Congestion control mechanism of TCP for achieving predictable throughput

Kana Yamanegi   Go Hasegawa   Masayuki Murata
Graduate School of Information Science and Technology, Osaka University
1-5 Yamadaoka, Suita, Osaka 565-0871, Japan
{k-yamanegi, hasegawa, murata}@ist.osaka-u.ac.jp

Abstract

The demand of Internet users for diversified services has increased due to the rapid development of the Internet, and applications requiring QoS guarantee, such as real-time media delivery services, have become popular. Our research group has proposed transport-layer approaches to provide such high-level quality of network services. In the present paper, we propose a congestion control mechanism of TCP for achieving predictable throughput. It does not mean we can guarantee the throughput, while we can provide the throughput required by an upper-layer application at high probability when network congestion level is not so high. We herein present the evaluation results for the proposed mechanism obtained in simulation experiments and confirm that the proposed mechanism can assure a TCP throughput if the required bandwidth is not so high compared to the physical bandwidth, even when other ordinary TCP (e.g., TCP Reno) connections occupy the link.

1 Introduction

The demand by Internet users for network quality has increased due to services becoming increasingly diversified and sophisticated because the remarkable degree to which the Internet has grown, which is due in part to access/backbone network technologies. Applications involving real-time media delivery services, such as VoIP, video streaming and TV meeting system, which require large and stable amounts of network resources in order to maintain Quality of Services (QoS), have experienced a dramatic level of development. For example, the quality of real-time streaming delivery applications is highly dependent on propagation delay and delay jitter. The available bandwidth on the end-to-end network path is also an important factor in order for rich contents, including voice and video, to be provided smoothly.

Some video streaming applications use User Datagram Protocol (UDP) as a transport-layer protocol, and control the data transmission rate by the application program, according to the network condition. However, these mechanisms have a large cost when modifying the application program for achieving the application-specific QoS requirements, and the parameter settings are very sensitive to various network factors. Furthermore, when such applications co-exist in the network and share the network bottleneck resources, we can not estimate the performance of the network or that of the applications, because the control mechanisms of such applications are designed and implemented independently, without considering the effect of interactions with other applications.

In our research group, we proposed transport-layer approaches for achieving QoS for such applications. Since TCP controls the data transmission rate according to the network condition (congestion level), we believe that the transport-layer approach is ideal for providing high-quality data transmission services. Furthermore, by implementing the mechanism into TCP, rather than introducing a new transport-layer protocol or modifying UDP, we can accommodate several existing TCP-based applications transparently , and we can minimize the degree of modification to provide predictable throughput.

In the present paper, we focus on achieving predictable throughput by TCP connections. Essentially, TCP cannot obtain guaranteed throughput because its throughput is dependent on, for example, the Round Trip Time (RTT) and the packet loss ratio of the network path and the number of co-existing flows [1]. Therefore, we intend to increase the probability at which a TCP connection achieves the throughput required by an upper-layer application, while preserving the fundamental mechanisms of congestion control in TCP. In other words, by *predictable throughput*, we mean the throughput required by an upper-layer application which can be provided at high probability when network congestion level is not so high. In the present paper, we propose the congestion control mechanism of TCP for achieving the required throughput in high probability, regardless of the network congestion level. We modify the degree of increase of the congestion window size of a TCP connection in the congestion avoidance phase, by using the information on the available bandwidth of the network path obtained by Inline Measurement TCP (ImTCP) [2], which has been proposed by our research group. The application examples of the proposed mechanism include TCP-based video/voice delivery services, such as Windows Media Player [3], and Skype [4]. We also show that we can control the sum of the throughput of multiple TCP connections, by extending the mechanism for one TCP connection. This mechanism may be used in the situation in which a stable throughput should be provided for the network traffic between two local area networks interconnected by IP-VPN.

The proposed mechanism is evaluated by simulation experiments using ns-2 [5]. We confirm that the proposed mechanism can achieve a TCP throughput of 10%-20% of the bottleneck link capacity even when the link is highly congested and there is little available bandwidth.

