Receiver–based Management Scheme of Access Link Resources for QoS–Controllable TCP Connections

Kazuhiro Azuma, Go Hasegawa and Masayuki Murata
Graduate School of Information Science and Technology, Osaka University
1-3 Machikaneyama, Toyonaka, Osaka 560-8531, Japan
E-mail: {k-azuma, hasegawa, murata}@ist.osaka-u.ac.jp

***Abstract*— Although the bandwidth of access networks is rapidly increasing with the latest techniques such as DSL and FTTH, the access link bandwidth remains a bottleneck, especially when users activate multiple network applications simultaneously. Furthermore, since the throughput of a standard TCP connection is dependent on various network parameters, including round–trip time and packet loss ratio, the access link bandwidth is not shared among the network applications according to the user's demands. In this thesis, we present a new management scheme of access link resources for effective utilization of the access link bandwidth and control of the TCP connection's throughput. Our proposed scheme adjusts the total amount of the receive socket buffer assigned to TCP connections to avoid congestion at the access network, and assigns it to each TCP connection according to characteristics in consideration of QoS. The control objectives of our scheme are (1) to protect short-lived TCP connections from the bandwidth occupation by long-lived TCP connections, and (2) to differentiate the throughput of the long-lived TCP connections according to the upper-layer application's demands. One of the obtained results from the simulation experiments is that our proposed scheme can reduce the delay of short-lived document transfer perceived by the receiver host by up to about 90% , while a high utilization of access link bandwidth is maintained.**

I. Introduction

The rapid increase in Internet users has been the impetus for the performance of backbone networks into solving network congestion posed against the context of increasing network traffic. However, little work has been done in the area of improving the performance of Internet servers despite the projected shift in the performance bottleneck from backbone networks to endhosts or access networks. For example, busy Web servers must have many simultaneous HTTP sessions, and server throughput degrades when effective resource management is not considered, even with large network capacity. Furthermore, Web proxy servers [1] must also accommodate a large number of TCP connections, since they are usually prepared by ISPs (Internet Service Providers) for their customers. In our previous work, therefore, we have proposed a TCP connection resource management scheme at endhosts to solve those problems and confirmed its effectiveness through simulation and implementation experiments [2].

On the other hand, the bandwidth of access networks is also rapidly increasing with the latest techniques, such as DSL (Digital Subscriber Line) and FTTH (Fiber to the Home). However, the access link bandwidth remains a performance bottleneck, especially when users activate multiple network applications simultaneously, as shown in Figure 1. In this figure, six TCP connections are established between a user host which becomes a TCP receiver host, and the hosts, A, B, C, D, E and F, which correspond to TCP sender hosts. Each of those connections corresponds to upper–layer applications such as P2P and FTP. For example, when the access link bandwidth is 4 Mbps, which is the typical value on the current Internet in Japan [3], 667 Kbps is assigned to each TCP connection when the access link bandwidth is fairly shared. However, since the throughput of the standard TCP connections is affected by various network parameters, including round–trip time (RTT) and packet loss ratio, the access link bandwidth is not shared equally among the network applications. For the same reason, we cannot expect a differentiated throughput for all TCP connections according to the user's demands and the application characteristics. For example, in Figure 1, we cannot intentionally increase the throughput of the TCP connection for P2P and FTP data transmission, and restrict that for the network update operation which should be done in the background.

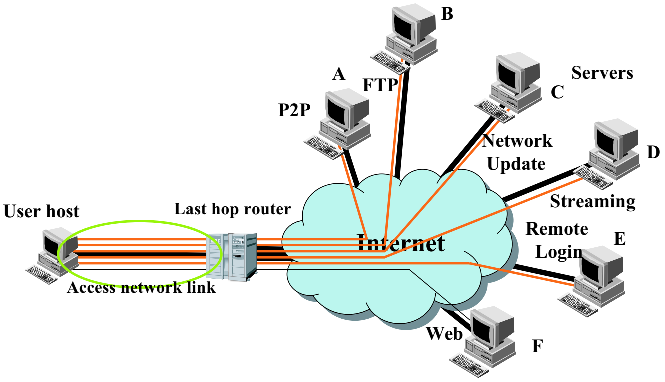


Figure. 1. Bottleneck at Access Network

Another problem we focus on in this paper is the performance unfairness between short–lived and long–lived TCP connections. When the access network link is congested and some incoming packets are discarded, the performance of the short–lived connections degrades seriously, compared with that of the long–lived connections [4], [5]. This problem significantly affects the user's perceived performance such as Web document transfer delay when they activate long–lived network applications simultaneously.

Therefore, in this paper, we present a new access link resources management scheme for the effective utilization of the access link bandwidth and the control of the performance of TCP connections. Our proposed scheme *virtually* adjusts the amount of the receive socket buffer for all TCP connections in order to avoid congestion at the access link [6], [7], and assigns it to each TCP connection according to its characteristics and the user's demands for the applications. All TCP connections at the endhost are categorized into two types, which are short–lived connections and long–lived connections. As mentioned above, since the data transfer time in short–lived connections increases greatly when a packet loss occurs, it is necessary to prioritize the short–lived connections, that is, to try not to discard the short–lived connection's packets at the access link. For long–lived connections, on the other hand, it is important to assign the access link resources according to the applications' characteristics and the user's demands, as mentioned above. Thus, the objective of our proposed scheme is to prioritize short–lived TCP connections and differentiate the throughput of long–lived TCP connections, while keeping the utilization of the access link.

