A transport-layer approach for achieving predictable throughput for Internet applications

Go Hasegawa, Kana Yamanegi and Masayuki Murata
Graduate School of Information Science and Technology, Osaka University
1-3, Yamadaoka, Suita, Osaka 560-0871, JAPAN
Email: hasegawa@cmc.osaka-u.ac.jp

***Abstract*—We propose a transport-layer approach to provide predictable throughput for applications requiring stable bandwidth resource, such as video streaming and VoIP. The proposed mechanism deploys an end-to-end approach: it dynamically sets the increase degree of the congestion window size of a TCP connection according to the measurement results of the available bandwidth of the network path, which is obtained by inline network measurement technique developed in our research group. In the present paper, we briefly introduce the proposed mechanism and show the performance evaluation results in the experimental network and in the commercial Internet environment. We exhibit that the proposed mechanism can achieve the required throughput with high probability without any support from intermediate network nodes.**

***Index Terms*—TCP, predictable throughput, end-to-end, bandwidth measurement**

I. Introduction

The Internet users' demands for network quality has increased due to services becoming progressively diversified and sophisticated because of the remarkable degree to which the Internet has grown, which is due in part to access and backbone network technologies. Applications involving real-time media delivery services, such as VoIP, video streaming and TV meeting systems, all of which have experienced a dramatic level of development, require large and stable amounts of network resources in order to maintain the Quality of Service (QoS). 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 to smoothly provide rich contents, including voice and video.

Most of such video streaming applications use User Datagram Protocol (UDP) as a transport-layer protocol, and the upper-layer application controls the data transmission rate according to the network condition [1]. However, these mechanisms have a large cost when modifying the application program for achieving 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 cannot 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.

Against the above problem, we propose a transport-layer approach for achieving predictable throughput. By predictable throughput, we mean the throughput required by an upper-layer application, which can be provided with high probability when the network congestion level is not extremely high. We modify the degree of the 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 previously proposed by our research group. The application examples of the proposed mechanism include TCP-based video or voice delivery services, such as Windows Media Player [3], RealOne Player [4], and Skype [5]. In [6], we have investigated the fundamental characteristics of the proposed mechanism through simulation experiments, and confirmed 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 residual bandwidth for the TCP connection.

In this paper, we briefly summarize the proposed mechanism and discuss implementation issues of the proposed mechanism. Furthermore, we show the performance evaluation results through the implementation experiments on the experimental networks, and on the commercial Internet environment between Tokyo and Osaka in Japan. From these results, we confirm that the proposed mechanism can achieve the required throughput with high probability in an actual network, as in the simulation results. We also evaluate the importance of bandwidth measurement for the proposed mechanism to confirm the effectiveness of the network resource monitoring for providing higher-level QoS.

II. 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.

Since the proposed mechanism changes its behavior in units of the Round Trip Time (RTT) of the connection, we introduce

978-0-7695-3106-9/08 $25.00 © 2008 IEEE
DOI 10.1109/ICN.2008.30

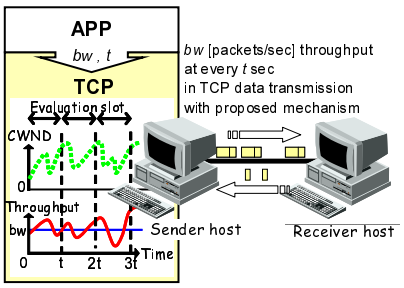


Fig. 1. Overview of the proposed mechanism

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 II-A and propose an algorithm to achieve the required throughput in Subsection II-B.

A. 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.

B. 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.

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};

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

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) Limitation of the increasing degree 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. However, some applications, including transferring sensing observation data and monitoring log on a system, generate data at a constant rate and does not require higher throughput than the designated one even when the network has enough bandwidth. For such applications we do not set k_{min}, meaning that the proposed mechanism does not achieve more than the required throughput even if the residual bandwidth is more than the required throughput.

k_{max}, on the other hand, should be set 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 [7], 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 [7].

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:

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

where A and P (packets/sec) are the current values for the available bandwidth and physical capacity as measured by ImTCP. In Equation (3), $A \cdot srtt_i$ indicates the maximum number of packets that the proposed mechanism can occupy within the network capacity without occurring packet losses. In Equation (4), $(g_i - tput_{i-1}) \cdot srtt_i$ indicates the number of packets required in order to obtain the target throughput when the network has insufficient available bandwidth.

