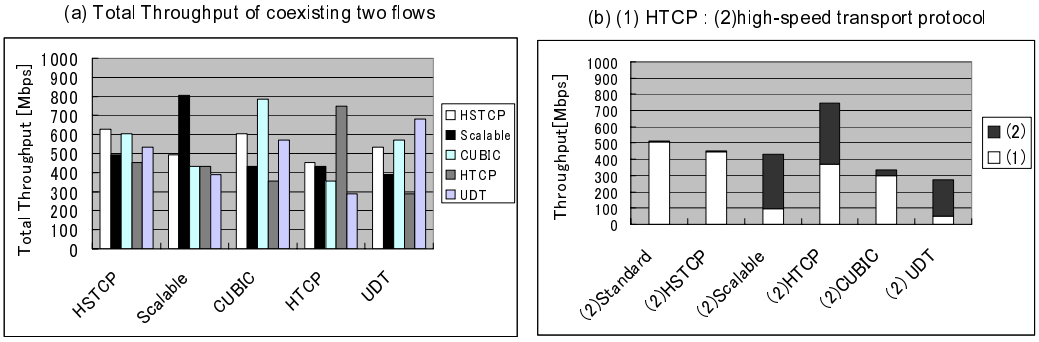


Figure 5.14: Throughput when two kinds of high-speed transport protocol flows coexist: (a) average throughputs for all pairs of protocols, (b) average throughput for each of the protocols when its flow coexists with HTCP flow.

5.4.3 Coexistence of short-lived Standard TCP flows

We examined the throughput performance in case that a high-speed transport protocol flow coexists with a number of short-lived Standard TCP flows. We performed simultaneous runs

of a single long-lived flow using one of the high-speed transport protocols and 3000 short-lived flows using the Standard TCP over the Japan-US international line as illustrated in Fig. 5.2(d). The socket buffer size of high-speed transport protocol flows is set to 0.95BDP. The transferred data size of each short-lived flow followed a Pareto distribution with a shape parameter of 1.3 and a mean of 100, 300, or 500 KB. The starting time of each transfer was randomly selected within a 300-s interval.

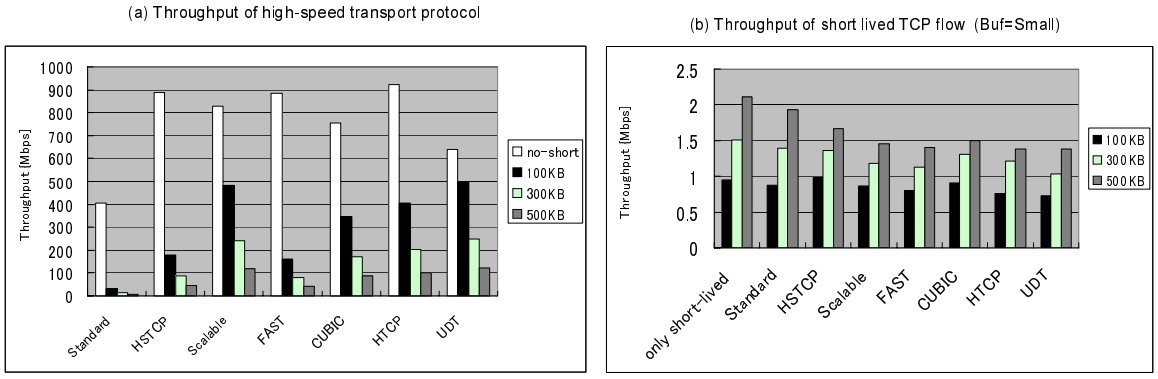


Figure 5.15: Throughput of (a) high-speed transport protocol control flows and (b) coexisting short-lived Standard TCP flows of various sizes when Buf=Small.

Figure 5.15(a) shows the average throughput (over 300 s) of a high-speed transport protocol flow coexisting with short-lived Standard TCP flows when the output buffer size of the edge routers is small. The leftmost-side bar in each group of bar graphs shows the throughput of a single high-speed transport protocol flow running alone on the path. It is clearly observed that the performance of a high-speed transport protocol flow was considerably affected by the coexisting short-lived Standard TCP flows, even through each of these flows was small one. The larger the averaged file size of the short-lived Standard TCP flow was, the larger the observed damage in the high-speed transport protocol flow became.

Figure 5.15(b) shows the averaged throughput of the short-lived Standard TCP flows defined by $(\sum S_i)/(\sum t_i)$, where S_i denotes the file size of flow i and t_i denotes the transfer time of flow i. The larger the average file size, the higher the averaged throughput of short-lived

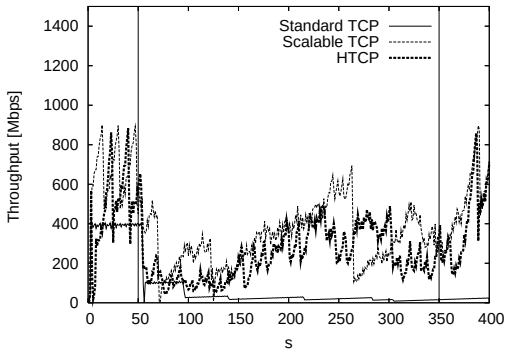


Figure 5.16: Throughput characteristics of high-speed transport protocol flows coexisting with short-lived Standard TCP flows for 50 –350 s.

Standard TCP flows. In cases with the average file size of 300KB and 500 KB, however, the average throughput of short-lived flows coexisting with a high-speed transport protocol flow was smaller than that without a coexisting high-speed transport protocol flow. Figure 5.16 plots the time-series of throughput of the high-speed transport protocol flows when short-lived Standard TCP flows with an average file size of 500 KB were randomly generated for 50-s and 350-s periods.

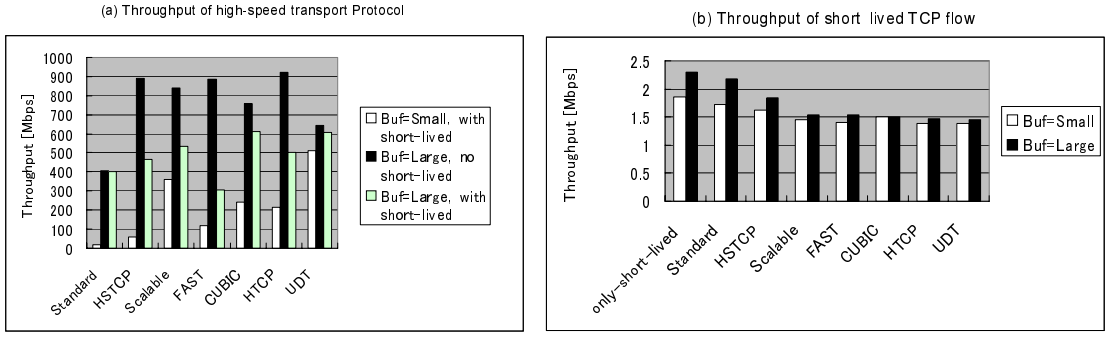


Figure 5.17: Effect of buffer size at edge routers: (a) throughput of high-speed transport protocol flows, (b) throughput of short-lived Standard TCP flows.

Figures 5.17(a) and (b) show the average throughput of a high-speed transport protocol flow and the average throughput of short-lived Standard TCP flows when the average file

size of short-lived Standard TCP flows is 500 KB.

These results indicate that coexisting short-lived Standard TCP flows could degrade the performance of high-speed transport protocol flows considerably although coexisting long-lived Standard TCP flows could not. This may be because the slow-start phases of short-lived standard TCP flows randomly change the available bandwidth. For example, the observed long-term averaged throughput of short-lived TCP flows at the ingress router is 32.6-Mbps when the averaged file size is 500-KB, and observed one second averaged instantaneous load is over 100[Mbps].

The performance of short-lived Standard TCP flows with relatively large file size was also adversely affected by coexisting high-speed transport protocol flows. Setting the buffer size at the bottleneck nodes larger could mitigate the degradation of throughput of both high-speed TCP flows and short-lived Standard TCP flows.

5.4.4 Coexistence of CBR UDP flows

We performed simultaneous runs of a single long-lived flow by one of the high-speed transport protocols and two CBR (constant bit rate) streams by the UDP protocol on the path shown in Fig. 5.2(d). Each stream consisted of 200-byte UDP packets sent at 1.6, 3.2, and 8 Mbps (representing $n\times$ 64-Kbps, where $n = 25, 50, 125$). We examined two scenarios: Case 1, in which a single high-speed transport protocol flow started 30 s after two UDP flows started; and Case 2, in which two UDP flows started 30 s after a single high-speed transport protocol flow started.

Figure 5.18(a) shows the jitter characteristics of two 8-Mbps UDP flows observed in Case 1 when the buffer size at the edge routers was small or large. Comparing the averaged jitter of two CBR flows with a coexisting high-speed transport protocol flow and without that (indicated by “only UDP flows”), it is obvious that the jitter of the CBR flows is slightly affected by the coexisting high-speed flow. Figure 5.18(b) shows the average throughput over 300 s of high-speed transport protocol flows when the buffer size at the edge routers

was small or large. In each grouped of bar graphs, the bar labeled “no UDP” shows the throughput characteristics of the corresponding high-speed transport protocol flow without any coexisting UDP flow. The throughputs of all the high-speed transport protocol flows were affected by coexisting UDP flows when the buffer size at the edge routers was small. We can also see that the adverse influence of UDP flow on the throughput of coexisting high-speed transport protocol flows was smaller when the output buffers were larger. Especially, throughput characteristics of FAST and CUBIC protocol flows were not affected by UDP flows with large output buffers.

Figure 5.19 shows the performance characteristics observed in Case 2. The tendencies of the jitter of UDP flows and the throughputs of the high-speed transport protocol flows were similar to those in Case 1, but the throughput of the Standard TCP flow differed between cases 1 and 2. Since the Standard TCP flow does not increase its window size aggressively during its congestion avoidance phase, it might not be affected by coexisting UDP flows in Case 2. In Case 1, on the other hand, where the UDP started first, the Standard TCP flow competed for bandwidth with UDP flows during its slow starting phase.

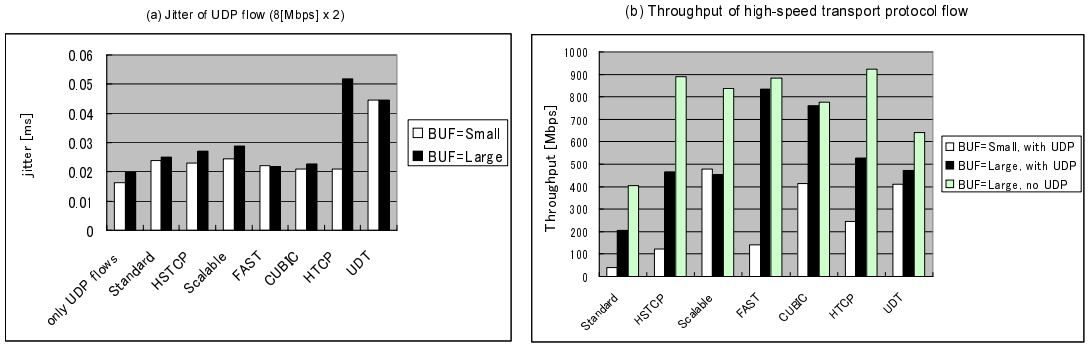


Figure 5.18: (a) Jitter of UDP flow and (b) throughput of high-speed transport protocol flow in Case 1.

Figure 5.20 shows examples of time-series throughput characteristics of a single flow by high-speed transport protocols (HSTCP and FAST) in cases of no UDP flows coexisting, of UDP flows starting beforehand (Case 1), and of UDP flows starting afterward (Case 2) when

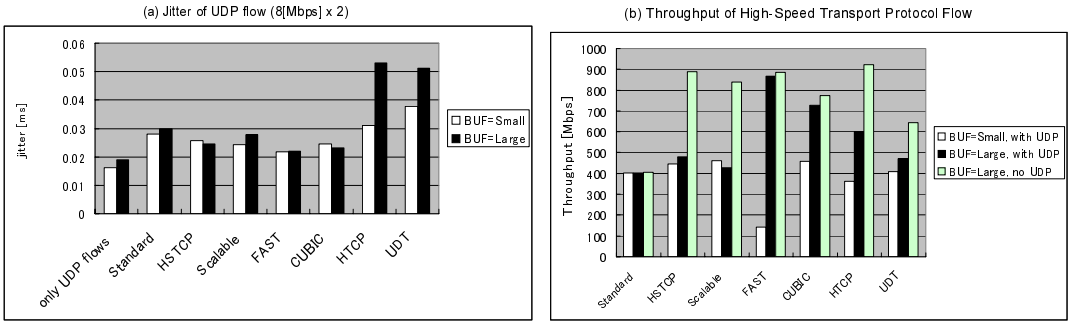


Figure 5.19: (a) Jitter of UDP flow and (b) throughput of high-speed transport protocol flow in Case 2.

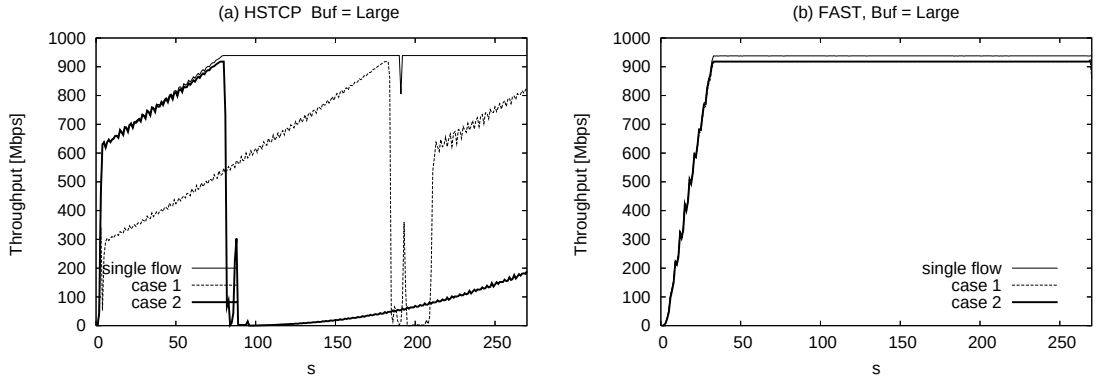


Figure 5.20: Throughput characteristics of high-speed transport protocol flows coexisting with UDP flows.

the buffer size at the edge routers was large.

In these experiments, the throughput of neither high-speed transport protocol flow could increase rapidly during the slow starting phase in Case 1. This may have been due to the packet losses caused by the coexisting UDP flows. In Case 2, although the UDP flows started after the TCP flow entered the congestion avoidance phase, occasional packet losses occurred when the throughput increased to near the maximum bandwidth. This prevented the TCP flow from achieving a good average throughput. When the buffer size is large, FAST and CUBIC flow were not affected by UDP flow in both cases.

Table 5.3 summarized mutual influences observed in the jitter characteristics of UDP flow and in the throughput characteristics of high-speed transport protocol flows when they were coexisting.

Table 5.3: observed characteristics in case of coexisting UDP and high-speed transport protocol flows

	jitter for UDP flows	throughput of high-speed protocol flows
HSTCP	affected	affected
Scalable	affected	affected
FAST	affected	not affected with large buffer at edge router
CUBIC	affected	not affected with large buffer at edge router
HTCP	largely affected	affected
UDT	largely affected	affected

These results demonstrated that the throughput of some high-speed transport protocols are adversely affected by coexisting UDP flows even if the load of the UDP flows is only 16 Mbps. We also found that UDP flows with a lower load (3.2 or 6.4 Mbps) could affect the performance of coexisting high-speed transport protocols when the output buffer size was small.