2 Proposed Mechanisms

Figure 1 shows an overview of the proposed mechanism. We assume that an upper-layer application sends bw (packets/sec) and t (sec) to the proposed mechanism, which is located at transport-layer. This means that the application requires average throughput bw at every interval of t sec in

TCP data transmission, and the proposed mechanism tries to achieve this demand. Note that by implementing the proposed mechanism, we also need to modify the socket interface to pass the value of required throughput from the upper-layer application to TCP. Here, bw is the *required throughput* and the time interval is referred to as the *evaluation slot*, as shown in Figure 1. We change the degree of increase of the congestion window size of a TCP connection to achieve a throughput of bw every t sec. Note that in the slow start phase, we use a mechanism that is identical to the original TCP Reno, i.e., the proposed mechanism changes the behavior of TCP only in the congestion avoidance phase. By minimizing the degree of modification of TCP source code, we expect that the original property of the congestion control mechanism can be preserved. We can also reduce the introduction of implementation bugs by basing on the existing TCP source code.

Also, we do not change the degree of decrease of the congestion window size from the original TCP (=0.5) and we describe in 2.2.2 that the proposed mechanism sets the degree of increase of the congestion window size to 1 when the network is not congested. Thus, the proposed mechanism can perform fairly with the original TCP connection when the network has sufficient residual bandwidth.

Since the proposed mechanism changes its behavior in units of the Round Trip Time (RTT) of the connection, we introduce a variable e as $t = e \cdot rtt$, where rtt is the RTT value of the TCP connection.

In what follows, we first introduce the calculation method of target throughput in each evaluation slot in Subsection 2.1 and propose an algorithm to achieve the required throughput in Subsection 2.2.

2.1 Calculating target throughput

We split an evaluation slot into multiple sub-slots, called *control slots*, to control the TCP behavior in a finer-grained time period. The length of the control slot is s (RTT), where s is $2 \leq s \leq e$. We set the throughput value we intend to achieve in a control slot, which is referred to as the *target throughput* of the control slot. We change the target throughput in every control slot and regulate the packet transmission speed in order to achieve the target throughput. The final goal is to make the average throughput in the evaluation slot larger than or equal to bw, the required throughput.

We use the smoothed RTT (sRTT) value of the TCP connection to determine the lengths of the evaluation slot and the control slot. That is, we set the length of the i-th control slot to $s \cdot srtt_i$, where $srtt_i$ is the sRTT value at the beginning of the i-th control slot. At the end of each control slot, we calculate the achieved throughput of the TCP connection by dividing the number of successfully transmitted packets in the control slot by the length of the control slot. We then set the target throughput of the i-th control slot, g_i (packets/sec), as follows:

$$\begin{cases} g_i = bw + (g_{i-1} - tput_{i-1}) \\ g_0 = bw \end{cases}$$

where $tput_i$ (packets/sec) is the average throughput of the i-th control slot. This equation means that the target throughput of the i-th control slot is determined according to the difference between the target throughput and the achieved throughput in the $(i-1)$-th control slot.

2.2 Achieving the target throughput by changing the congestion window size

Although it may seem that one simple method by which to achieve the target throughput by TCP would be to fix the congestion window size to the product of the target throughput and RTT and to keep the window size even when packet losses occur in the network, such a straightforward method would introduce several problems in the network congestion that could not be resolved. In addition, such a method would result in severe unfairness with respect to co-existing connections using the original TCP Reno. Therefore, in the proposed mechanism, the degree of modification of the TCP congestion control mechanism is minimal in order to maintain the original properties of TCP. This means that the degree of the congestion window size is increased only in the congestion avoidance phase of a TCP connection. This does not modify the TCP behavior in the slow start phase or when a TCP connection experiences packet loss(es).