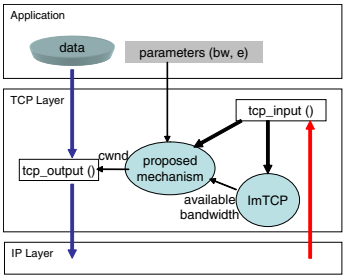


Fig. 2. Outline of implementation architecture

III. Implementation

Figure 2 shows the architecture of the proposed mechanism implemented in Linux 2.6.16.21 kernel system. When new data is generated at the application, the data is passed to the TCP layer through the socket interface [8]. The data is passed to the IP layer after TCP protocol processing by the `tcp_output()` function and the resulting IP packets are injected into the network. On the other hand, an ACK packet that arrives at the IP layer of the sender host is passed to the `tcp_input()` function for TCP protocol processing. The congestion window size of a TCP connection is updated when an ACK packet is passed to the `tcp_input()` function. Therefore, the control program for congestion window size for the proposed mechanism should be implemented in the `tcp_input()` function. Linux 2.6.16 kernel system unifies the interfaces for congestion control mechanisms, and enables to implement congestion control algorithm as a module. In this paper, we implement the proposed mechanism as a module in Linux 2.6.16.21 kernel system.

The `tcp_input()` function calls the `cong_avoid()` function and updates the congestion window size when an ACK packet arrives. The module for the proposed mechanism determines the congestion window size according to its algorithm described in Section 2, and splits time into evaluation/control slot in the `cong_avoid()` function. On the other hand, ImTCP, which we utilize to obtain the available bandwidth of the network path, calculates the available bandwidth in the `tcp_input()` function [9]. The proposed mechanism inquires ImTCP for the available bandwidth in the `cong_avoid()` function, and changes the degree of increase of the congestion window size based on the value.

Figure 3 shows the flow chart of the `cong_avoid()` function of the proposed mechanism. First, the `cong_avoid()` function compares the congestion window size ($cwnd$) and the slow start threshold ($ssthresh$). When $cwnd$ is smaller than $ssthresh$, the congestion window size is updated by the slow start algorithm as TCP Reno. On the other hand, when $cwnd$ is larger than $ssthresh$, the congestion window size is determined based on the algorithm of the proposed mechanism. In the congestion avoidance phase, the proposed mechanism checks the passed time from the beginning of the present evaluation/control slots and judges the end of the slots. When the passed time is longer than the length of the evaluation/control slots, the proposed mechanism calculates the average throughput in the slot and initializes the variables for the next slots. Next, the increase degree of the congestion window size is the determined on consideration of k_{max},

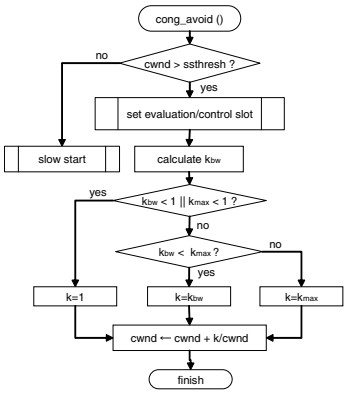


Fig. 3. Flow chart of the `cong_avoid()` function

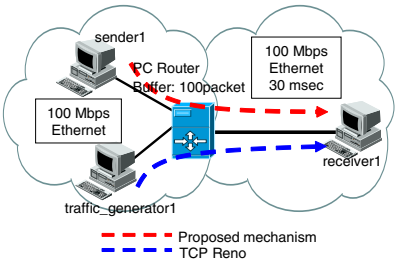
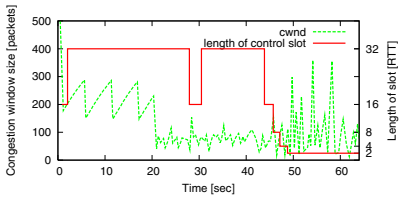


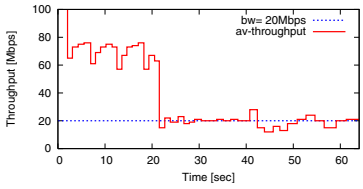
Fig. 4. Experimental network environment