5.5 Concluding remarks

Through experiments in the various network environments including open 10Gbps-class network testbed between US and Japan, we are investigating what happens when high-speed transport protocols are used in the global Internet. Our results (in cases with 1-Gbps end-hosts) indicated that, when long-lived data transfer flows of high-speed transport protocols run in coexistence with even small amounts of Internet application traffic such as short-term web browsing flows and long-term video streaming flows, the performance of not only the web access and video streaming but also that of the high-speed transfer flows is degraded. In other words, such circumstances are neither effective nor efficient in terms of bandwidth sharing. Our ultimate goal is to find a feasible and cost-effective way for the future Internet comprising shared heterogeneous networks to simultaneously provide high-throughput data transfer as well as various kinds of other applications traffic.

At this moment, while we have not found a silver bullet to this problem, we have been knowing several interesting characteristics to pursue the problem, that is pros and cons of each existing high-speed transport protocol in each typical realistic situation. We expect, by sharing deep insights given from a variety of experimental results in a variety of realistic network conditions, to finally develop some improved high-speed data transport protocols and/or new management mechanisms for the intermediate nodes to meet our goal.

Chapter 6

Quality of Assured Service through Multiple DiffServ Domains

6.1 Introduction

The Intserv framework has been proposed as a technology to provide QoS in the Internet and assure the quality per micro flow. It, however, needs to perform signaling processes before sending data, and thus lacks scalability because of the huge amounts of status information that must be maintained by each router. Therefore it has not really deployed in the real Internet.

Consequently, DiffServ technology has been proposed in the IETF, and has very simple architecture consisting of a marking strategy in ingress routers and packet management based upon those marks of the packets in core routers. Namely, packets are classified into a number of service classes in ingress routers, and are marked with a value to indicate their classes. They are then managed depending on their classes at core routers.

So far, two different types of Per Hop Behavior (PHB) have been proposed: Expedited Forwarding (EF)[KK99] and Assured Forwarding (AF) PHB[HBWW99]. EF PHB provides service like a virtual wire, whereas AF PHB offers the assurance of minimum bandwidth

based on a Service Level Agreement (SLA).

From the results of various studies and evaluations over testbeds, the QoS delivered by the DiffServ framework, especially that offered via AF PHB, is rather poor; i.e., a variety of factors influence its QoS characteristics, for example the size of the packet, the value of RTT(Round Trip Time), the value of SLA, as well as the parameter sets for queue management in nodes[SNP99]. To resolve these issues, many studies have proposed various techniques by mainly focusing on the following three points for modification, (1) the transport protocol at the sender,(2) marking strategies at the marker, and (3) dropping policies at the router[YN98].

In [Tei98], the architectures to attain QoS of flow over the Internet, i.e., over multiple domains are shown, and experiments on them are reported. The QoS WG membership constructed the DiffServ testbed for interdomain, and especially focused on EF service and showed their experimental results. [IEP] is the log of a recent meeting on the IETF of IEPREP BOF, and there were discussions about the importance of a policy mechanism governing the entire Internet (i.e., over multiple domains) in times of crisis and emergency.

In [Her99], two ways to control the connection over multiple domains are compared. One employs a QoS server at each domain, which negotiate with each other for QoS provisioning. In the other, each domain independently negotiates with adjacent domains. The material has shown that a prompt and efficient policy exchange is needed to guarantee the end-to-end QoS over multiple domains, and that a QoS server works well for that.

To implement policy exchange over domains, new protocols are proposed in [ETMK99]. While surveying issues in the AF service category, we have found that [Fang99] is the only study that examined QoS characteristics of flows over multiple DiffServ domains, and most of reports, except for [Fang99], have investigated QoS characteristics within a single DiffServ domain, although actual networks are composed of a large number of domains.

In [Fang99], W. Fang et al. concluded that they could see no distinct differences in throughput characteristics of flows, regardless of whether they went through a single domain

or multiple ones. However, in their model, parameters set up at nodes were different in each domain without any explanation, and the model considered as an intra-domain is a very limited one. Therefore, in this chapter, we will extensively study the throughput characteristics of AF flows over multiple DiffServ domains.

The organization of this chapeter is as follows: Sections 6.2 and 6.3 provide the simulation configuration and results, respectively, and Section 6.4, we give some concluding remarks.

6.2 Simulation Scenarios

In Fig. 6.1, we show the configuration of our evaluative model, through which we investigated the throughput characteristics of AF service and its achievable QoS. If we apply the DiffServ framework to the current Internet, an ISP domain will serve as a DiffServ domain. Users will be able to contract with an ISP there to obtain some assured bandwidth, and a target rate R_t; SLA should be assured. In the same manner, adjacent domains contract with each other at a domain boundary. Edge routers in DiffServ domains have meter and marker functions, (called conditioner function in [BBC+98]), and according to metered results, packets from end users are classified into two groups: IN and OUT, which respectively correspond to the DSCP AF11 and AF12 in the DiffServ framework [NBBB98]. When the metered arrival rate is over R_t, the marker marks the packet with OUT. Otherwise, the packet will be marked with IN. For example in Fig. 6.2, packets are marked with IN or OUT at the entrance of domain A according to rules stated previously and go through domain A. When they arrive at the edge router of domain B, packets marked with IN are metered again and will be remarked with OUT if the amount of IN packets arriving into domain B exceeds the contracted bandwidth, say $\sum R_t$, between domain A and B. Packets marked with OUT are not checked at all and go through the edge router.

To examine the possibility of AF service through multiple DiffServ domains, we will employ the simulation model shown in Fig. 6.2. Model(a) in Fig. 6.2 is an inter-domain model,

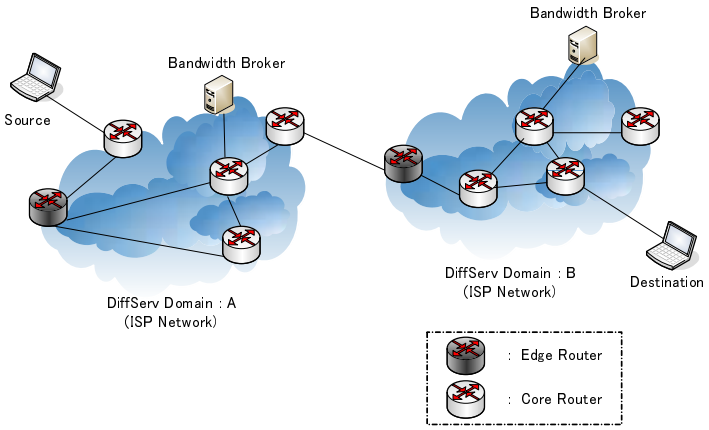


Figure 6.1: Diffserv network

which has two different Diffserv domains, domains A and B. Model (b) is also examined just for comparison, which is an intra-domain one consisting of a single Diffserv domain, domain A. It has one difference from model (a), i.e., (a) has a conditioner function at the edge of domain B,whereas model (b) does not.

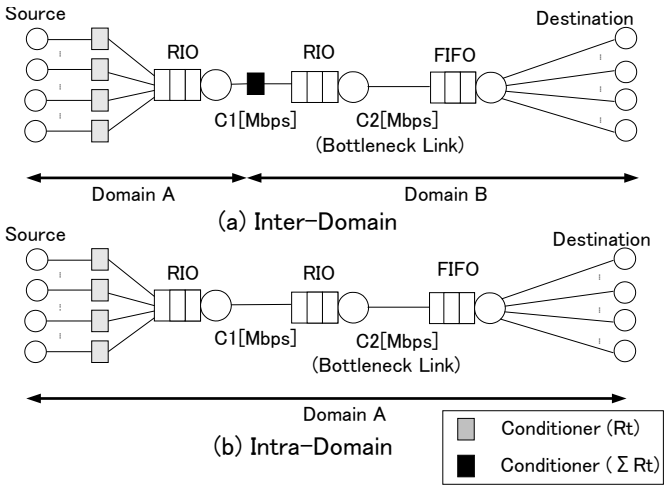


Figure 6.2: Simulation model for AF service with multiple queues

Parameters used in both models are as follows: R_t is the target rate of each flow, $\sum R_t$ denotes the contracted bandwidth between adjacent domains, domain A and B, and C_1 and

C_2 respectively denote the output link capacity of upstream and downstream RIO queues. We assume that the bandwidth on every link is large enough for routers to forward IN packets without any loss within each domain. Thus if packets are discarded, it will occur at only the edge router between two domains because of some inappropriate management in domain A.

There are two alternative schemes to realize the conditioner function at edge routers; one is a method via Token Bucket and the another uses a TSW (Time Sliding Window) algorithm introduced in [Fang99]. We do not show any obvious differences in the characteristics of Token Bucket and TSW in our preliminary simulation, as are shown in several other papers [IN98, CKKW, ADD00]. The size of Token Bucket greatly affects throughput characteristics, and one that is too large degrades the throughput performance, so it is sometimes referred to as a ”TCP hostile network element” in [Hus00]. Therefore, we employ a TSW mechanism for a conditioner in our simulation, in which the shaper is not used to allow flows to use available bandwidth left by the contracted one. All routers within domains are equipped with a RIO (RED with IN/OUT) queueing mechanism, which is based upon the RED algorithm. When congestion is detected in each router, packets marked with IN are preserved via the RIO algorithm, which aggressively discards packets marked with OUT compared to packets marked with IN, and as a consequence, end users are able to maintain a minimum contracted throughput.

We used Network Simulator version 2(ns-2)[Pro] and Sean Murphy’s Diffserv package, which is a contributed package for ns-2[Mur]. We also added some codes to simulate the TSW mechanism and other components. Each source node (# of source is nine) sends one TCP flow, which has a sack flavor and is used in many applications in the Internet, for example ftp, telnet, http, etc. We set the packet size to 1000 [byte] referred to [TMW97], the buffer size in each RIO queue to 200 [packet] and RIO parameters were set as below.

- (min_{in}, max_{in}, $Pmax_{in}$)= (100, 150, 0.02)
- (min_{out}, max_{out}, $Pmax_{out}$)= (50, 100, 0.1)

6.3 Simulation Results

Our major concern was to clarify how flows going through multiple DiffServ domains are different from those within a single domain in terms of their throughput characteristics. W. Fang et al. examined such flows in simulations over a limited model and concluded that there are no differences in throughput properties under both situations. They pointed out that the number of re-marked packets at boundary nodes was too small to have an impact on throughput characteristics.

In what follows, we examine the impact of re-marked packets on throughput characteristics by focusing on the following two items, which were not mentioned. First we investigate the influence of marking strategies adopted at a marker, and then examine the influence of an intra-domain network configuration.

6.3.1 Influence of the marking policy at a marker on throughput characteristic

(a) AF service with multiple queues

TSW is the algorithm enabling meter and marker functions in the DiffServ framework, in which the average arrival rate is calculated in a fixed time period, AVG-INTERVAL second (here, we set 1 second by referring to[FSN00]) and in cases where the metered average arrival rate is greater than R_t, arrival packets are marked with OUT. Otherwise, packets will be marked with IN with some degree of probability. We call this marking policy based on R_t the ”original_mark” one.

In addition, we investigate the properties of another marking policy in [Fan], which moderates OUT packets generated in ingress routers because of the following reasons. First, packets with OUT are more likely to be discarded in comparison with ones with IN in RIO queues. Second, the TCP protocol halves its window size after detecting discarded packets, thereby leading to severe throughput degradation during an interval. Therefore, packets are

marked with OUT in the following way in [Fan].

Suppose that an arrival rate is metered to be over R_t. In that case, the arriving packet is always marked with OUT in the original one, but will be marked with OUT with "some probability" in the new one. The "some probability" depends on the number of IN packets having already successively gone through until then; the probability becomes larger with the number of the successive IN packets. This allows the amount of packets marked with IN to exceed R_t for each flow. We call this policy "excess_in_mark".

Concerning provisioning matters in Diffserv domains, it is stated that the bottleneck link bandwidth is wider than (not equal) to the sum of the contracted rates of each flow [GDJL00]. Our preliminary simulation results in the single domain model have shown that when bottleneck link bandwidth is configured over $1.3 \times \sum R_t$, the contracted of each flow is achieved. From this result, we examined each provisioning situation shown in Table 6.1 over Fig. 6.2.

Table 6.1: Network provisioning in Fig. 3 ($\sum R_t$ = 24 [Mbps])

	C_1[Mbps]	C_2 [Mbps]
Case 1	$1.5\sum R_t = 36$	$1.4\sum R_t = 33.6$
Case 2	$3.0\sum R_t = 72$	$1.4\sum R_t = 33.6$
Case 3	$3.0\sum R_t = 72$	$2.0\sum R_t = 48$

Table 6.2 shows the throughput characteristics TH [Mbps] for each flow on model (a) in Fig. 6.2, adopting the original_mark policy at each conditioner. Table 6.3 shows one that adopts an excess_in_mark policy. The mark, "*" in the TH column, indicates that the corresponding flows do not achieve a throughput greater than or equal to their R_t. We observed that all flows can achieve their R_t when the original_mark policy is adopted as the policy in a marker. However, degradations on the throughput of some flows can be observed when an excess_in_mark policy is adopted. The flows with throughput degradation are very similar in terms of their large R_t and RTT. Even though in Case 3, the bottleneck link is twice that of $\sum R_t$, flow # 9 does not achieve its target rate.

For comparison, we ran a simulation in the single domain model shown in Fig. 6.2 (b),

Table 6.2: Throughput characteristics in inter-domain (original_mark)

#	R_t [Mbps]	RTT [ms]	Case 1 TH[Mbps]	Case 2 TH[Mbps]	Case 3 TH[Mbps]
1	1	30	3.304	2.951	4.816
2	2	30	3.637	3.671	6.151
3	5	30	6.056	5.735	7.829
4	1	40	2.271	2.353	4.421
5	2	40	3.010	3.068	4.579
6	5	40	5.583	5.466	6.282
7	1	50	2.173	2.086	3.052
8	2	50	2.811	2.922	3.864
9	5	50	5.036	5.183	5.612

and give the obtained throughput in Table 6.4 and Table 6.5 by respectively adopting the two policies.

In Table 6.4 and Table 6.5, no deterioration in the throughput characteristics can be observed in either marking policy within a domain. Namely, the throughput degradation is only observed in flows going through multiple domains using the excess_in_mark policy.

To investigate why the throughput characteristics of flows going through multiple domains degrades, we examined the number of IN and OUT packets in individual flows at certain points along the way. Table 6.6 shows the # of remarked packets per flow in Fig. 6.2(a) with excess_in_mark policy at the marker. Here, we will focus on one specific flow, whose throughput characteristics degraded, which is Case 1 in Table 6.3.

Let's check the number of packets with IN and OUT in flow #9; Table 6.6 indicates they are respectively 26092 and 390 at the ingress router of domain A. In addition, 2329 packets are re-marked from IN to OUT at an ingress conditioner of domain B, and consequently the number of arriving packets marked with IN and OUT at the entrance of domain B is respectively 23763 and 2719. Accordingly, the number of packets marked with OUT in domain B is almost seven times larger than that of domain A. The packet loss probability can increase as well in domain B, thereby resulting in degradation of the throughput characteristics. On

Table 6.3: Throughput characteristics in inter-domain (excess_in_mark)

#	R_t [Mbps]	RTT [ms]	Case 1 TH[Mbps]	Case 2 TH[Mbps]	Case 3 TH[Mbps]
1	1	30	3.596	3.569	6.168
2	2	30	4.049	3.773	5.639
3	5	30	5.723	5.205	6.623
4	1	40	3.031	3.021	4.502
5	2	40	3.414	3.371	4.894
6	5	40	4.542*	5.054	5.931
7	1	50	2.135	2.466	4.095
8	2	50	3.057	2.775	4.036
9	5	50	4.302*	4.220*	4.885*

the other hand, in flow #1, which has relatively small R_t and RTT, a smaller number of packets are re-marked, as shown in Table 6.6. This only slightly increases the number of OUT packets, from 6761 to 7936, which does not cause the throughput performance to degrade.