In the proposed mechanism, the sender TCP updates its congestion window size $cwnd$ in the congestion avoidance phase according to the following equation when it receives an ACK packet from the receiver TCP:

$$cwnd \leftarrow cwnd + \frac{k}{cwnd} \tag{1}$$

where k is the control parameter. From the above equation, we expect that the congestion window size increases by k packets in every RTT. The main function of the proposed mechanism is to regulate k dynamically and adaptively, whereas the original TCP Reno uses a fixed value of $k = 1$. In the rest of this subsection, we explain how to change k according to the network condition and the current throughput of the TCP connection.

2.2.1 Increasing the degree of the congestion window size

Here, we derive k_j^{bw}, which is an ideal value for the degree of increase of the congestion window size when the j-th ACK packet is received from the beginning of the i-th control slot, so that the TCP connection achieves g_i of the average throughput. For achieving the average throughput g_i in the i-th control slot, we need to transmit $(g_i \cdot srtt_i \cdot s)$ packets in $(s \cdot srtt_i)$ sec. However, since it takes one RTT to receive the ACK packet corresponding to the transmitted packet, and since it takes at least one RTT to detect packet loss and retransmit the lost packet, we intend to transmit $(g_i \cdot s \cdot srtt_i)$ packets in $((s-2) \cdot srtt_i)$ sec.

We assume that the sender TCP receives the j-th ACK packet at the n_j-th RTT from the beginning of the control slot, and the congestion window size at that time is $cwnd_{n_j}$. Since the congestion window size increases by k packets every RTT, we can calculate p_{snd}, the number of packets that would be transmitted if we use k_j^{bw} for k in Equation (1) in the rest of the control slot, the length of which is $(s - 2 - n_j) \cdot srtt_i$ sec:

$$p_{snd} = (s - n_j - 1)cwnd_{n_j} + \frac{k_j^{bw}}{2}(s - n_j - 1)(s - n_j)$$

On the other hand, p_{need}, i.e., the number of packets that should be transmitted in order to obtain g_i, is calculated as follows:

$$p_{need} = g_i \cdot srtt_i \cdot s - a_j$$

where a_j is the number of transmitted packets from the beginning of the control slot to when j-th ACK packet is received. Then, we can calculate k_j^{bw} by solving the equation $p_{snd} = p_{need}$ for k_j^{bw};

$$k_j^{bw} = \frac{2\{g_i \cdot srtt_i \cdot s - a_j - (s - n_j - 1)cwnd_{n_j}\}}{(s - n_j - 1)(s - n_j)} \quad (2)$$

In the proposed mechanism, we use the above equation to update k for Equation (1) when the sender TCP receives a new ACK packet.

2.2.2 Limitation of k based on the available bandwidth

By using Equation (2) for determining k, the degree of increase of the congestion window size becomes too large when the current throughput of a TCP connection is far below the target throughput. Values of k that are too large would cause bursty packet losses in the network, resulting in a performance degradation due to retransmission timeouts. On the other hand, when the network has sufficient residual bandwidth, the degree of increase of the congestion window size in Equation (2) becomes smaller than 1. This results in a lower throughput increase than TCP Reno. Therefore, we limit the maximum and minimum values for k, which are denoted by k_{max} and k_{min}, respectively. We simply set $k_{min} = 1$ to preserve the basic characteristics of TCP Reno. On the other hand, we should set k_{max} such that bursty packet losses are not invoked, whereas the target throughput should be obtained. Thus, we decide k_{max} according to the following considerations.

First, when the proposed mechanism has obtained the target throughput in all of the control slots in the present evaluation slot, we determine that the available bandwidth of the network path would be sufficient to obtain the target throughput of the next control slot. Therefore, we calculate k_{max} so as to avoid packet losses by using the information of the available bandwidth of the network path. Here, the information about the available bandwidth of the network path is estimated by ImTCP [2], which is the mechanism of inline network measurement. ImTCP measures the available bandwidth of the network path between sender and receiver hosts. In TCP data transfer, the sender host transfers a data packet and the receiver host replies to the data packet with an ACK packet. ImTCP measures the available bandwidth using this mechanism, that is, ImTCP adjusts the sending interval of data packets according to the measurement algorithm and then calculates the available bandwidth by observing the change of ACK arrival intervals. Because ImTCP estimates the available bandwidth of the network path from data and ACK packets transmitted by an active TCP connection in an inline fashion, ImTCP does not inject extra traffic into the network. ImTCP is described in detail in [2].