TABLE I
PC SPECIFICATIONS OF THE EXPERIMENTAL NETWORK ENVIRONMENT

	sender1	traffic_generator1	receiver1
CPU	Pentium 4 1.90GHz	Pentium 4 1.7GHz	Xeon 2.80GHz
Memory	1024 MB	2048 MB	2048 MB
Kernel	Linux 2.6.16.21	Linux 2.6.16.21	Linux 2.6.16.21



(a) Changes in window size and control slot length



(b) Changes in throughput

Fig. 5. Changes in cwnd, control slot length and throughput in the experimental network

k_{min} and k_j^{bw}, which is calculated according to Equation (2). Finally, $cwnd$ is updated by Equation (1).

IV. Experimental Results

A. Performance evaluations using an experimental network

We first evaluate the proposed mechanism in an experimental network. Figure 4 shows the experimental network environment. This network environment consists of a PC router in which Dummynet [10] is installed, an endhost that generates cross traffic (traffic_generator1), an endhost that uses the proposed mechanism (sender1), and an endhost that receives packets from each endhost (receiver1). All endhosts and the PC router are connected by a 100-Mbps Ethernet networks. We configured the Dummynet setting so that the minimum RTT between the sender1 and the receiver1 becomes 30 msec. Table I shows the specifications of the endhosts of the experimental network environment.

In this experiment, we set $e = 32$ for the length of evaluation slot, and s, the length of control slot, is initialized to 16. The cross traffic is generated by traffic_generator1, and sends the packets to receiver1. We set the size of TCP socket buffer on traffic_generator1 to limit the maximum throughput of each TCP Reno connection to approximately 4 Mbps, and the amount of the cross traffic is changed by the number of the TCP Reno connections. The value of HZ at the sender host (sender) is set to 20,000.

1) Changes in congestion window size and throughput: In this section, we evaluate the behavior of the proposed mechanism against the changes in the amount of cross traffic. In this experiment, we use one connection using the proposed mechanism, and set bw to 20 (Mbps), which is equal to 20% of the bottleneck link capacity. To change the congestion level of the network, we change the number of TCP Reno connections between traffic_generator1 and receiver1 to 5, 25 and 40 in every 20 seconds. Figure 5(a) shows the changes in the congestion window size and the length of control slot, and Figure 5(b) shows the changes in the average throughput in each evaluation slot.

The results for 0-20 seconds shown in Figures 5(a) and 5(b) show that when five TCP Reno connections co-exist with a TCP connection of the proposed mechanism, the proposed mechanism can obtain the required throughput by keeping the same behavior as the normal TCP connection ($k = 1$ in Equation (1)). In this period, the available bandwidth is sufficiently large to obtain the required throughput, because there are only five connections, each of whose maximum throughput is limited to 4 Mbps, in the network that have 100 Mbps capacity. Thus, the proposed mechanism sets $k = k_{min}$ (=1).

The results for 20-40 seconds, in which case there are 25 TCP Reno connections, we observe that the proposed mechanism has a faster increase in the congestion window size than that in 0-20 seconds. This is because it is impossible to obtain the required throughput with the identical behavior to TCP Reno, due to the increase of the amount of competing traffic. Consequently, the proposed mechanism changes the degree of increase of the congestion window size in order to achieve the required throughput.

Furthermore, the results after 40 seconds with competing 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 pro-

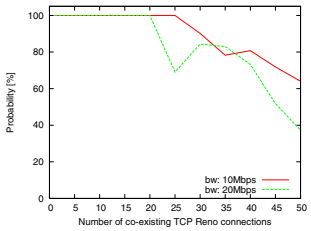


Fig. 6. Ratio of evaluation slot required throughput achieved

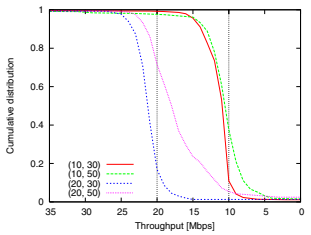


Fig. 7. Cumulative distribution function of throughput

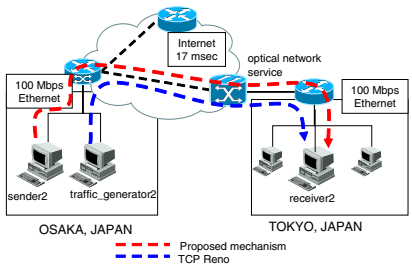


Fig. 8. Experimental system with the Internet environment

TABLE II
PC SPECIFICATIONS OF THE INTERNET ENVIRONMENT

	sender2	traffic_generator2	receiver2
CPU	Pentium 4 3.40GHz	Xeon 3.60GHz	Xeon 2.66GHz
Memory	1024 MB	2048 MB	1024 MB
Kernel	Linux 2.6.16.21	Linux 2.6.17	Linux 2.4.21

posed 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 in the experimental network environment.