For comparison, Table 6.7 gives the number of re-marked packets for the original_mark policy in Case 1. Quite a few packets are re-marked, indicating that the re-marking does not cause the performance degradation.

(b) Influence of combination of marking policy

Next we will examine the impact on throughput characteristics of using a combination of marking policies. Figure6.3 shows the simulation model, in which there are three different domains, A,B, and C. Flows A and B go through two domains, and each of them experience two markers.

Table 6.8 shows what kind of policy is used at the edge of each domain. Domain A uses an excess_in_mark policy, while domain B adopts the original_mark policy. Two cases are examined: domain C employs the excess_in_mark policy in Case 1, but the original_mark policy in Case 2. Table 6.9 shows the throughput characteristics for each flow and the mark "*" indicates that the corresponding flow does not achieve its R_t.

Table 6.4: Throughput characteristics in intra-domain (original_mark)

#	R_t [Mbps]	RTT [ms]	Case 1 TH[Mbps]	Case 2 TH[Mbps]	Case 3 TH[Mbps]
1	1	30	2.964	2.930	5.114
2	2	30	3.549	3.581	5.885
3	5	30	5.958	5.915	7.452
4	1	40	2.446	2.383	4.364
5	2	40	3.391	2.985	4.876
6	5	40	5.389	5.525	6.276
7	1	50	2.141	2.150	3.398
8	2	50	2.791	2.972	3.758
9	5	50	5.243	5.027	5.610

Referring to Table 6.9, we focus on the results of the case where C_1 and C_2 are set at 36 [Mbps] and 67.2 [Mbps]. In such a case, flows #6 and #9 in flow group A cannot attain their target throughput R_t from Table 6.9. On the other hand, flows #15 and #18 (in Flow B) can achieve their R_t, although they are the same as flows #6 and #9 in terms of their R_t and RTT. The only difference is their marking policy. In addition for the case where C_1 is set to 72 [Mbps] and C_2 is set to 96 [Mbps], the same goes for flows #9 and #18. The throughput characteristics are determined by the marking policy employed at the upstream, but are not dependent on the marking policy employed at the downstream.

6.3.2 Cases of multiple RIO queues in a domain

In the previous subsection, we showed that the marking policy can affect the throughput performance of flows going through multiple domains. In addition, it has been demonstrated that an appropriate, original_mark policy can successfully prevent performance degradation. However, the simulation model in Fig. 6.2 is very simple, and consists of only one node in domain A. In fact, packets usually go through several nodes within a domain. Therefore, we now present a case of multiple nodes with RIO queues within a domain, which is illustrated in Fig. 6.4. The model shown in Fig. 6.4(a) is an inter-domain one, and (b) is an intra-domain

Table 6.5: Throughput characteristics in intra-domain (excess_in_mark)

#	R_t [Mbps]	RTT [ms]	Case 1 TH[Mbps]	Case 2 TH[Mbps]	Case 3 TH[Mbps]
1	1	30	2.901	2.984	6.006
2	2	30	3.748	3.730	5.828
3	5	30	5.988	5.808	7.109
4	1	40	2.619	2.672	4.544
5	2	40	3.159	3.108	4.622
6	5	40	5.452	5.247	6.254
7	1	50	2.223	2.168	3.423
8	2	50	2.868	2.680	3.846
9	5	50	5.091	5.140	5.704

configuration just for comparison. In Fig. 6.4(a), there are three groups of TCP flows, A1, A2, and AB, and each group contains nine TCP flows. Flow AB goes through both of domains A and B, while Flows A1 and A2 consist of only intra-domain flows. Flows A1 and A2 share some queues with Flow AB, and they may make IN packets of Flow AB bursty. We respectively set C_1= 36 [Mbps], C_2= 66 [Mbps], and C_3= 30 [Mbps].

Table 6.10 shows results obtained in simulations lasting 50 seconds; the number of lost packets is denoted by N_{loss}, and the packet loss rate is denoted by P_{loss}. N_{remark} indicates the number of re-marked packets, and P_{remark} represents a ratio of the number of re-marked packets to that of all generated IN packets. Only a small number of IN packets are re-marked with OUT in model (a), and we can see from Table 6.11 that there is not a significant difference between the performance of the two models.

To explain the reason for this phenomenon, we will focus on Flow AB and compare the distribution of its departure time interval at Point A and that of its interarrival time at Point B, both of which are illustrated in Fig. 6.5. They are very similar. Going through multiple queues does not make the packet interarrival time at the ingress of domain B so bursty, and this successfully prevents IN packets from being adversely re-marked with OUT, as shown in Table 6.8 (only 64 packets are re-marked from IN to OUT).

Table 6.6: # of re-marked packets (excess_in_mark)

#	R_t	RTT	Edge of Domain A			Edge of Domain B	
	[Mbps]	[ms]	IN	OUT	IN → OUT	IN	OUT
1	1	30	13969	6761	1175	12794	7936
2	2	30	17412	5603	1501	15911	7104
3	5	30	31924	2977	2788	29136	5765
4	1	40	12543	5901	1054	11489	6955
5	2	40	16005	4300	1418	14587	5718
6	5	40	29048	1414	2578	26470	3992
7	1	50	10260	4208	945	9315	5153
8	2	50	15062	3558	1276	13786	4894
9	5	50	26092	390	2329	23763	2719

6.4 Concluding Remarks

In this chapter we have investigated the throughput behavior of TCP flows over multiple AF DiffServ domains. In particular, we focused on the influence of packets re-marked at the domain boundary on the throughput characteristics, we were able to clarify the following items:

- Re-marked packets at a Diffserv domain boundary can degrade end-to-end throughput, i.e. quality of service.
- Applying an appropriate marking policy, (e.g. the original_mark policy presented in Section 6.1, which is based on a contracted rate) at the ingress node in each DiffServ domain, can reduce the number of re-marked packets.
- The network configuration of an intra-domain has less effect on throughput degradation.

Table 6.7: # of re-marked packets (original_in_mark)

#	R_t	RTT	Edge of Domain A			Edge of Domain B	
	[Mbps]	[ms]	IN	OUT	IN → OUT	IN	OUT
1	1	30	6324	13167	13	6311	13180
2	2	30	12302	9756	27	12275	9783
3	5	30	30483	6974	59	30424	7033
4	1	40	6134	9082	17	6117	9099
5	2	40	12264	7438	28	12236	7466
6	5	40	29899	3246	57	29842	3303
7	1	50	6252	6302	11	6241	6313
8	2	50	12272	5296	29	12243	5325
9	5	50	28579	1715	66	28513	1781

Table 6.8: Combination of marking policy

Case1			
	Edge of Domain A	Edge of Domain B	Edge of Domain C
FlowA	excess_mark		excess_mark
FlowB		original_in_mark	excess_mark
Case2			
	Edge of Domain A	Edge of Domain B	Edge of Domain C
FlowA	excess_mark		original_mark
FlowB		original_in_mark	original_mark

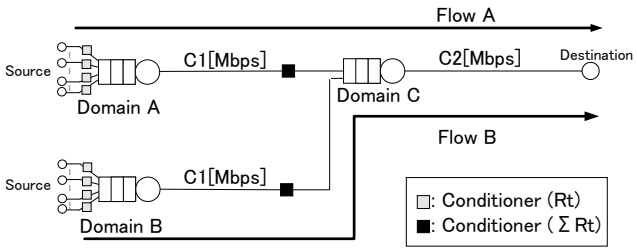


Figure 6.3: Simulation model for combination of marking policies

Table 6.9: Throughput characteristics in Fig. 6.3

	#	R_t [Mbps]	RTT [ms]	C_1=36,C_2=67.2 Case1 [Mbps]	C_1=36,C_2=67.2 Case2 [Mbps]	C_1=72,C_2=96 Case1 [Mbps]	C_1=72,C_2=96 Case2 [Mbps]
Flow A	1	1	30	3.556	3.647	5.962	6.429
	2	2	30	3.900	4.013	6.484	5.686
	3	5	30	5.542	5.770	6.808	5.775
	4	1	40	3.054	3.015	4.173	4.696
	5	2	40	2.940	3.106	5.091	4.758
	6	5	40	4.688*	4.400*	5.583	5.286
	7	1	50	2.341	2.318	3.712	3.728
	8	2	50	2.930	2.668	4.134	3.840
	9	5	50	4.508*	3.973*	4.591*	4.153*
Flow B	10	1	30	2.734	2.855	4.680	5.211
	11	2	30	3.298	3.637	4.955	5.462
	12	5	30	5.788	5.843	7.371	7.387
	13	1	40	2.223	2.284	3.708	4.030
	14	2	40	2.977	2.918	4.407	4.520
	15	5	40	5.457	5.406	6.290	6.335
	16	1	50	2.081	2.063	3.048	3.321
	17	2	50	2.848	2.863	3.804	3.639
	18	5	50	5.132	5.118	5.594	5.649

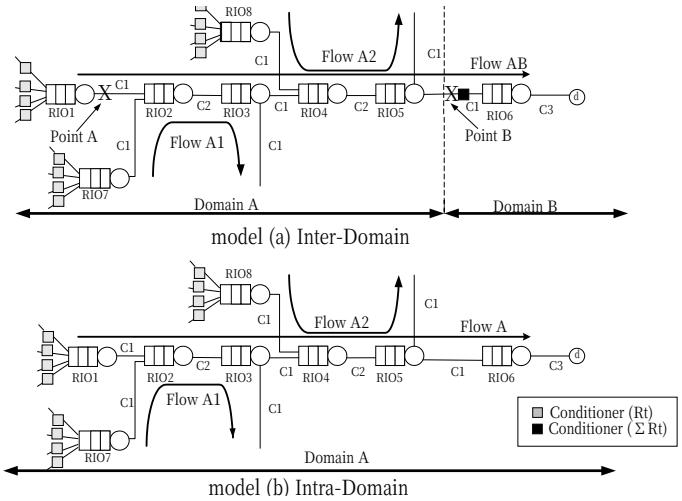


Figure 6.4: Simulation model for multiple RIO queues in a domain

Table 6.10: Packet loss prob.(simulation period = 50[sec])

	model(a)	model(b)
N_{loss}/P_{loss} in RIO2(IN)	0/0.0	0/0.0
N_{loss}/P_{loss} in RIO2(OUT)	379/0.00336	465/0.004141
N_{loss}/P_{loss} in RIO4(IN)	0/0.0	0/0.0
N_{loss}/P_{loss} in RIO4(OUT)	133/0.001166	193/0.001707
N_{loss}/P_{loss} in RIO6(IN)	0/0.0	0/0.0
N_{loss}/P_{loss} in RIO6(OUT)	283/0.007109	228/0.0005757
N_{remark}	64	-
P_{remark}	0.00044	-

Table 6.11: Throughput characteristics in Fig. 6.4

	#	R_t [Mbps]	RTT [ms]	model(a)	model(b)
Flow AB/A	1	1	30	2.027	2.036
	2	2	30	2.807	2.958
	3	5	30	5.475	5.371
	4	1	40	1.941	1.945
	5	2	40	2.623	2.640
	6	5	40	5.215	5.143
	7	1	50	1.687	1.628
	8	2	50	2.390	2.466
	9	5	50	4.965	4.984
Flow A1	10	1	30	3.025	3.201
	11	2	30	3.864	3.842
	12	5	30	5.864	5.767
	13	1	40	2.533	2.547
	14	2	40	3.222	3.073
	15	5	40	5.448	5.342
	16	1	50	2.146	2.103
	17	2	50	2.769	2.739
	18	5	50	5.211	5.290
Flow A2	19	1	30	2.884	3.079
	20	2	30	3.678	3.582
	21	5	30	6.107	5.814
	22	1	40	2.501	2.539
	23	2	40	3.074	3.464
	24	5	40	5.630	5.656
	25	1	50	2.377	2.160
	26	2	50	2.992	2.944
	27	5	50	5.265	5.196

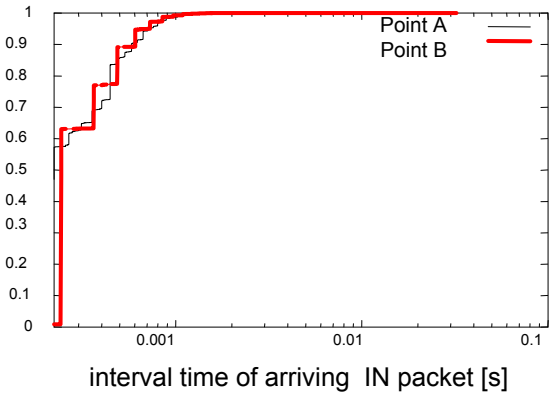


Figure 6.5: Distribution of interval time of arriving packets marked with IN

Chapter 7

Adaptive Early Packet Discarding Scheme to Improve Network Delay Characteristics of Real-Time Flows

7.1 Introduction

The performance of real-time network applications is greatly affected by delay and loss experienced by packets as they traverse a network. Some real-time applications set limits for acceptable network delay. For example, VoIP defines service classes based on an end-to-end packet delay limit and the rate of packet loss for flows in a network [SSS03]. In such applications, packets delayed longer than an acceptable limit are invalidated by their applications when they reach their destinations, even though they have successfully arrived at the receiver. These packets are useless for their applications, and thus they just impose an excess load on the network. In general, if network resources are exhausted by traffic that will eventually be discarded, significant performance deterioration will be seen even though those resources are fully utilized.

In this chapter therefore, by extending our preliminary work [KTO06], an early packet

discarding scheme is proposed and its effectiveness in congested networks is demonstrated through simulation. In the scheme, those packets not contributing to the performance of real-time applications are discarded in advance at intermediate nodes. The proposed scheme includes two kinds of router-supported mechanisms: one is called Maximum Transmission Queue Delay (MTQ), which resembles the concept of a Maximum Transmission Unit (MTU), and the other is Queue Delay To Live (QTL), which is similar to the mechanism for Time To Live (TTL). Both of these require the use of an additional header field in IP or UDP packets (e.g. an IPv6 optional header) to convey the queuing delay information for each packet.

It is assumed that a fixed amount of minimum delay (e.g. propagation and transmission delays) can be known or predicted for packets traversing from the sender to the receiver before they are sent, and thus only the variable part of the delay (mainly queuing delays) has been taken into account in this scheme. For example, if the acceptable total network delay for an application is 50 [ms] and the fixed delay on an end-to-end path is 30 [ms], the acceptable total queuing delay is 20 [ms]. It is further assumed that real-time application flows are treated separately at network nodes to other elastic traffic such as TCP flows, in terms of the bandwidth (i.e. the queuing buffer) shared by the flows. Thus, this chapter will focus only on real-time flows and their queuing delay in a network. As noted later in Section 7.3, the intention is to apply the proposed scheme not to end equipment for improving performance of individual flows, but to network nodes via a network provider to improve overall network performance (and for increasing the number of flows accommodated). The above assumptions are applicable to such a scenario.

To evaluate the proposed scheme on real-time application flows in terms of increasing the number of packets that could be utilized by the receiver-side application, Effective Packet Loss Rate (EPLR) is introduced. EPLR is equal to one minus the ratio of the number of packets that successfully reach the receiver along a path in experiencing a total queuing delay shorter than the acceptable upper limit induced by the delay-sensitive application.