Next, when the proposed mechanism has not obtained the target throughput in the previous control slot, the proposed mechanism will not obtain the target throughput in the following control slots. We then set k_{max} so as to obtain a larger throughput than the available bandwidth of the network path. This means that the proposed mechanism would steal bandwidth from competing flows in the network in order to achieve the required bandwidth by the upper-layer application.

In summary, the proposed mechanism updates k_{max} by using the following equation when the sender TCP receives a new ACK packet:

$$k_{max} = \begin{cases} A \cdot srtt_i - cwnd \\ \qquad (\forall x\{(1 \le x < i) \vee (tput_x < g_x)\}) \quad (3) \\ min(A + (g_i - avgA_{i-1}), P) \cdot srtt_i - cwnd \\ \qquad (\exists x\{(1 \le x < i) \wedge (tput_x < g_x)\}) \quad (4) \end{cases}$$

where A and P (packets/sec) are the current values for the available bandwidth and physical capacity as measured by ImTCP, $avgA_i$ is the average available bandwidth in the i-th control slot.

2.3 Length of the control slot

In general, the length of the control slot (s) controls the trade-off relationship between the granularity of the throughput control and the influence on the competing traffic. For example, if we use a small value for s, it becomes easier to obtain the required throughput because we update the target throughput g_i more frequently. On the other hand, the smaller value of s means that the congestion window size is changed so drastically that we achieve the average throughput in smaller control slot, which results in a larger effect on other competing traffic. Therefore, we should set s to be as large as possible, while maintaining the required throughput, adoptively to network condition.

The algorithm is based on the following considerations. First, when the proposed mechanism has not obtained the target throughput although we set k_{max} by using Equation (4), we halve s in order to achieve the target throughput. Second, when the proposed mechanism has achieved the target throughput with k_{max} calculated by Equation (3) and the congestion window size is satisfied with $cwnd \ge bw \cdot srtt_i$, we can expect that the proposed mechanism could achieve the target throughput even when we increase the length of the control slot. Therefore, we double s in the next evaluation slot.

2.4 Maintaining multiple connections

In this subsection, we depict the mechanism that controls the sum of the throughput of multiple TCP connections, by extending the mechanism in Subsections 2.2 and 2.3. In this mechanism, we assume that multiple TCP connections are maintained at transport-layer proxy nodes such as TCP proxy [6], and the throughput is controlled at the proxy nodes (Figure 2). The proposed mechanism is intended to achieve the required throughput, bw, of the sum of multiple TCP connections at every t sec interval, and the multiple TCP connections use the same values for the length of the evaluation and control slots, which are set based on the minimum sRTT measured by the sender-side proxy node. Here, the sender-side proxy node can identify the number of active TCP connections. This assumption is natural when we use explicit proxy mechanisms such as TCP proxy and SOCKS [7].

To determine k in Equation (1) for each connection, we can simply extend Equation (2) to multiple TCP connections as follows:

$$k_j^{bw} = \frac{2\{(g_i \cdot srtt_i \cdot s - a_j^{sum})/N_{\rm pm} - (s - n_j - 1)cwnd_i^{n_j}\}}{(s - n_j - 1)(s - n_j)}$$

where a_j^{sum} is the sum of packets that TCP senders have sent when receiving the j-th ACK, and $N_{\rm pm}$ is the number of the active TCP connections. We use this equation for all

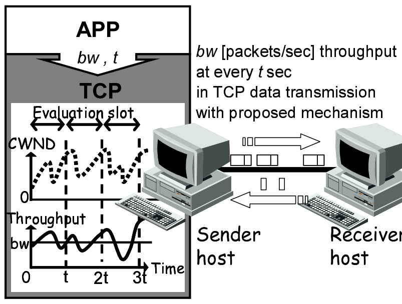
APP
bw , t
TCP
Evaluation slot
CWND
0
Throughput
bw
0 t 2t 3t
Time
bw [packets/sec] throughput at every t sec in TCP data transmission with proposed mechanism
Sender host
Receiver host