2) Probabilities of obtaining required throughput: We next show the relationship between the performance of the proposed mechanism and the number of co-existing TCP Reno connections in more detail. We set $bw = 10, 20$ (Mbps), and use one connection for the proposed mechanism in the network. Figure 6 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.

Figure 6 indicates that the proposed mechanism can obtain the required throughput with high probability even when the connections co-exist in the network. This result is quite similar to the simulation result described in Subsection 3.1. However, the ratio in which the proposed mechanism can achieve the required throughput decreases when bw is 20 Mbps and the number of TCP Reno connection is 25. In this case, the proposed mechanism cannot set the length of control slot to its appropriate value according to the congestion level, because the length of control slot drastically changes by doubled or halved, as described in Subsection 2.2.3. Thus, the ratio in which the proposed mechanism can achieve the required throughput decreases due to the frequent changes in the length of control slot. Against this problem, we plan to improve the control algorithm of the length of control slot as a future work.

We next show the cumulative distribution function of the average throughput in each evaluation slot (Figure 7), when we set bw to 10 and 20 Mbps and the number of the TCP Reno connections to 30 and 50. We observe from Figure 7 that when the number of co-existing TCP Reno connections is 30, approximately 80 % of the evaluation slots can achieve more than the required throughput, and the rest of them can achieve the throughput close to the required throughput. On the other hand, the ratio becomes degraded when there are 50 competing TCP Reno connections. However, most of the evaluation slots which cannot achieve the required throughput can achieve the throughput close to the required throughput.

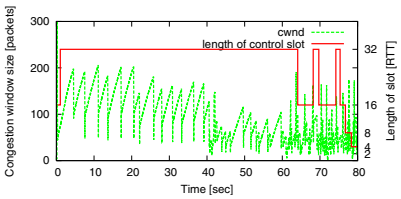
B. Experiments in an actual Internet environment

We finally confirm the performance of the proposed mechanism in the commercial Internet environment. Figure 8 shows the network environment, which consists of two local area networks in Osaka, Japan and Tokyo, Japan, which are connected to the Internet. The network environment consists of an endhost that generates cross traffic (traffic_generator2), an endhost that uses the proposed mechanism (sender2), and an endhost that receives packets from both endhosts (receiver2) across the Internet. The path of the commercial Internet network between Osaka and Tokyo passes 100-Mbps optical fiber services, and the local area networks in Osaka and Tokyo is constructed by a 100-Mbps Ethernet networks. Table II shows the specifications of the endhosts of the experimental system. Through preliminary investigations, we confirmed the following characteristics regarding the network between Osaka and Tokyo:

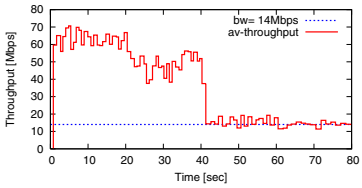
- Seventeen hops exist in the network path from Osaka to Tokyo.
- The minimum value of RTTs is 17 msec.
- The upper limit of the bandwidth between Osaka and Tokyo is 70 Mbps.

In this experiment, we set $e = 32$ for the length of evaluation slot, and s, the length of control slot, is initialized to 16. The cross traffic is generated by traffic_generator2, and sends the packets to receiver2. We set the size of TCP socket buffer on traffic_generator2 to limit the maximum throughput of each TCP Reno connection to approximately 3 Mbps, and the amount of the cross traffic is changed by the number of the TCP Reno connections. On the other hand, the size of TCP socket buffer on sender2 and receiver2 is enough large.