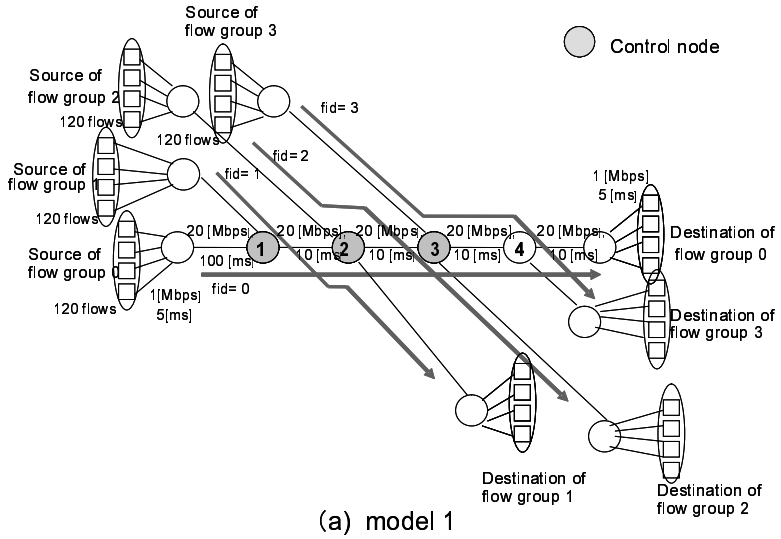
Since the flows competing for and sharing the network resource have individual different conditions, our goal is to achieve a good overall performance in the sense of max-min fairness on the application-induced delay performance among all flows. Namely, in the present work, we try to minimize the worst condition flows' EPLR on the network instead of the EPLR averaged over flows.

This chapter is organized as follows. An extensive network simulation and its results are described in Sections 7.2 (overview of simulation models), 7.3 (homogeneous environment), and 7.4 (heterogeneous environment). Finally, some conclusions are presented in Section 7.5.

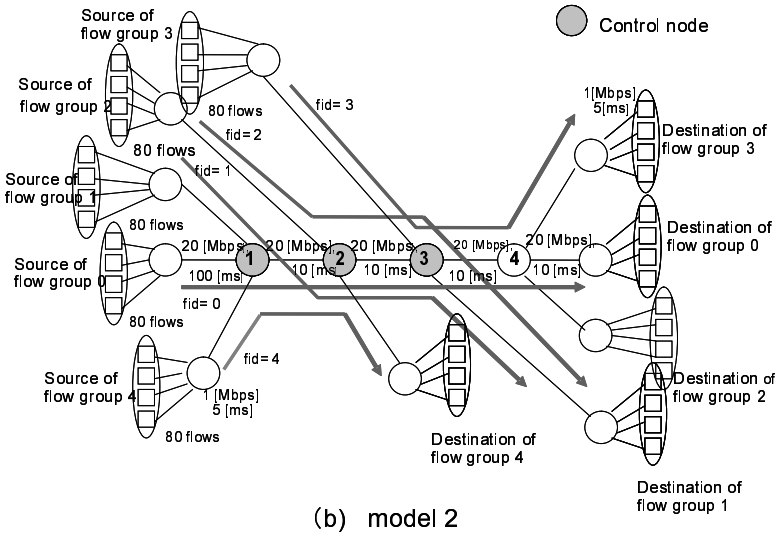
7.2 Simulation models

We performed the simulation using ns release 2.27 [Pro], with MTQ and QTL mechanisms additionally implemented to manage the queues. The simulation models we used are outlined in Fig. 7.1. We focused on two network configurations, models 1 and 2. In both models, the length of each queuing buffer is equal to 200 packets for each output link with bandwidth of 20 [Mbps]. In both models, a number of real-time application flows on the network are grouped such that flows in the same group have the same source and destination along the same path, and thus will experience an identical delay on average. These flows meet queuing delays only at nodes 1, 2, and 3 (the gray-colored nodes in the figure) due to flow competition.

Table 7.1 lists the number of flows for each group and the total transmission rate of the flows in the group. In this configuration, flow group 0 is most likely to compete with other flow groups (at three nodes) and to suffer the largest queuing delay. Only two flow groups (group 0 and one other) will compete with each other at a node in model 1, while three flow groups will do so in model 2. We targeted one traffic configuration in model 1, where the average utilization is 96 %, while two traffic configurations are considered in model 2, where average utilization are 96 % and 84 %. Since the results we observed in



(a) model 1



(b) model 2

Figure 7.1: Simulation model

the simulation showed similar trends for all configurations, the simulation results for model 2(average utilization is 96 %) only will be shown, unless otherwise noted.

We applied an ON-OFF burst model to traffic of each real-time flow, in which an exponential distribution for its ON and OFF periods [Fio00] was used. We examined several types of traffic, with average ON/OFF durations set to 250/650 [ms], 350/650 [ms], 450/650 [ms] and 3500/6500 [ms]. In each ON period, fixed sized packets flow at a constant rate. The average rate (over ON and OFF periods) of each flow was fixed at 80 [kbps], and the packet length was 200 [byte] (e.g. G.711 VoIP codec). Since we observed similar characteristics for the results for all traffic types, the results for only one type (that with a reasonable burstiness, i.e. average ON/OFF duration of 350/650 [ms]) will be shown due to space limitations.

Table 7.1: No. of flows in heavily congested scenario in model 1 (top) and in model 2 (middle) and in moderately congested scenario in model 2 (bottom)

(a) model 1, average link utilization: 96 %

Congested link	Flow id(fid)				No. of flows	Total rate [Mbps]
	0	1	2	3		
$1 \rightarrow 2$	120	120			240	19.2
$2 \rightarrow 3$	120		120		240	19.2
$3 \rightarrow 4$	120			120	240	19.2

(b) model 2, average link utilization: 96 %

Congested link	Flow id(fid)					No. of flows	Total rate [Mbps]
	0	1	2	3	4		
$1 \rightarrow 2$	80	80			80	240	19.2
$2 \rightarrow 3$	80	80	80			240	19.2
$3 \rightarrow 4$	80		80	80		240	19.2

In our simulation models, the maximum queuing delay at each node is 16 [ms], because at most 200 packets each of length 200 [byte] are waiting to be sent to a link with a bandwidth of 20 [Mbps]. A packet in flow group 0 competes with traffic in the other flow groups at three nodes along the path, and thus the total queuing delay can be as much as 48 [ms]. Hence,

(c) model 2, average link utilization: 84 %

Congested link	Flow id(fid)					No. of flows	Total rate [Mbps]
	0	1	2	3	4		
$1 \rightarrow 2$	70	70			70	210	16.8
$2 \rightarrow 3$	70	70	70			210	16.8
$3 \rightarrow 4$	70		70	70		210	16.8

we examined three cases of acceptable total queuing delay: 10 (strict), 20 (moderate), and 30 (loose) [ms], which could correspond to three different application delay constraints or be derived from three different network environments that have different fixed delays. Since we observed similar characteristics for the results for the two cases with acceptable delay of 20 [ms] and 30 [ms], only the results for cases with 10 [ms] and 20 [ms] will be shown due to space limitations.

As mentioned in Section 7.1, we use Effective Packet Loss Rate (EPLR) as a metric for evaluating performance with respect to delay in real-time applications. This is the ratio of the number of packets discarded at nodes or invalidated by the receiver (due to a larger delay than the acceptable total queuing delay), to the total number of packets sent by the sender.

7.3 Simulation results in homogeneous environments

We evaluated our mechanisms via network simulations in homogeneously congested environments in which all flows had an identical delay requirement (i.e. an acceptable total queuing delay).

Each flow traversed one or more nodes in which it competed with other flows, where each node had an output queuing buffer corresponding to a maximum local queuing delay of 16 [ms]. Therefore, if neither MTQ nor QTL were used, very high EPLRs could be seen even for a moderate acceptable total queuing delay of 20 [ms], as illustrated in Fig. 7.2. In particular, flows in flow group 0 experienced very high EPLRs because they competed with other flows either in the same flow group or in different ones at three nodes. In order to

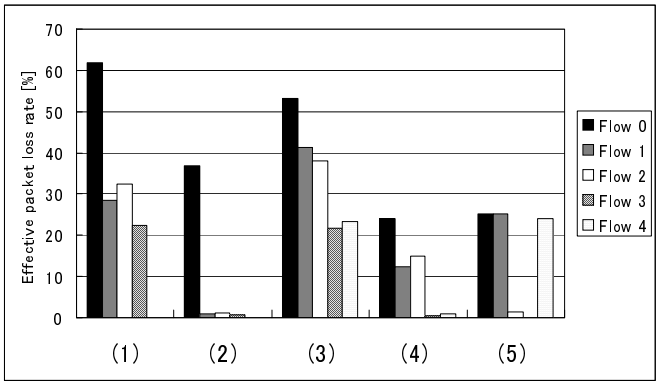


Figure 7.2: *EPLR* (effective packet loss rate), using neither MTQ nor QTL

Table 7.2: Parameter list of Fig. 7.2

Cases	Models	acceptable queuing delays [ms]
(1)	model 1	10
(2)	model 1	20
(3)	model 2	10
(4)	model 2	20
(5)	model 2 in Sec.7.3.4	10

reduce the high EPLRs, in Sections 7.3.1, 7.3.2, and 7.3.3, schemes with MTQ only, QTL only, and QTL plus MTQ are used.

Note that an MTQ value of more than 16 [ms] is meaningless, because that is the maximum queuing delay at each node. Meanwhile, a QTL value of more than 48 [ms] can also be considered meaningless, because this is higher than the total queuing delay in traversing at most three congested nodes along a path.

7.3.1 Effectiveness of MTQ mechanism

Setting MTQ for a packet is equivalent to limiting the length of the queuing buffer at every node the packet traverses. In general, as the length of the queuing buffer at network nodes decreases, the queuing delays experienced by packets decrease. The number of packets invalidated due to the acceptable delay limit at the receiver also decreases, while the packet losses due to buffer overflow at network nodes increase. This gives rise to a trade-off for lowering EPLR. To clarify the relationship between EPLR and MTQ, we first show the packet losses occurring at intermediate nodes in the network, and then investigate EPLR.

Figure 7.3 represents the network packet loss rate of each flow group when applying the MTQ mechanism for a variety of different MTQ values in the widest possible range. In this case, for the original queuing buffer length of 16 [ms], the packet loss rate is about 2 % at worst (i.e. in flow group 0). As the MTQ value decreases, the loss rate increases just slightly as long as the MTQ value is larger than a threshold of about 1.8 [ms]. Below this threshold however, the loss rate drastically increases due to the queuing buffer being so short that it cannot absorb a moderate burstiness in those flows.

Figures 7.4 (a) and (b) show EPLR for a strict acceptable queuing delay of 10 [ms] with a variety of MTQ values in models 1 and 2, respectively. Fig. 7.4(c) also shows EPLR against MTQ, for a moderate acceptable queuing delay of 20 [ms] in model 2. Note that each figure shows a very large EPLR corresponding to an MTQ of 16 [ms], and this is equivalent to the EPLR achieved without applying our scheme. This can also be seen in cases (1) - (4) of

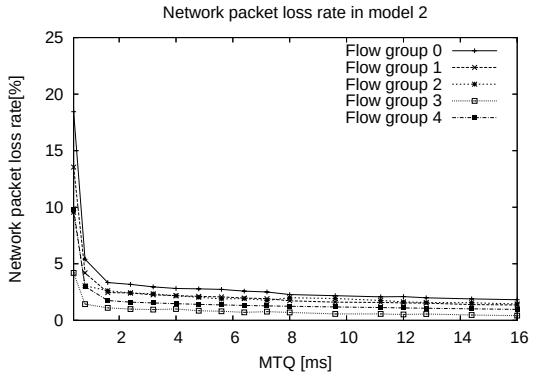


Figure 7.3: network packet loss rate, using MTQ alone

Fig. 7.2. This clearly indicates a general and significant reduction of EPLR by applying the MTQ mechanism.

In Fig. 7.4 (a) and (b) for an acceptable total queuing delay of 10 [ms], when the MTQ value exceeds about 3.4 [ms] the EPLR of flow group 0 suddenly increases. This indicates that a number of packets in that group suffer from large queuing delays over the 10 [ms] limit allowed by the application. That is, setting the MTQ value in a range from 1.8 to 3.3 [ms] significantly improves the EPLR of that flow group traversing three congested nodes along its path. Considering flow groups 1, 2 and 3 in the first figure, these groups which traverse one congested node show acceptably small EPLRs when MTQ is set in a wide range from 1.8 to 10 [ms], even though the reduction of MTQ within this range results in negligible deterioration (increase) of EPLR. Therefore, an MTQ value of 3.3 [ms] can be considered to be optimal for the overall performance of all flow groups for this setting. Similarly, in the second figure, while the optimal MTQ for flow groups 1 and 2 which traverse two congested nodes is about 5 [ms], the same MTQ value of 3.3 [ms] for flow group 0 still seems to be optimal. These results imply that, for a flow obeying an acceptable total queuing delay of D and traversing N equally congested links along a path, setting the MTQ to D/N (or slightly less) significantly reduces the queuing delay experienced by the flow. An MTQ value less than D/N ensures that the total queuing delay experienced by the packet traversing N nodes

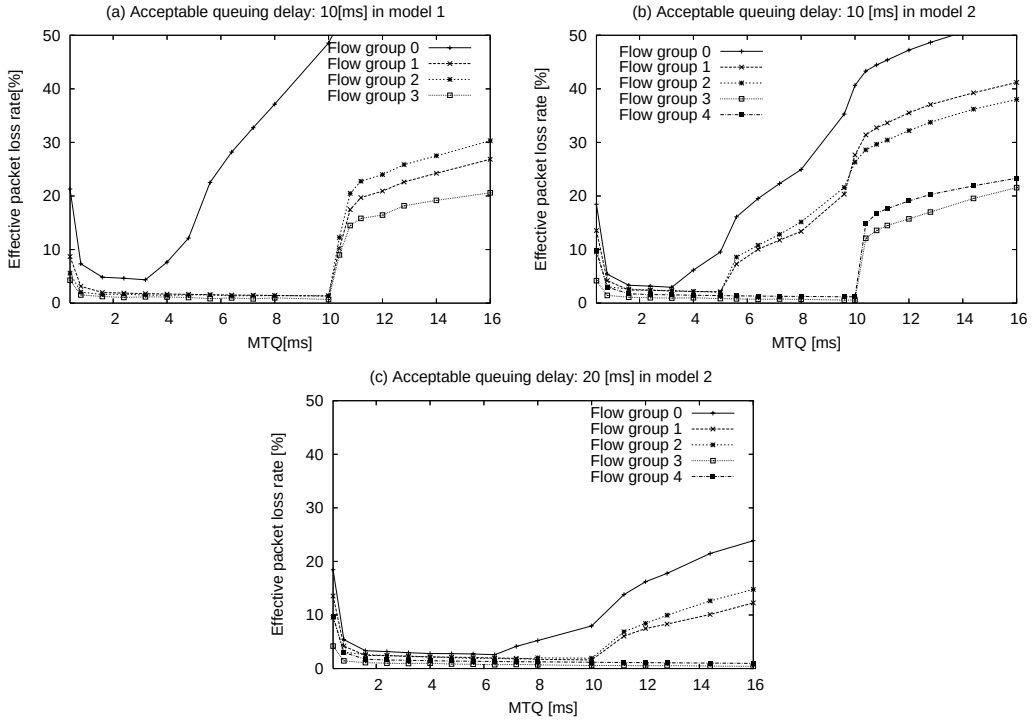


Figure 7.4: *EPLR* (effective packet loss rate), using MTQ alone

is less than D if the packet is not discarded at those nodes.