Figure 1. Overview of the proposed mechanism

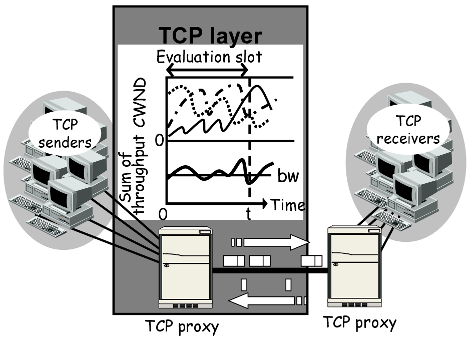

Figure 2. Mechanism for multiple TCP connections

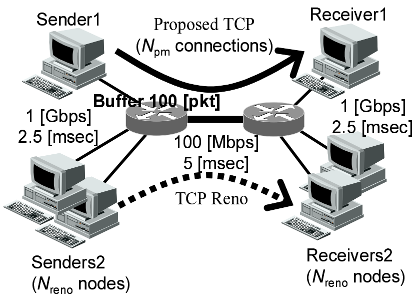

Figure 3. Network model for simulation experiments

TCP connections. This equation means that the degree of the increase of the congestion window size is calculated by distributing the number of packets needed for achieving the target slot to the active TCP connections,

3 Simulation Results and Discussions

In this section, we evaluate the proposed mechanism by simulation experiments using ns-2. Figure 3 shows the network model. This model consists of sender/receiver hosts, two routers, and links between the hosts and router. We set the packet size to 1,000 Bytes. The bandwidth of the bottleneck link is set to 100 Mbps (which corresponds to 12,500 packets/sec), and the propagation delay is 5 msec. A DropTail discipline is deployed at the router buffer and the buffer size is set to 100 packets. The number of TCP connections using the proposed mechanism is N_{pm}, and the number of TCP Reno connections, for creating background traffic, is N_{reno}. The bandwidth of the access links is set to 1 Gbps, and the propagation delay is 2.5 msec. For the proposed mechanism, we set $t = 32\cdot$ RTT (e = 32) for the length of evaluation slot. Here, 32 RTT corresponds to approximately 1 sec in this network model. In addition, s, the length of control slot, is initialized to 16.

3.1 Case of one connection

We first evaluate the performance of the proposed mechanism for one TCP connection. In this simulation, we set $N_{\mathrm{pm}} = 1$, and bw is 2,500 (packets/sec), which is equal to 20% of the bottleneck link capacity. To change the congestion level of the network, we change N_{reno} to 1, 10, 40 every 5 seconds. Figure 4 shows the changes in the congestion window size, the average throughput, and the length of the control slot of the TCP connection with the proposed mechanism. In this figure, the vertical grid represents the boundaries of the evaluation slots.

The results for 0-5 seconds shown in Figure 4(a) show that when one TCP Reno connection co-exists with a TCP connection of the proposed mechanism, the proposed mechanism can obtain the required throughput while performing almost equivalently to TCP Reno. In this period, the available bandwidth is sufficiently large to obtain the required throughput, because there are only two connections in the network that have capacities of 100 Mbps. Thus, the proposed mechanism sets $k = k_{min}$ (=1), resulting in the fairness with the TCP Reno connection being maintained.

The results for 5-10 seconds, in which case there are 10 TCP Reno connections, we observe that the proposed mechanism has a faster increase in the congestion window size compared to that of TCP Reno connections. This is because, in this case, it is impossible to obtain the required throughput with the identical behavior to TCP Reno, due to the increase in the amount of competing traffic. Consequently, the proposed mechanism changes the degree of increase of the congestion window size (k) in order to achieve the required throughput.