We first evaluate the behavior of the proposed mechanism against the change in the amount of cross traffic. In this experiment, we use one connection for the proposed mechanism, and set bw to 14 (Mbps), which is equal to 20% of the bottleneck



(a) Changes in window size and control slot length



(b) Changes in throughput

Fig. 9. Changes in cwnd, control slot length and throughput in the Internet experiment

link capacity. To change the congestion level of the network, we change the number of TCP Reno connections between traffic_generator2 and receiver2 to 0, 5, 25, and 40 in every 20 seconds. Figure 9(a) shows the changes in the congestion window size and the length of control slot, and Figure 9(b) shows the changes in the average throughput in each evaluation slot.

The results for 0-20 seconds in Figure 9, when there is only one connection for the proposed mechanism, we can find that the upper limit of the throughput between Osaka and Tokyo is 70 Mbps because the proposed mechanism can obtain at most approximately 70 Mbps. In addition, the results after 20 seconds in Figure 9 is almost equivalent to the results on the experimental network shown in Subsection 4.2.1. That is, the results for 20-40 seconds show that the proposed mechanism can obtain more than the required throughput by keeping the same behavior as the normal TCP connection, and the results for 40-60 seconds show that it can achieve the required throughput by having a faster increase in the congestion window size, The results after 60 seconds, we observe that the length of control slot is changed to a smaller value, and the proposed mechanism can achieve the required throughput. 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 in the commercial Internet environment.

We next evaluate the average throughput in each evaluation slot, when we set bw to 7 and 14 Mbps and the number of the TCP Reno connections to 30 and 50. Figure 10 shows the cumulative distribution function of the average throughput in each evaluation slot. From Figure 10, we can observe that the ratio of achieving the required throughput is slightly smaller than that in the experimental network in Figure 7. One of the possible reason is that there are short-lived connections, including web-traffic, in the Internet environment, those traffic has highly bursty nature. Since the proposed mechanism is based on TCP, it can not adapt to shorter-term changes of the network condition than its RTT. Another reason is that

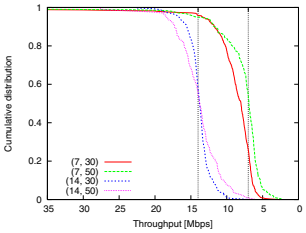


Fig. 10. CDF of throughput in the Internet experiment

the measurement accuracy of the available bandwidth of the network path operated by ImTCP becomes slightly degraded in the actual Internet environment. However, most of the evaluation slots which cannot achieve the required throughput can achieve the throughput close to the required throughput. Thus, we conclude that the proposed mechanism works well even in the actual Internet environment.

V. Conclusion

In this 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. The proposed mechanism is modified 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. We implemented the proposed mechanism on Linux 2.6.16.21 kernel system, and confirmed from implementation evaluation that the proposed mechanism works well in the actual networks.

In future studies, we will evaluate the performance of the proposed mechanism in other actual network environments. In addition, we would like to confirm the applicability of the proposed mechanism to actual upper-layer applications, such as a real-time video streaming.

References

[1] M. Handley, S. Floyd, J. Padhye, and J. Widmer, "TCP friendly rate control (TFRC): Protocol specification," *RFC 3448*, Jan. 2003.

[2] C. L. T. Man, G. Hasegawa, and M. Murata, "ImTCP: Tcp with an inline measurement mechanism for available bandwidth," *Computer Communications Journal special issue of Monitoring and Measurements of IP Networks*, vol. 29, pp. 1614–1626, June 2006.

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

[4] RealNetworks Corporation, "Rhapsody & RealPlayer," available from http://www.real.com/.

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

[6] K. Yamanegi, G. Hasegawa, and M. Murata, "Congestion control mechanism of TCP for achieving predicatable throughput," in *Proceedings of ATNAC 2006*, Dec. 2006, pp. 117–121.

[7] C. L. T. Man, G. Hasegawa, and M. Murata, "A simultaneous inline measurement mechanism for capacity and available bandwidth of end-to-end network path," *IEICE Transactions on Communications*, vol. E89-B, pp. 2469–2479, Sept. 2006.

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

[9] T. Tsugawa, G. Hasegawa, and M. Murata, "Implementation and evaluation of an inline network measurement algorithm and its application to TCP-based service," in *Proceedings of NOMS 2006 E2EMON Workshop 2006*, Apr. 2006.

[10] L. Rizzo, "IP_DUMMYNET," available from http://info.iet.unipi.it/~luigi/ip_dummynet/.