Consequently, an appropriate MTQ value range could keep EPLR small for all flows. In the above scenario, Fig. 7.3 indicates that packet losses remain low if the MTQ is larger than a threshold (1.8[ms]), which is smaller than D/N ($10/3 \approx 3.3$ [ms]), where N is the largest number of congested links through which the worst case flows traverse. Therefore, for every flow, setting MTQ to 3.3 [ms] means that no packets experience a total queuing delay larger than 10 [ms], and only a small number of packets are discarded at intermediate nodes. Figure 7.4(c) shows EPLR for an acceptable total queuing delay of 20 [ms], and also supports the above findings. Compared with the previous figures, the delay performance of flow groups 3 and 4 seems not to be improved by any MTQ value. The reason might be that since these flow groups traverse only one congested node with an acceptable queuing delay of 20 [ms], the original queuing buffer length of 16 [ms] is short enough to avoid significant delays. Note that, obviously the above simple D/N-rule is not always suitable. For example, the number N of the congested nodes may not be known or N may be such large that D/N becomes shorter than a limit queuing buffer length necessary to avoid unacceptable packet losses.

In congested networks, a packet experiencing a long queuing delay at a node is likely to ultimately exceed the acceptable total queuing delay limit due to additional queuing delays at following nodes, even if the delay experienced at that node does not exceed its limit. Setting MTQ to an appropriate value might aggressively discard in advance those packets that will probably exceed the limit before reaching their destinations, and thus improve the overall delay performance. However, an appropriate value of MTQ, derived from a trade-off between network packet losses and queuing delay, is sometimes difficult to determine in reality. The lower boundary of the effective value range of MTQ depends directly on packet losses at a node, which are related to link utilization and incoming traffic characteristics (e.g. burstiness) at that node. The upper boundary is strongly dependant on how long a queuing delay will be experienced at each of the subsequent nodes, which is related to the flow competition

at those nodes. Both are not always easy to estimate, and in some cases are difficult. Note that in this simple version of MTQ, the MTQ value in a packet traversing network nodes does not change node by node. This simple version might therefore be ineffective when, for example, a number of heterogeneously congested nodes are on a path. Thus, using only an MTQ mechanism in a network may not always be effective. We will examine the use of QTL in the next subsection.

7.3.2 Effectiveness of QTL mechanism

The EPLRs when only the QTL mechanism is used are shown in Fig. 7.5 (a), (b) and (c) for a variety of QTL values.

These figures clearly show that the EPLR is reduced most for every flow when QTL is set to exactly the acceptable total queuing delay limit; if QTL exceeds this optimal limit, the delay performance deteriorates greatly. QTL can be set to a smaller value in order to aggressively discard in advance those packets likely to exceed the limit before reaching their destinations. However, the figures show that setting QTL to a value smaller than the optimal limit is detrimental to some degree, unlike for MTQ. Note that Fig. 7.5 (a) for model 1 shows an interesting phenomenon due to the nature of QTL. For flow groups 1, 2, and 3 traversing one congested node only, setting QTL is equivalent to setting MTQ, and thus a QTL value larger than the original queuing buffer length of 16 [ms] would be expected to have no effect. However, this is not the case for flow groups 2 and 3 because if the QTL value becomes larger, the larger volume of traffic in flow group 0 will survive when it competes with groups 2 and 3 at nodes 2 and 3 respectively. This implies that an appropriate QTL setting will become more important as the number of nodes along a path increases.

A QTL value exactly equal to the acceptable total queuing delay results in the conservative discarding of packets which have already exceeded the total delay limit. Other simulation results (obtained in our simulation but not shown in this paper) also support this observation that such a QTL setting is always optimal in terms of reducing EPLR regardless

of the simulation model, optimal values, or degree of traffic burstiness.

This simple rule for setting the QTL value is of practical importance from an operational standpoint. However, the EPLR reduction achieved by using an optimal QTL alone seems inferior compared to using optimal MTQ alone. Therefore, in the next subsection we will examine the usage of QTL and MTQ together, in order to exploit the combination of conservative discarding via QTL and aggressive discarding through MTQ.

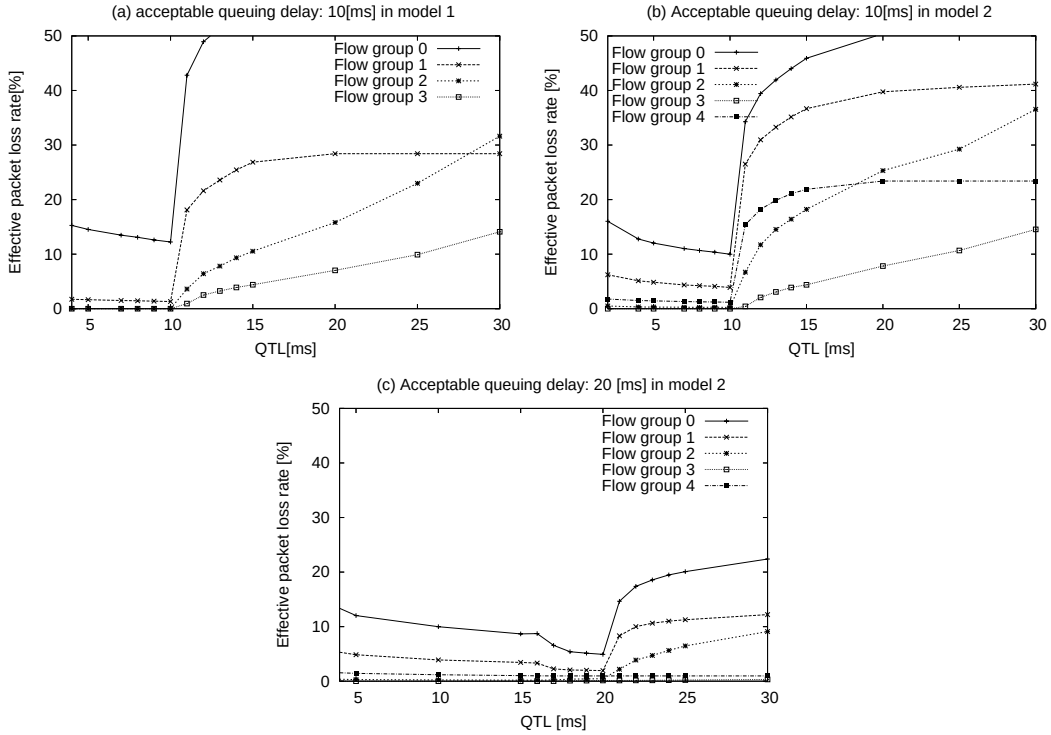


Figure 7.5: *EPLR* (effective packet loss rate), using QTL alone

7.3.3 Effectiveness of setting MTQ and QTL simultaneously

Figures 7.6 (a) and (b) show EPLR for an acceptable total queuing delay of 10 [ms] when QTL and MTQ are used together. QTL is set to 10 [ms], equal to the queuing delay requirement. MTQ is set within a range from 0 to 16 [ms], although an MTQ value greater than the

QTL value (10 [ms]) is meaningless in this combination (i.e. it is equivalent to setting QTL alone).

The benefits of using MTQ together with QTL (compared to QTL alone) are clearly shown by the fact that reducing MTQ from 10 [ms] causes a reduction in EPLR for flow group 0, which is the worst flow group in terms of EPLR.

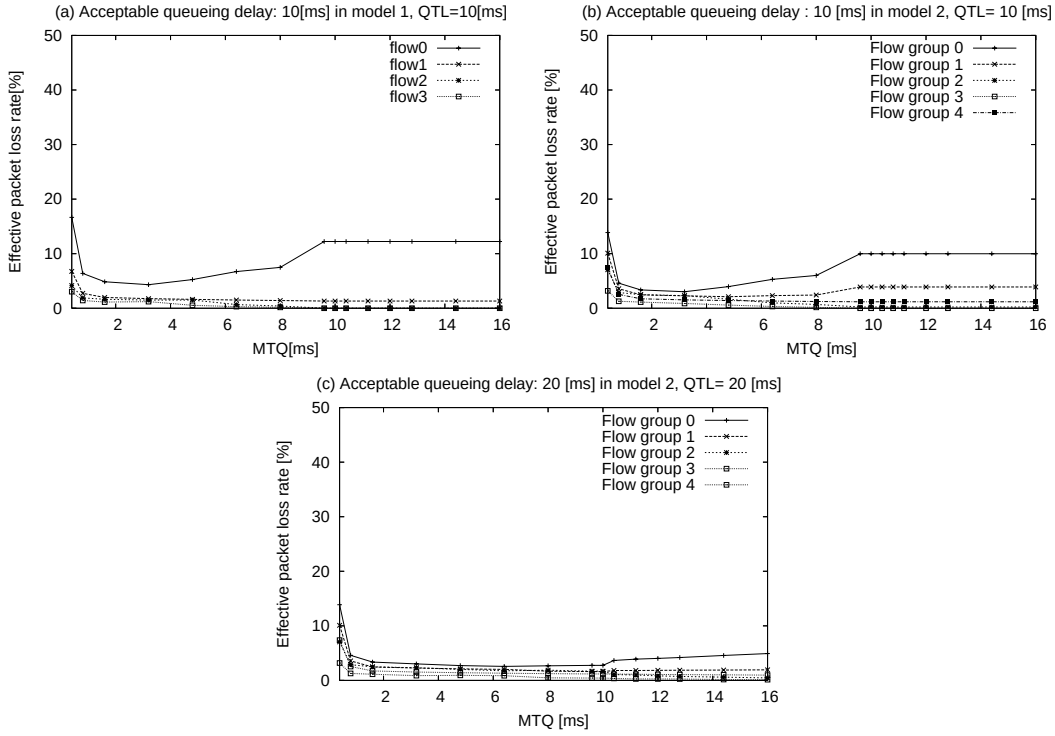


Figure 7.6: *EPLR* (effective packet loss rate), using QTL plus MTQ

Compared with Fig. 7.4 (a) and (b), the advantage of using QTL with MTQ (compared to MTQ alone) is that a wider range of MTQ values (at least those less than the QTL value) can result in a considerable reduction of EPLR. On the other hand, when the acceptable total queuing delay is 20 [ms] (Fig. 7.6(c)), the effect of setting MTQ is not clear because the queuing buffer (equivalent to 16 [ms]) is already shorter than the acceptable total queuing delay. Note that setting QTL to a value larger than $n \times MTQ$, where n is the number of intermediate nodes along the path, is meaningless when setting MTQ and QTL simultaneously.

Because the maximum waiting time for each packet per node is MTQ, the total queuing delay can not be longer than $n \times MTQ$. Thus, if MTQ and QTL will be used simultaneously to exploit a combined effect, they should be set within the range: $MTQ < QTL < n \times MTQ$ where n is the number of intermediate nodes.

Before ending this subsection, we present in-depth analysis of the performance of the proposed scheme, in using four parameter configurations, (a) no parameters set, (b) MTQ = 5[ms], (c) QTL = 10[ms] and (d) MTQ = 5[ms], QTL = 10[ms] in model 2, where total acceptable queuing delay is 10[ms].

The Figure 7.7 presents the EPLR of each flow, showing its proportion of (1): the packets discarded at nodes due to buffer overflow, MTQ, or QTL, and (2): the packets discarded by the application. By adopting our scheme, the number of packets discarded by the application is drastically reduced, while the number of packet discarded at nodes is slightly increased.

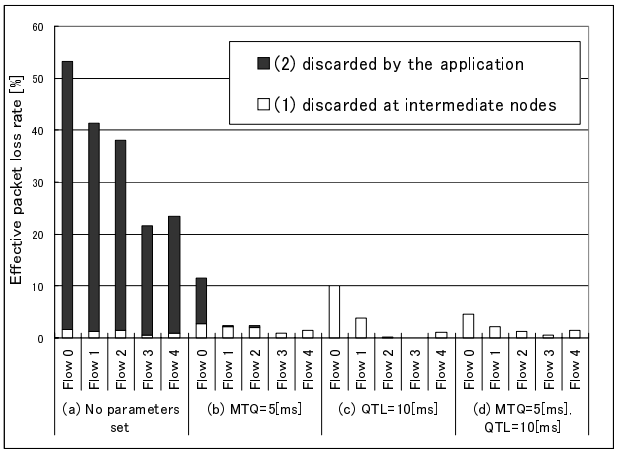


Figure 7.7: Proportion of EPLR cased by application-side and network-side

Table 7.3 presents the number of packets discarded at each nodes due to the buffer overflow or MTQ(1*), due to the QTL(2*), the sum of packets discarded at all nodes(3*) and the sum of packets discarded at the receiver-side application due to violating its delay limitation(4*). In case (a) where the no parameters set, the number of packets discarded at each node(3*) is small, while the number of packets discarded at the application(4*) becomes

large. In case (b), the number of discarded packet decreases in descendant (downstream) nodes, while a considerable number of packets are discarded by the application (4*). On the other hand, in case (c), the number of discarded packets increases especially at descendant nodes, while no packets are discarded by the application. Finally, in case (d) where adopting MTQ and QTL simultaneously, the number of discarded packets at each node is relatively balanced and the total number of discarded packet is minimized.

In addition, average queuing delay at each node is presented in Table 7.4. We found that MTQ, in case (b), seems to evenly reduce the average queuing delay at all nodes, while QTL, in case (c), improves that at descendant nodes. Thus, the degree of improvement in the delay characteristics will be high by adopting both parameters simultaneously.

Table 7.3: No. of packet losses at each node

	node1	node2	node3	3*	4*
	1*/2*	1*/2*	1*/2*		
(a)	10970/-	7527/-	5440/-	23937	687194
(b)	16011/-	11460/-	8731/-	36202	38105
(c)	-/13116	-/22520	-/24527	60163	0
(d)	16011/0	11460/0	4671/5645	37787	0

Table 7.4: Average queuing delay at each node

	node 1 [ms]	node 2 [ms]	node 3 [ms]
(a)	4.24	3.92	3.76
(b)	1.52	1.28	1.20
(c)	2.64	1.36	0.88
(d)	1.52	1.28	0.96

For each setting, the distribution of the end-to-end delay (i.e. the total delay along the path) for packets in flow group 0 is plotted in Fig.7.8. By setting QTL to 10 [ms], the end-to-end delay would be limited to 150 [ms] (the fixed delay of 140 [ms] plus the maximum queuing delay of 10 [ms]). By applying our scheme, the proportion of packets experiencing shorter delays would increase.

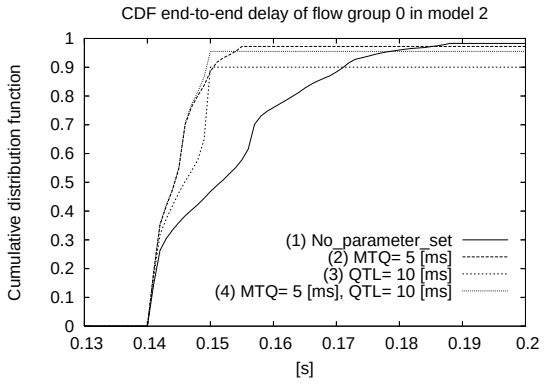


Figure 7.8: Cumulative distribution function for end-to-end-delay in flow group 0

7.3.4 Summary in homogeneous environments

In Fig. 7.9, we will summarize the effectiveness of using MTQ alone, QTL alone, and a combination of both. Figure 7.9 (a) shows the EPLR for each flow group in model 1 when the acceptable queuing delay is 10 [ms], while Figs. 7.9 (b) and (c) show cases where the acceptable total queuing delays are 10 and 20 [ms] in model 2 in congested networks. Obviously, in all of these scenarios, the EPLR is reduced drastically by setting MTQ and/or QTL (cases (2) through (5) in the figures) compared with case (1) where neither MTQ nor QTL is used.