Furthermore, the results after 10 seconds with 40 TCP Reno connections show that the congestion window size of the proposed mechanism increases faster than that of previous cases, and that the length of control slot, s, is changed to a smaller value. This result indicates that the proposed mechanism controls its congestion window size with a smaller length of the control slot to obtain required throughput because sufficient throughput cannot be achieved by merely changing the degree of increase of the congestion window size. As a result, the proposed mechanism can obtain the required throughput even when there are 40 competing TCP Reno connections. Thus, we have confirmed that the proposed mechanism can effectively obtain the required throughput by changing the degree of increase of the congestion window size and the length of the control slot according to the network congestion level.

We next show the relationship between the performance of the proposed mechanism and the number of co-existing TCP Reno connections in greater detail. We set $N_{\mathrm{pm}} = 1$, and bw is 10% (1,250 packets/sec) and 20% (2,500 packets/sec) of the bottleneck link capacity. Figure 5 shows the ratio of the number of evaluation slots in which the proposed mechanism obtains the required throughput to the total number of evaluation slots in the simulation time. In this simulation experiment, the simulation time is 60 seconds. For the sake of comparison with the proposed mechanism, we also show the simulation results obtained using TCP Reno (labeled as "Reno") and modified TCP (labeled as "constant"), which uses a constant congestion window size of $bw \cdot srtt_{min}$ (packets) even when packet drops occur. Here, $srtt_{min}$ is the minimum sRTT value for the TCP connection.

Figure 5 indicates that the original TCP Reno can obtain the required throughput for 100% of evaluation slots when a few background connections co-exist, because the original TCP Reno fairly shares the bottleneck link bandwidth with all of the connections. However, when the number of co-existing connections (N_{reno}) increases, TCP Reno cannot obtain the required throughput because it shares the bandwidth with numerous connections. We can also observe that the TCP with constant window size cannot achieve the required throughput when N_{reno} is larger than 10. In this situation, the network congestion cannot be resolved because the congestion window size is not decreased even when

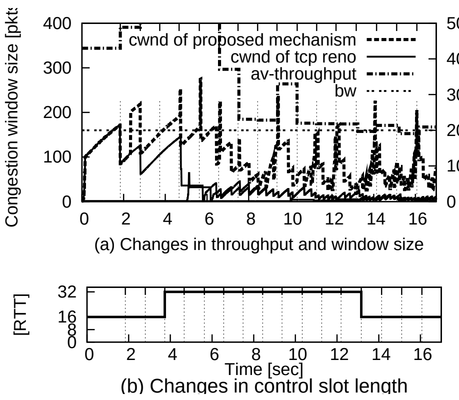

Figure 4. Changes in congestion window size, average throughput and length of control slot

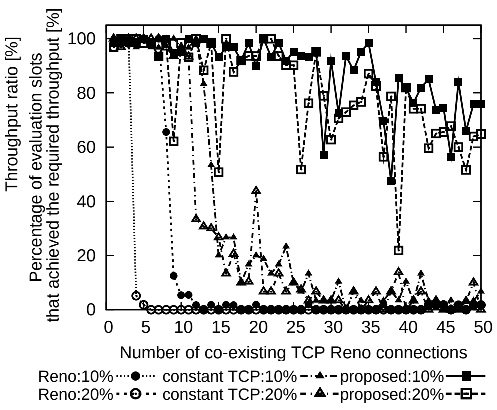

Figure 5. Percentage of evaluation slot required throughput achieved

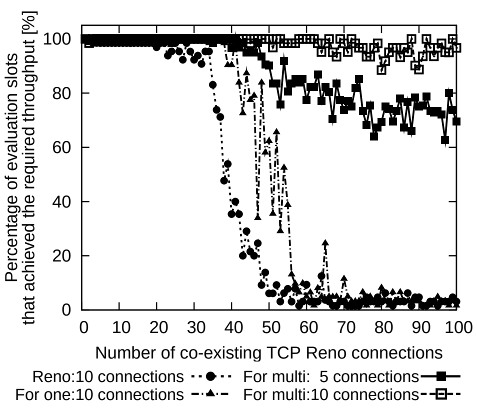

Figure 6. Performance comparison for multiple connections

packet losses occur in the network. Compared to the above mechanisms, the proposed mechanism can obtain the required throughput with high probability even when several connections co-exist in the network. This means that the proposed mechanism can control the trade-off relationship between the aggressiveness of the proposed mechanism and the degree of influences on competing traffic.