Since the optimal MTQ is not always easily found, case (2) is not always realistic even though they exhibit the best performance. The case with QTL only (case (4)) can be improved by adding MTQ (case (5)), while the case with MTQ only (case (3)) can also be improved by adding QTL(case (5)), in terms of reducing the EPLR of the worst flow group 0 and balancing EPLRs of all flow groups. In other words, the worst flows can be improved at the expense of performance deterioration in the other good flows. It can be concluded that combining QTL and MTQ leads to a relatively good overall delay performance when QTL is set exactly to the acceptable total queuing delay limit and MTQ is set to some non-optimal value smaller than the limit, e.g. 50% of the limit.

Furthermore, we examine other scenarios in which the network is not so congested. We

can find that proposed scheme is effective to improve the delay characteristics of real time flows also in those scenarios. Figure 7.9 (d) shows the EPLR for each flow group when the traffic load is relatively light as shown in Table 7.1(c). We also examine less congested two scenarios where the number of flows in each flow group is 40 (average link utilization is 48 %) or 60 (that is 72 %). In both scenarios, EPLR becomes zero for each flow group regardless of MTQ and/or QTL, i.e., in all five cases. Thus, our proposed scheme is not harmful (even useless) in lightly congested networks.

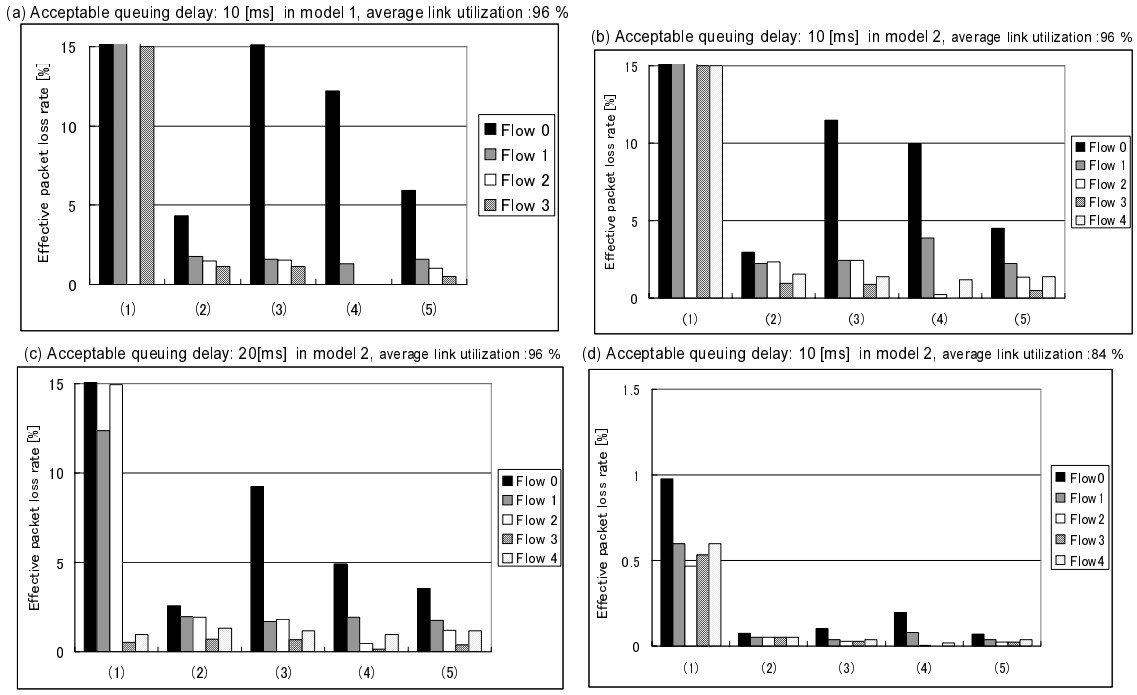


Figure 7.9: Comparison of *EPLR* (effective packet loss rate) in using none, MTQ alone, QTL alone, and MTQ plus QTL

7.4 Simulation results in heterogeneous environments

We examine our mechanisms in three heterogeneous environments using model 2. Subsection 7.4.1 deals with heterogeneity in terms of traffic volume over different paths/links,

Table 7.5: Parameter list of Figure 7.9

	Parameters [ms]	
Case	(a), (b), (d)	(c)
(1)	No parameters set	
(2)	MTQ : 3.3	MTQ : 6.6
(3)	MTQ : 5	MTQ : 10
(4)	QTL : 10	QTL : 20
(5)	MTQ : 5, QTL : 10	MTQ : 10, QTL : 20

where the number of flows on each different path is not the same. Subsections 7.4.2 and 7.4.3 examine configurations with heterogeneity with respect to the acceptable queuing delay for different flows. In subsection 7.4.2, flows on the same path have different acceptable queuing delay requirements, while in subsection 7.4..3 flows on different paths have different acceptable queuing delays. To ensure the marginal quality of every flow, we try to find an appropriate setting of MTQ/QTL so as to improve worst flow EPLR even if there will be different types or conditions of flows.

7.4.1 Different number of flows on different paths

We consider a scenario such that the number of flows for each flow group is different as listed in Table 7.6. The number of competing flows at each link (240, 220, and 200), and thus the utilization also, decreases link by link from the flow source to the sink. Case (5) in Fig. 7.2 shows the EPLR observed for each flow group for an acceptable queuing delay of 10 [ms] when neither MTQ nor QTL is used. This indicates that only node 1 (link $1 \rightarrow 2$) is so congested that only the flow groups which compete with one another at that node (i.e. groups 0, 1, and 4) suffer from an equally large EPLR.

Figure 7.10 shows the relationship between the parameter settings of MTQ/QTL and the EPLR of each flow group for an acceptable queuing delay of 10 [ms]. Figures 7.10(a), (b), and (c) show results for scenarios using MTQ alone, QTL alone, and QTL (set to 10 [ms]) plus MTQ, respectively.

Table 7.6: No. of flows in model 2

Congested link	Flow id(fid)					No. of flows	Total rate [Mbps]
	0	1	2	3	4		
$1 \rightarrow 2$	60	80			100	240	19.2
$2 \rightarrow 3$	60	80	80			220	17.6
$3 \rightarrow 4$	60		80	60		200	16.0

In Fig.7.10(a), the range of suitable values for MTQ is different to that observed in Fig. 7.4(b) for the homogeneous case. This can be explained as follows. Consider a packet in flow group 0 at the first node, which experiences a queuing delay larger than the upper bound (3.3 [ms]) of the appropriate MTQ value range for the homogeneous case, but much smaller than the acceptable queuing delay limit (10 [ms]). Even with this delay, additional delays at the following nodes 2 and 3 are unlikely to make the packet ultimately exceed the delay limit, because these nodes are less congested than the first node 1. Similar arguments can be applied to the EPLRs for flow groups 1 and 2. This result implies that MTQ values should be carefully set depending on various network conditions.

The optimal value for QTL seen in Fig. 7.10(b), on the other hand, is 10 [ms], which is identical to the optimal value in Fig. 7.5(b). From Fig.7.10(c) we can further conclude that using QTL together with MTQ can effectively reduce the EPLR for a wide range of MTQ values, even in this heterogeneous scenario.

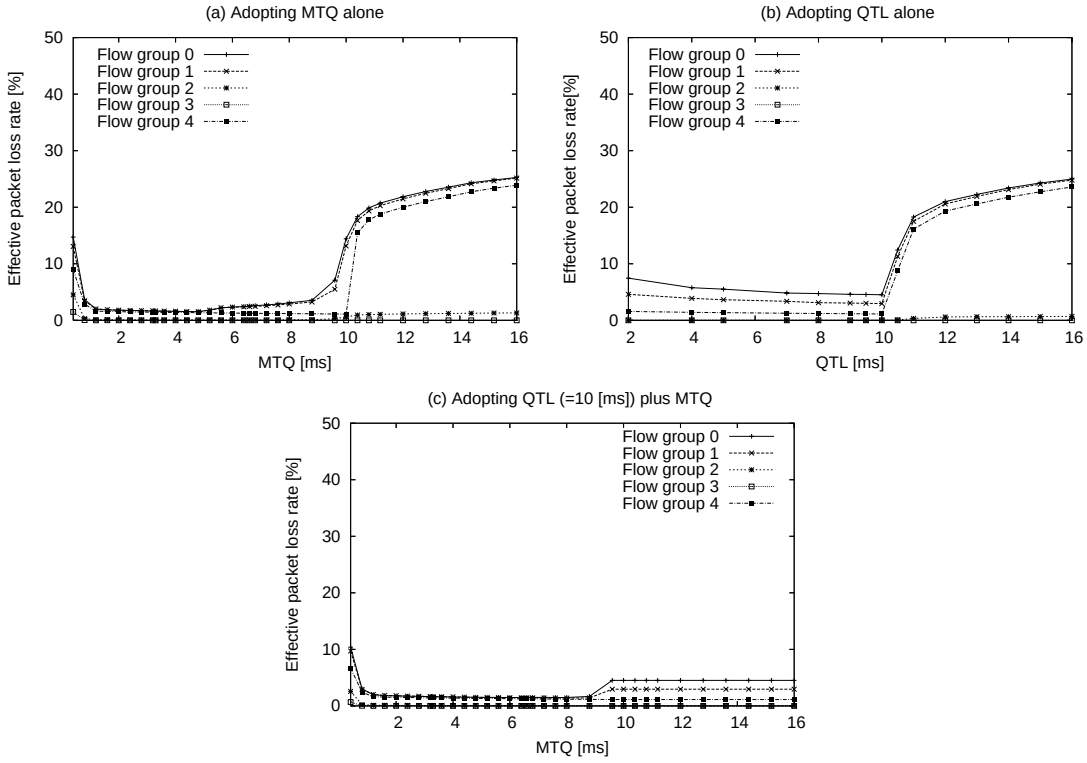


Figure 7.10: *EPLR* (effective packet loss rate) when the number of flows in each group is different, acceptable queuing delay: 10 [ms]

7.4.2 Different delay requirements of flows on the same path

Consider a scenario such that the flows in each group in Fig. 7.1(b) are divided into two subgroups, referred to as subgroups 1 and 2, each of which has 40 flows. As listed in Table 7.7, the flows in subgroup 1 have a stricter delay requirement than those in subgroup 2.

Figures 7.12 show the EPLR for each subgroup. Note that when our scheme was not used, very high EPLRs were observed in this congested network, e.g. over 50 % for subgroup 1 in flow group 0, over 40 % for subgroup 1 in flow group 1, which can be seen the EPLRs at MTQ value of 16[ms] in Fig. 7.11

Table 7.7: Acceptable queuing delay limit for each subgroup flow

Flow group	Subgroup	Indication	Acceptable queuing delay [ms]
0	1	f0-1	10
	2	f0-2	20
1	1	f1-1	10
	2	f1-2	20
2	1	f2-1	10
	2	f2-2	20
3	1	f3-1	10
	2	f3-2	20
4	1	f4-1	10
	2	f4-2	20

In Fig. 7.12, cases (1)–(4) present the EPLR results using MTQ alone, where several tactics for setting MTQ are examined. First we investigated the case where all flows have an identical MTQ value, with results shown in Fig. 7.11. We found an optimal value for MTQ of 3.3 [ms], which is the same as in the homogeneous case illustrated in Fig. 7.4(b). The EPLR for each flow using this MTQ setting is shown in case (1) of Fig. 7.12, which performs the best of all.

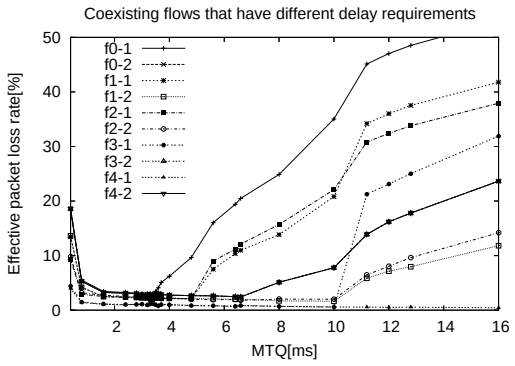


Figure 7.11: *EPLR* (effective packet loss rate), using MTQ alone

In case (2), the MTQ is set differently in each subgroup such that an optimal value for

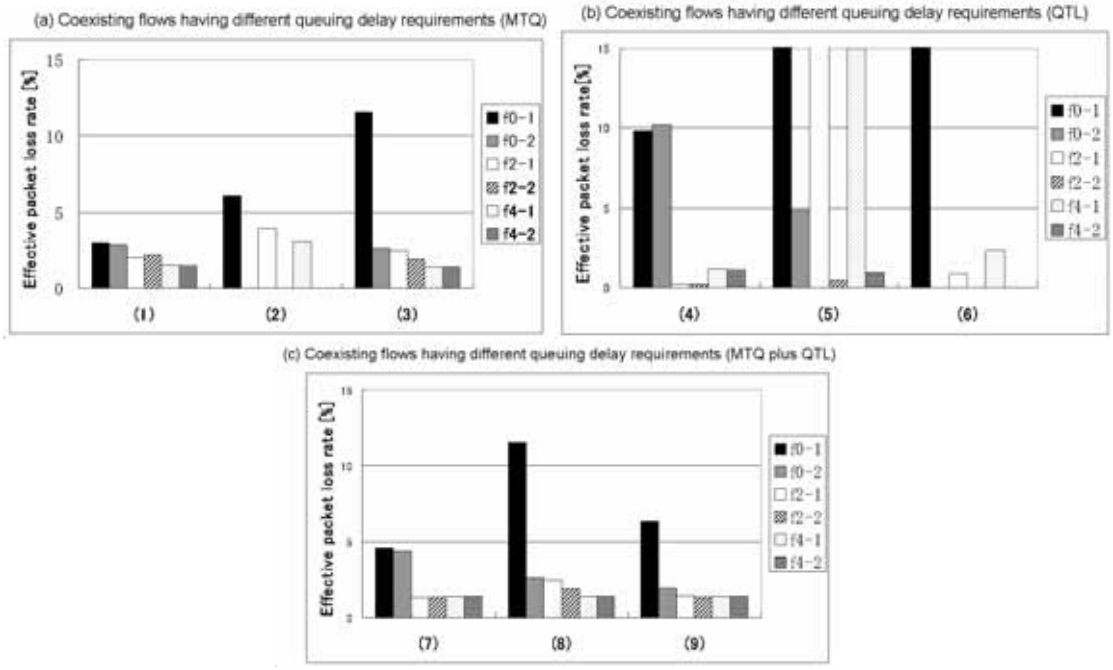


Figure 7.12: *EPLR* (effective loss rate) when coexisting flows on the same path have different queuing delay requirements

Table 7.8: Parameter list of Fig. 7.12

Cases	parameters [ms]
(1)	f0-1–f4-2 : MTQ=3.3
(2)	f0-1,f1-1,f2-1,f3-1,f4-1 : MTQ=3.3
	f0-2,f1-2,f2-2,f3-2,f4-2 : MTQ=6.6
(3)	f0-1,f1-1,f2-1,f3-1,f4-1 : MTQ=5
	f0-2,f1-2,f2-2,f3-2,f4-2 : MTQ=10
(4)	f0-1–f4-2 : MTQ=5.0
(5)	f0-1–f4-2 : QTL=10
(6)	f0-1–f4-2 : QTL=20
(7)	f0-1,f1-1,f2-1,f3-1,f4-1 : QTL=10
	f0-2,f1-2,f2-2,f3-2,f4-2 : QTL=20
(8)	f0-1–f4-2 : MTQ=5, QTL=10
(9)	f0-1,f1-1,f2-1,f3-1,f4-1 : MTQ=5, QTL=10
	f0-2,f1-2,f2-2,f3-2,f4-2 : MTQ=5, QTL=20

each individual flow in the homogeneous case is applied. Thus, the MTQ setting is 3.3 [ms] for the flows having a strict delay limit (in subgroup 1), and 6.6 [ms] for the non-strict flows (in subgroup 2). It can be clearly seen that the strict flows suffer greatly in competing with the non-strict flows, because whenever the queue length increases to more than 3.3 [ms], all packets in the strict flows will be discarded while those in the non-strict flows will not. Compared with case (1), EPLR of subgroup 2 is improved, at the expense of degrading subgroup 1's EPLR.

In case (4), the MTQ is set to be the same in all flows as some non-optimal but moderate (i.e. conservative) value, say 50 % of the acceptable queuing delay for the strict flows in subgroup 1. Similarly to case (2), case (3) set the MTQ differently to each subgroup where a MTQ value larger than the optimal one for each individual flow is applied. These results indicate that when MTQ alone is applied, to achieve an overall good EPLR all flows should have the same MTQ value set based on their strictest queuing delay requirement, that is the value optimal to the strictest flows in the homogeneous case.

Cases (5)–(7) of Fig. 7.12 display the results for when only QTL is applied. In case (5), the QTL is set to be the same in all flows as the optimal value for the strict flows, that is the acceptable queuing delay of 10 [ms] for flow subgroup 1. In case (6) however, the QTL is set for all flows to be the same as the optimal value for the non-strict flows (subgroup 2). In case (7), the QTL is set differently for each subgroup such that the subgroup's acceptable queuing delay is applied, that is 10 [ms] for the strict subgroup 1 flows, and 20 [ms] for the non-strict subgroup 2 flows. As in case (2) for MTQ, the strict flow performance is degraded through competition with the non-strict flows. These results show that when QTL only is used, all flows should be set to the same QTL value based on their strictest queuing delay requirement.

Cases (8)–(9) of Fig. 7.12 show the EPLR where both MTQ and QTL parameters are set. In these cases, we assumed that the optimal value for MTQ was unknown, so instead an MTQ value of 50% of the strictest acceptable queuing delay requirement was used, i.e. 5.0

[ms]. The QTL is set to be the same in all flows based on the delay requirement for subgroup 1 in case (8). In case (9) however, the QTL is set differently in each subgroup so that the subgroup's own delay requirement is applied.

We investigate the main cause of packet discarding on each flow group at intermediate nodes in cases (8) and (9). As illustrated in Fig.7.13, the QTL-based packet loss (i.e. more than 10[ms] cumulative queuing delay experienced by a packet) occurred only on flow group 0 while the MTQ-based packet loss (i.e. more than 5 [ms] local queuing delay experienced by a packet) appeared on every flow group. This is because only flow group 0 competes with other flow groups at more than two nodes along its path where the maximum queuing delay at each node could not exceed 5[ms]. Comparing results in Fig. 7.12 seems to indicate that

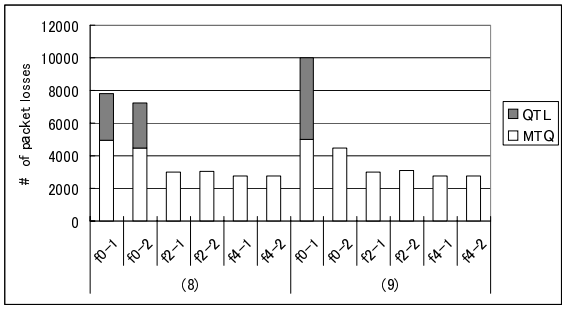


Figure 7.13: Proportion of network-based packet losses by QTL and MTQ

case (8) shows the best performance in terms of ensuring the marginal quality of every flow and that even when both QTL and MTQ are used, all flows should have the same QTL value set equal to the strictest acceptable queuing delay, i.e. 10 [ms] in this case.

7.4.3 Different delay requirements of flows on different paths

For another type of heterogeneity, we consider a scenario in which each flow group traversing each different path has a different delay requirement, as listed in Table 7.9. Here, flow groups 0, 3, and 4 have a strict acceptable queuing delay limit, while flow groups 1 and 2 obey a

non-strict delay limit. First we investigated the situation in which all flows have an identical MTQ value, as shown in Fig. 7.14. We found that in terms of averaged EPLR over all flows, the optimal value for MTQ is 3.3 [ms], which is again the same value as observed in Figs. 7.4 and 7.11.

Table 7.9: Acceptable queuing delay limit for all subgroup flows

Indication	Flow group no.	Acceptable queuing delay [ms]
f0	0	10
f1	1	20
f2	2	20
f3	3	10
f4	4	10

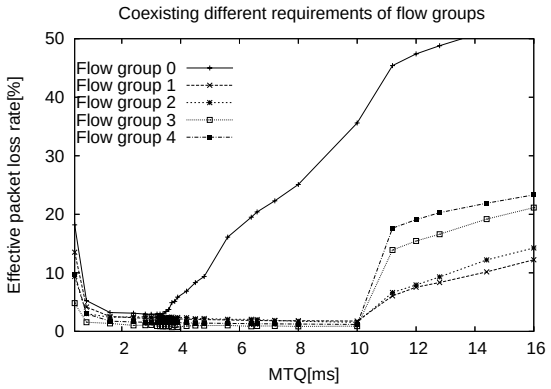


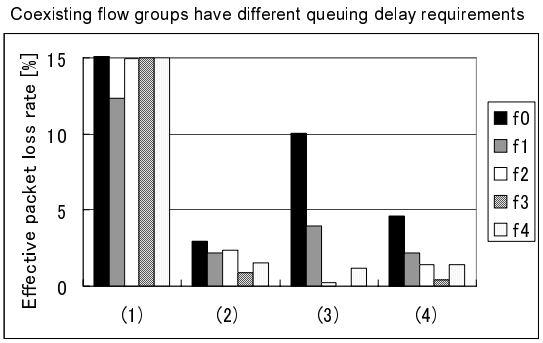
Figure 7.14: *EPLR* (effective packet loss rate), using MTQ alone

Figure 7.15 shows the EPLR in each flow group for four cases: (1) using neither MTQ nor QTL, (2) MTQ only (optimal value), (3) QTL only (optimal value), and (4) MTQ (moderate value) plus QTL (optimal value). These results are consistent with those obtained in the previous scenarios, and it can be seen that (at least in heavily congested networks),

1. MTQ/QTL can significantly improve the EPLR of all flows;
2. setting MTQ to the optimal value achieves the best result in terms of the averaged EPLR over all flows, although the optimal value may not always be easy to determine;

3. applying QTL alone (even when set to the optimal value) is not always effective in improving the EPLR of the worst case flows (e.g. flow group 0);

4. using MTQ and QTL together can produce very good EPLR performance when the QTL value is optimal (that is, the strictest acceptable queuing delay), even if the MTQ value is non-optimal (say, 50% of the strictest acceptable queuing delay).



Parameter list of Fig. 7.15

No.	Parameters [ms]
(1)	Set no parameters
(2)	f0–f4 : MTQ=3.3
(3)	f0–f4 : QTL=10
(4)	f0–f4 : MTQ=5, QTL=10

Figure 7.15: *EPLR* (effective packet loss rate) when coexisting flows on different paths have different queuing delay requirements

7.5 Concluding Remarks

In this chapter, an adaptive scheme was proposed for earlier discarding of packets in real-time application flows by using two mechanisms (MTQ and QTL). In the scheme, a packet experiencing too great a queuing delay is discarded at intermediate nodes based on a limit

for the total queuing delay the packet is experiencing along the path (QTL) and/or a limit for the local queuing delay the packet is experiencing at each node (MTQ).

The approach was evaluated through network simulations in both homogeneous and heterogeneous environments. In the homogeneous scenarios, the number of flows on each path was identical and all flows had an identical queuing delay requirement (i.e. limit). Meanwhile in the heterogeneous cases, the number of flows on each different path was not the same or the flows coexisting on the network had different delay requirements. Using an appropriate MTQ value effectively improved delay characteristics (i.e. reduced EPLR), but it was generally not easy to determine such a suitable value for MTQ. Delay characteristics were also improved by using QTL alone, with a value set to the acceptable queuing delay limit for flows with the strictest requirements; however there may still be room for improvement when the delay limit is relatively small compared with the length of the queuing buffer at intermediate nodes. By using QTL plus MTQ appropriately, i.e. setting QTL to the strictest acceptable queuing delay limit and MTQ to some value smaller than the delay limit (e.g. 50% of the delay limit), good overall delay performance was achieved.

Experimental results for the proposed scheme in basic scenarios have been shown, and these indicate that the scheme has great potential in improving the queuing delay performance of real-time flows in a congested network. However, further investigation and enhancement are necessary in order to develop practical systems based on the scheme. To clarify the proper scope and limitations of the scheme and to allow it to be used reliably, more theoretical analysis may be needed, as well as more extensive simulations or real world experiments. From such future investigations, a more qualitative relationship could be established between the delay requirement (the acceptable queuing delay), the achievable performance (the EPLR), and the network environment (e.g. flow topology, traffic burstiness, and link utilization).

Real networks generally have a more complex meshed topology, involving a number of network nodes and with a large heterogeneity in both traffic characteristics and delay

requirements. When applying the scheme to such networks, more sophisticated tuning of the MTQ parameter may be required in order to achieve a satisfactory balance in overall delay performance. To do this, the fairness of the queuing delay performance over all flows in the network should be addressed quantitatively. The MTQ mechanism presented in this paper is quite a simple version that just results in limiting the queuing buffer length equally at every node the packets traverse. Although this is shown to be effective in basic scenarios, a possible enhancement to fully exploit the potential of hop-by-hop processing for each packet is to make MTQ values more dynamic, whereby the value can be changed hop-by-hop like QTL. This could provide more flexibility to cope with greater traffic heterogeneity. This kind of enhancement, however, would also need to take cost and scalability into consideration.

Chapter 8

Concluding Remarks

8.1 Summary

In the present thesis, the performance of various congestion controls conducted between end hosts and at intermediate nodes was evaluated in order to realize various end-to-end communication qualities that meet the diversity requirements observed of the network. Various evaluations on JGNII and a network simulator revealed that individualized congestion control are necessary in order to satisfy various application and diversity requirements of the network. The ultimate goal of the present research is to find a feasible and cost-effective method for the future Internet comprising shared heterogeneous networks to simultaneously provide high-throughput data transfer as well as various types of application traffic. Evaluation on JGNII and a network emulator using new management mechanisms for the intermediate nodes might be a necessary approach to find the solution.

First, in Chapter 4, I investigated the throughput characteristics of a variety of high-speed transport protocols (HSTCP, Scalable TCP, FAST, and SABUL) recently proposed for transmitting a huge amount of data on fast long-distance networks, through experiments on the Japan Gigabit Network. All protocols treated in the present experiments exhibited good throughput performance compared to Standard TCP. These throughput performances

are listed in from best to worst as follows: SABUL > FAST > Scalable TCP > HSTCP, whereas the likelihood of packet loss are as follows: SABUL > Scalable TCP > FAST > HSTCP. The sending rates of SABUL and FAST could be adjusted rapidly and appropriately in response to dynamic changes in competitive UDP traffic. For TCP-based protocols, the performance was affected considerably by the type of receiver-side OS due to the difference in strategy of sending back ACK packets and the ability of the SACK option. The fact that SABUL and FAST outperform implies that the conventional congestion avoidance mechanism of TCP based on receiving ACK packets may not be suitable for fast long-distance networks. For high-speed transport protocols, it seems desirable to separate the congestion control from the error control, and furthermore to make the rate control smooth and rapid. However, all protocols treated herein suffered from the problem of unfairness or undesirable interference when multiple connections of different protocols shared a common link. Further progress in high-speed transport protocols is needed to solve this problem.

In Chapter 5, the performance of multiple high-speed transport protocol connections was investigated through experiments in various network environments, including an open 10-Gbps-class network testbed between US and Japan. In addition, the use of high-speed transport protocols on the global Internet is currently being investigated. The obtained results (with 1-Gbps end-hosts) indicated that, when long-lived data transfer flows of high-speed transport protocols run in conjunction with even small amounts of Internet application traffic, such as short-term web browsing flows and long-term video streaming flows, the performance of not only web access and video streaming but also of the high-speed transfer flows is degraded. In other words, such circumstances are neither effective nor efficient in terms of bandwidth sharing. The ultimate goal is to find a feasible and cost-effective method of bandwidth sharing on the future Internet comprising shared heterogeneous networks to simultaneously provide high-throughput data transfer as well as various types of application traffic. At present, there is no silver bullet with which to solve this problem. However, several interesting characteristics of this problem, i.e., the pros and cons of each high-speed

transport protocol in each typical realistic situation, are being investigated. By sharing the insights obtained from a variety of experimental results for a variety of realistic network conditions, it is hoped that improved high-speed data transport protocols and/or new management mechanisms will be developed for the intermediate nodes.

The preliminary results seem to indicate that the slow-start is relevant to high-throughput transfer in fast long-distance networks. When no other TCP flows coexist, the original slow-start leading to exponential cwnd growth is effective for promptly obtaining the maximum throughput. However, when multiple TCP flows compete with each other, the collision of multiple slow-start results in extremely unstable conditions. To manage both the prompt acquisition of high throughput and the stable competition with other flows, the slow-start mechanisms should be investigated in greater detail and should be modified for use on fast long-distance networks.

In Chapter 6, the throughput behavior of TCP flows over multiple AF DiffServ domains was investigated. In particular, we focused on the influence of packets re-marked at the domain boundary on the throughput characteristics and were able to clarify the following. (1) Re-marked packets at a DiffServ domain boundary can degrade the end-to-end throughput, i.e., the quality of service. (2) Applying an appropriate marking policy, (e.g., the original_mark policy presented in Section 3.1, which is based on a contracted rate) at the ingress node in each DiffServ domain can reduce the number of re-marked packets. (3) The network configuration of an intra-domain has less effect on throughput degradation.

In Chapter 7, an adaptive scheme was proposed for earlier discarding of packets in real-time application flows by using two mechanisms (MTQ and QTL). In this scheme, a packet that is experiencing too large a queuing delay is discarded at an intermediate node based on a limit for the total queuing delay experienced by the packet along the path (QTL) and/or a limit for the local queuing delay experienced by the packet at each node (MTQ). The approach was evaluated through network simulations in both homogeneous and heterogeneous environments. In the homogeneous scenarios, the number of flows on each path was iden-

tical, and all flows had an identical queuing delay requirement (i.e., limit). Meanwhile, in heterogeneous cases, the number of flows on each path was not the same, or the flows co-existing on the network had different delay requirements. Using an appropriate MTQ value effectively improved the delay characteristics (i.e. reduced EPLR), but it was generally not easy to determine such a suitable value for MTQ. The delay characteristics were also improved by using QTL alone, with a value set to the acceptable queuing delay limit for flows with the strictest requirements. However, there may still be room for improvement when the delay limit is relatively small compared with the length of the queuing buffer at intermediate nodes. By using QTL plus MTQ appropriately, i.e., by setting QTL to the strictest acceptable queuing delay limit and MTQ to some value smaller than the delay limit (e.g., 50% of the delay limit), good overall delay performance was achieved. Experimental results in basic scenarios indicate that the proposed scheme has great potential for improving the queuing delay performance of real-time flows in a congested network. However, further investigation and enhancement are necessary in order to develop practical systems based on the proposed scheme. To clarify the proper scope and limitations of the proposed scheme and to allow the proposed scheme to be used reliably, more theoretical analysis may be necessary, as well as more extensive simulations and real-world experiments.