3.2 Case of multiple connections

Next, we demonstrate the performance of the proposed mechanism for multiple TCP connections described in Subsection 2.4. In the simulation, we establish multiple TCP connections between Sender 1 and Receiver 1 in Figure 3, and the proposed mechanism at Sender 1 controls the throughput of the connections. We set $bw = 2,500$ (packets/sec), $N_{\rm pm}$= 5 and 10. This setting means that a total throughput of 2,500 (packets/sec) would be achieved for the 5 or 10 TCP connections. The maximum value of the congestion window size of co-existing TCP Reno connections is 100 packets. Here, we assume that the TCP sender host knows the current information on the available bandwidth and physical capacity of the network path. This assumption is necessary in order to focus on evaluating the algorithm described in Subsection 2.4.

Figure 6 shows the percentage of the number of evaluation slots in which the proposed mechanism can obtain the required throughput to the total number of evaluation slots in the simulation time. This figure shows the results when we use 10 connections without the proposed mechanism (labeled as "Reno"), those when we use the proposed mechanism for each of 10 connections, where bw=250 (packets/sec) (labeled as "For one"), and those when the proposed mechanism for multiple connections is used (labeled as "For multi"). This figure shows that the original TCP Reno without the proposed mechanism cannot obtain the required throughput when the number of the co-existing connections becomes larger than 30. When we use the proposed mechanism for each of 10 connections, the performance is not so good when the number of competing connections exceeds 40. This is because bursty packet losses occur in this case because the multiple connections simultaneously inject several packets into the network based on the available bandwidth information estimated by each connection. On the other hand, the proposed mechanism for multiple connections can obtain the required throughput with high probability even when the number of the co-existing TCP Reno connections increases. This is because the problem of the proposed mechanism for one connection is solved by sharing k_{max} with the multiple connections, as described in Subsection 2.4. In addition, the performance for $N_{\rm pm} = 10$ is better than that for $N_{\rm pm} = 5$ to achieve the required throughput. This is because the effect of sharing k_{max} becomes larger when a larger number of connections is accommodated.

4 Conclusion

In the present paper, we focused on upper-layer applications requiring constant throughput, and proposed the TCP congestion control mechanism for achieving the required throughput with a high probability. Through simulation evaluations, we demonstrated that the proposed mechanism for one connection can achieve the required throughput with a high probability, even when there is almost no residual bandwidth of the network path, and that the extended mechanism performs effectively to provide the required throughput for multiple TCP connections.

In future studies, the proposed mechanism will be implemented on actual systems and its performance will be evaluated in an actual network environment which is more complicated than a simulation environment.

References

[1] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP throughput: a simple model and its empirical validation," in *Proceedings of ACM SIGCOMM'98,* Sept. 1998.

[2] L. T. M. Cao, G. Hasegawa, and M. Murata, "An Inline measurement method for capacity of end-to-end network path," in *Proceedings of IM'2005 E2EMON Workshop 2005,* May 2005.

[3] Microsoft Corporation, "Microsoft Windows Media - Your Digital Entertainment Resource," available from `http://www.microsoft.com/windows/windowsmedia/default.mspx`.

[4] Skype Technologies Corporation, "Skype -The whole world can talk for free.," available from `http://www.skype.com/`.

[5] T. V. Project, "UCB/LBNL/VINT network simulator - ns (version 2)," available from `http://www.isi.edu/nsnam/ns/`.

[6] I. Maki, G. Hasegawa, M. Murata, and T. Murase, "Throughput analysis of TCP proxy mechanism," in *Proceedins of ATNAC 2004,* Dec. 2004.

[7] M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, and L. Jones, "SOCKS Protocol Version 5," *RFC 1928,* Apr. 1996.