8.2 Future research

In this thesis, a number of issues regarding congestion control in high-speed networks were examined. In the future, the following areas will be investigated.

The performance of high-speed transport protocols that work between end hosts was investigated in the preset research. These results indicate that it is difficult to realize efficiency and fairness via existing high-speed transport protocols. The key to realizing these properties is the inference of the internal state of the network. HSTCP and Scalable TCP use packet loss information as a trigger for controlling cwnd. Therefore, they update cwnd after the detection of packet loss. FAST uses delay information as a trigger for controlling cwnd. On

the other hand, CUBIC and the HTCP protocol use the elapse time from the last packet loss as a trigger for controlling cwnd. To infer the internal status of a network, it may be useful to use information both on the delay and the packet loss, as in Compound TCP, proposed most recently and was not targeted in the research because there were no implementation available. In addition, observing historical trends of packet losses and delay might also be useful for inferring the network status, for example, the minimum value for RTT and the elapse time from the last packet loss event might be stored. Observing the performance of high-speed transport protocols through experiments over JGNII, the requirements and limitations of congestion control will be investigated further by conducting these controls between end hosts.

The preliminary experimental results on the JGN2 seem to indicate that the slow-start is relevant to high-throughput transfer in fast long-distance networks. When no other TCP flows coexist, the original slow-start leading to exponential cwnd growth is effective to promptly obtain the maximum throughput. However, when multiple TCP flows compete with each other, the collision of multiple slow-starts results in extremely unstable conditions. This will be considered further in order to improve the behavior of the slow-start phase.

Experimental results presented for the proposed scheme, MTQ and QTL, in basic scenarios indicate that the proposed scheme has great potential for improving the queuing delay performance of real-time flows in a congested network. However, further investigation and improvements are necessary in order to develop practical systems based on the proposed scheme. In order to clarify the proper scope and limitations of the proposed scheme and to allow the proposed scheme to be used reliably, further theoretical analysis may be needed, as well as more extensive simulations or real-world experiments. Based on such future investigations, a more qualitative relationship could be established between the delay requirement (acceptable queuing delay), the achievable performance (EPLR), and the network environment (e.g., flow topology, traffic burstiness, and link utilization). Real networks generally have a more complex meshed topology, involving a number of network nodes and having

a large heterogeneity in both traffic characteristics and delay requirements. When applying the proposed scheme to such networks, more sophisticated tuning of the MTQ parameter may be required in order to achieve a satisfactory balance in overall delay performance. To do this, the fairness of the queuing delay performance over all flows in the network should be addressed quantitatively. The MTQ mechanism presented herein is a simple version that limits the queuing buffer length equally at each node traversed by a packet. Although this is shown to be effective in basic scenarios, a possible enhancement to fully exploit the potential of hop-by-hop processing for each packet is to make MTQ values more dynamic, whereby the value can be changed hop-by-hop, as in QTL. This could provide more flexibility to cope with greater traffic heterogeneity. However, the greater cost and scalability that would be incurred by this type of enhancement should be taken into consideration.

Bibliography

[ABB+02] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, S. Meder, and S. Tuecke, "GridFTP : Protocol Extensions to FTP for the Grid", GridForum Draft, April 2002.

[AC06] J. J. Alcaraz and F. Cerdan, "Slope based discard: A buffer management scheme for 3G links supporting TCP traffic", in *International Wireless Communications and Mobile Computing Conference*, July 2006.

[AD] S. E. Agarwal and D.A., "HighSpeed TCP Study: Characteristics and Deployment Issues", Technical report.

[ADD00] I. Alves, J. DeRezende, and L. Demoraes, "Evaluating Fairness in Aggregated Traffic Marking", in *Proc. of Globecom'2000*, Nov. 2000.

[All03] M. Allman, "RFC 3465: TCP Congestion Control with Appropriate Byte Counting(ABC)", IETF(Experimental), Feb. 2003, URL: ⟨http://www.rfc-editor.org/rfc/rfc3465.txt⟩.

[APS99] M. Allman, V. Paxson, and W. Stevens, "RFC 2581: TCP Congestion Control", IETF, April 1999, URL: ⟨http://www.rfc-editor.org/rfc/rfc2581.txt⟩.

[aut] "TCP Tuning Guide", URL: ⟨http://www-didc.lbl.gov/TCP-tuning⟩.

[BBC+98] S. Blake, D. Black, M. Carlson, E.Davies, Z. Wang, and W. Weiss, "RFC 2475: An Architecture for Differentiated Services", IETF, Dec. 1998, URL: ⟨http://www.rfc-editor.org/rfc/rfc2475.txt⟩.

[bc] "Lambda Joins Award Winning Grid Demo at SC2002", URL: ⟨http://www.gridtoday.com/02/1209/100861.html⟩.

[BCC+98] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Parridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang, "Recommendation on queue management and congestion avoidance in the Internet", in *RFC2309*, April 1998.

[BCRJ04] H. Bullot, R.L. Cottrell, and R.Hughes-Jones, "Evaluation of Advanced TCP Stacks on Fast Long-Distance Production Networks", in *Proc. of PFLDnet2004*, Feb. 2004.

[CAK+05] R.L. Cottrell, S. Ansari, P. Khandpur, R. Gupta, R. Hughes-Jones, M. Chen, L. McIntosh, and F. Leers, "Characterization and evaluation of TCP and UDP-based transport on real networks", in *Proc. of PFLDnet2005*, Feb. 2005.

[CF98] D. Clark and W. Fang, "Explicit Allocation of Best Effort Packet Delively Service", *Transactions on Networking, to appear*, Vol. 6, No. 4, pp. 362–373, August 1998.

[CKKW] K. Chang, V. Kapoor, E. Kusmierek, and C. Wighe, "Comparison of Packet Marking schemes in Differentiated Service", University of Minnesota, Department of CSE Technical Report CS-8221.

[CL03] D. H. Choe and S. H. Low, "Stabilized vegas", in *Proceedings of IEEE Infocom*, April 2003.

[ETMK99] T. Ebata, M. Takihiro, S. Miyake, and M. Koizumi, "Inter-Domain QoS Provisioning and Accounting", IETF work in progress, Oct. 1999.

[Fan] URL: ⟨http://www.cs.priceton.edu/wfang/papers.html.ns-riotsw.tar.gz⟩.

[Fan99] W. Fang, "The "Expected Capacity" Framework : Simulation Results", Princeton Univ Technical Report, TR-601-99, 1999.

[FGE03] W. Feng, M. K. Gardner, and E.Weigle, "Automatic Flow-Control Adaptation for Enhancing Network Performance in Computational Grids", *Journal of Grid Compiting*, Vol. 1, No. 1, pp. 63–74, Nov. 2003.

[Fio00] Pierre M. Fiorini, "Voice over IP (voip) for enterprise networks: Performance implications and solutions", in *International CMG Conference*, pp. 545–556, 2000.

[FJ93] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance", *IEEE/ACM Transactions on Networking*, Vol. 1, No. 4, pp. 397–413, August 1993.

[Flo03] S. Floyd, "RFC 3649: HighSpeed TCP for Large Congestion Windows", IETF, Dec. 2003, URL: ⟨http://www.rfc-editor.org/rfc/rfc3649.txt⟩.

[FP00] W. Feng and P.Tinnakornsrisuphap, "The Failure of TCP in High-Performance Computational Grids", in *Proc. of SC2000*, Nov. 2000.

[FPes] W. Fang and L. Peterson., "TCP mechanisms for Diffserv Architecture", Technical Report TR-605-99, September 1999; 12 Pages.

[FSN00] W. Fang, N. Seddigh, and B. Nandy, "RFC 2859: A Time Sliding Window Three Colour Marker(TSWTCM)", IETF, June. 2000, URL: ⟨http://www.rfc-editor.org/rfc/rfc2859.txt⟩.

[FTD03] C. Fraleigh, F. Tobagi, and C. Diot, "Provisioning IP backbone networks to support latency sensitive traffic", in *INFOCOM2003*, 2003.

[GDJL00] M. Goyal, A. Durresi, R. Jain, and C. Liu, "Performance Analysis of Assured Forwarding", IETF draft, Feb. 2000.

[GG03] Y. Gu and R. L. Grossman, "SABUL: A Transport Protocol for Grid Computing", *Journal of Grid Computing*, Vol. 1, pp. 377–386, 2003.

[GG05] Y. Gu and R. L. Grossman, "Optimizing UDP-based Protocol Implementation", in *Proc. of PFLDnet2005*, Feb. 2005.

[GGP97] L. Georgladls, R. Guerin, and A. Parekh, "Optimal multiplexing on a single link: Delay and buffer requirements", *IEEE Transactions on Information Theory*, Vol. 43, No. 5, pp. 1518–1535, September 1997.

[HBWW99] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski, "RFC 2597: Assured Forwarding PHB Group", IETF, June 1999, URL: ⟨http://www.rfc-editor.org/rfc/rfc2597.txt⟩.

[Her99] S. Herzog, "Inter-Domain Policy Architeture", Presentation of aaaarch RG at 47th IETF Meeting, Jan. 1999.

[Hir06] M. Hirabaru, "Impact of Bottleneck Queue Size on TCP Protocols and Its Measurement", *IEICE TRANS COMMON*, Vol. E89-B, No. 1, Jan. 2006.

[HKL+06] Sangtae Ha, Yusung Kim, Long Le, Injong Rhee, and Lisong Xu, "A Step toward Realistic Performance Evaluation of High-Speed TCP variants", in *Proc. of PFLDnet2006*, Feb. 2006.

[Hus00] G. Huston, "RFC 2990:Next Step for the IP QoS Architecture", IETF, Nov. 2000, URL: ⟨http://www.rfc-editor.org/rfc/rfc2990.txt⟩.

[IEP] URL: ⟨http://www.ietf.org/proceeding/01dec/minutes/IEPREP⟩.

[IN98] J. Ibanez and K. Nichols, "Preliminally Simulation Evaluation of an Assured Service", presented in IETF meeting, Aug. 1998.

[inj] "BIC LAB, Network Protocol Testing Tool and Protocol testing scenarios and environment", URL: ⟨http://www.hpcc.jp/pfldnet2006/slides/p2-03.pdf⟩.

[ipe] "Iperf", URL: ⟨http://dest.nlanr.net/Projects/Iperf⟩.

[JN98] Ibenz J. and K. Nichols, "Preliminaly Simulation Evaluation of an Assured Service", Technical report, 1998.

[JWL04] C. Jin, D.X. Wei, and S. H. Low, "FAST TCP : motivation, architectire, algorithms, performance", in *IEEE Infocom*, March. 2004.

[Kel03] T. Kelly, "Scalable TCP: Improving Performance in Highspeed Wide Area Networks", *Computer Communication Review*, Vol. 32, No. 2, April. 2003.

[KHTO04] K. Kumazoe, Y. Hori, M. Tsuru, and Y. Oie, "Transport Protocols for Fast Long-Distance Networks : Comparison of Their Performance in JGN", in *Proc. of SaintWS2004*, Jan. 2004.

[KK99] V. Nichols K. and Poduri K., "RFC 2598: An Expedited Forwarding PHB", IETF, July 1999, URL: ⟨http://www.rfc-editor.org/rfc/rfc2598.txt⟩.

[KMT+05] Y. Kitaguchi, A. Machizawa, M. Tsuru, Y. Oie, and K. Hakozaki, "The advanded network time synchronous system by self-discarding packet technique", *Information Processing Society of Japan*, Vol. 46, No. 4, pp. 1017–1024, April 2005.

[KTO06] K. Kumazoe, M. Tsuru, and Y. Oie, "Improving delay characteristics of real-time flows by adaptive early packet discarding", in *The International Conference on Information Networking*, January 2006.

[LLS] Y.T. Li, D. Leith, and R. N. Shorten, "Experimental evaluation of TCP protocols for high-speed networks", *Transactions on Networking, to appear*.

[lsr] "Internet2 Land Speed Record", URL: ⟨http://lsr.internet2.edu⟩.

[ml] URL: ⟨http://www.postel.org/pipermail/end2end-interest/2003-October/003582.html%⟩.

[MMY04] Y. Matsushita, T. Matsuda, and M. Yamamoto, "A packet-discarding technique achieving fairness between wired and wireless TCP sessions", in *International Conference on Mobile Computing nd Ubiquitous Networking*, January 2004.

[Mur] S. Murphy, "DiffServ additions to ns-2".

[NBBB98] K. Nichols, S. Blake, F. Baker, and D.L. Black, "RFC 2474: Definition of the Differentiated ServiceField(DS Field) in the IPv4 and IPv6 Headers", IETF, Dec. 1998, URL: ⟨http://www.rfc-editor.org/rfc/rfc2474.txt⟩.

[NSP99] B. Nandy, N. Seddish, and P. Pieda, "Diffserv's Assured Forwarding PHB: What Assurance does the Customer Have?", in *Proc. of NOSSDAV*, 1999.

[Pro] VINT Project, "Network Simulator version 2".

[RF95] A. Romanow and S. Floyd, "Dynamics of TCP traffic over ATM networks", *IEEE Journal on Selected Areas In Communications*, Vol. 13, No. 4, pp. 633–641, May 1995.

[RX05] I. Rhee and L. Xu, "CUBIC : A New TCP-Friendly High-Speed TCP Varient", in *Proc. of PFLDnet2005*, Feb. 2005.

[SL04] R.N. Shorten and D. J. Leith, "H-TCP : TCP for high-speed and long-distance networks", in *Proc. of PFLDnet2004*, Feb. 2004.

[SNP99] N. Seddish, B. Nandy, and P. Pieda, "Bandwidth Assurance Issues for TCP flows in a Differentiated Services Network", in *Proc. of Globecom'99*, March 1999.

[SSS03] A. Schmitter, A. Th. Schwarzbacher, and T. D. Smith, "Analysis of network conformity with voice over IP specifications", in *ISSC2003*, pp. 82–86, July 2003.

[tcp] "tcpdump", URL: ⟨http://www.tcpdump.org⟩.

[Tei98] B. Teitelbaum, "Qbone Architecture v1.0", Internet2 QoS Working Group Draft, Aug. 1998.

[TKFO03] M. Tsuru, Y. Kitaguchi, H. Fukuoka, and Y. Oie, "On the practical active network with the minimal functionality", in *The second International Workshop on Active Network Technologies and Applications*, pp. 45–52, May 2003.

[TMW97] K. Thompson, G.J. Miller, and R. Wilder, "Wide-Area Internet Traffic Patterns and Characteristics", *IEEE Network*, Vol. 11, No. 6, Nov.–Dec. 1997.

[tun] "TCP Tuning Guide", URL: ⟨http://www-didc.lbl.gov/TCP-tuning/⟩.

[Veg] "Advanced Protocol Design", URL: ⟨http://www.cs.arizona.edu/protocols⟩.

[wan] "The WAN in Lab (WiL)", URL: ⟨http://wil.cs.caltech.edu/⟩.

[yan] "TCP/IP Transfer Tests", URL: ⟨http://www.hep.ucl.ac.uk/~ytl/tcpip/transfertests/index.html⟩.