The access link resource management scheme proposed in this paper is implemented in a TCP receiver host, which corresponds to the user host in Figure 1. There are two major reasons for this choice. One is that in the congestion control mechanism of standard TCP [8], [9], a sender host cannot exactly estimate the congestion level of the access link near a receiver host because of the congestion control being performed by the sender host. Another reason is that we cannot control the behavior of TCP sender hosts to differentiate their throughputs because each TCP connection lives independently on the other connections. That is, the best way is for the receiver host to control the utilization of the access network resources. We also note that our proposed scheme does not modify the congestion control mechanism of TCP, and network protocol structures.

The rest of this paper is organized as follows: In Section II, we propose a new access link resource management scheme and confirm its effectiveness by detailing the results we obtained in the simulation experiments in Section III. Finally, we present our concluding remarks in Section IV.

II. Our Approach and Algorithm

Our proposed scheme can be divided into two mechanisms: adjusting the amount of the receive socket buffer for all TCP connections and assigning it to each TCP connection.

A. Adjusting the Amount of Receive Socket Buffer for All TCP Connections

As mentioned in Section I, this mechanism is for controlling the arrival rate of packets at the access link and avoiding congestion there. Since the network congestion level dynamically changes, we adjust the amount of the receive socket buffer for all TCP connections at regular intervals. In detail, our proposed scheme periodically measures the RTTs of all TCP connections at the receiver host, and adjusts the amount of the receive socket buffer for all TCP connections according to the measured results as follows:

- When the RTTs of all TCP connections do not increase, we determine that the access link resources are still sufficient and increase the amount of the receive socket buffer for all TCP connections.
- When the RTTs of all TCP connections increase, we decrease the amount of the receive socket buffer for all TCP connections, since it is likely that the congestion occurs at the access link.
- Otherwise, we do not change the amount of the receive socket buffer for all TCP connections.

It is important that the amount of the receive socket buffer for all TCP connections be limited to the value determined above, even when the system has sufficient memory capacity and larger memory space can be assigned for the receive socket buffer. This is because if the receive socket buffer size for each TCP connection is too large, the packet transmission rate of the connection unnecessarily increases, which causes the congestion of the access link.

We also note that the meaning of *virtually* adjusting the amount of the receive socket buffer for all TCP connections is to adjust the advertised window size, which reports the current available size of the receive socket buffer to the TCP sender [8], instead of increasing/decreasing the actual size of the receive socket buffer.

B. Assigning Receive Socket Buffer to TCP Connections

Before we assign receive a socket buffer to each TCP connection, we categorize all TCP connections into short–lived or long–lived. This is because the objectives of our scheme are to prioritize short–lived TCP connections, to differentiate the throughput of long–lived TCP connections in consideration of the applications' QoS, and to keep the utilization of the access link. However, the TCP receiver cannot know whether a TCP connection is short–lived or long–lived, since the data size transferred by the TCP connection is not informed in advance. Therefore, in our proposed scheme, we use a threshold–based approach. That is, we use a threshold value for the receive socket buffer of each TCP connection and categorize the connection by whether the assigned receive buffer size exceeds the threshold value or not. Since all TCP connections are initially categorized as short–lived in this approach, these states are expressed as "initial state" instead of "short–lived" and as "persistent state" instead of "long–lived" in our scheme. The threshold value is set to the receive socket buffer size, in case we consider all of the TCP connections currently at the receiver host to be in a persistent state.

Then, the receive socket buffer size assigned to each TCP connection is determined as follows. We first assign the receive socket buffer to initial connections preferentially, and then to persistent connections.

(1) For initial TCP connections

We assign the receive socket buffer for initial connections to improve the arrival packet rate from the initial connections at the receiver host. At the same time, our proposed scheme tries not to unnecessarily reduce the throughput of persistent connections when prioritizing initial connections. Therefore, the receive socket buffer size assigned to each initial connection is determined in consideration of the increase algorithm of the congestion window size in TCP's slow start phase.

(1–a) When the amount of the receive socket buffer for all TCP connections is sufficient

In this case, the receive socket buffer required by all initial connections can be assigned. Since an initial connection is likely to be in the slow start phase, we focus on the increase algorithm of the congestion window size in the slow start phase to avoid degrading the throughput of persistent connections. That is, the assigned size to the initial connection i is determined according to the number of RTTs from the beginning of the connection, which is described as t_i. Consequently, the receive socket buffer size required by connection i in this case becomes $2 \cdot 2^{t_i}$ packets.

(1–b) When the amount of the receive socket buffer for all TCP connections is insufficient

In this case, the receive socket buffer required by all initial connections cannot be assigned. Therefore, the receive socket buffer is distributed to all initial connections proportionally to the difference between $2 \cdot 2^{t_i}$ and the threshold value mentioned above. This is originated by the consideration that it is necessary to prioritize TCP connections just after beginning their data transmission, since they are likely to have small window sizes.

Here, we define the amount of the receive socket buffer for all TCP connections as B, the number of initial connections as N_{is}, and the threshold value as $threshold_i$. Then, the receive buffer size assigned to initial connection i, R_i, can be described as the following equations:

$$R_i = \begin{cases} 2 \cdot 2^{t_i} & \text{(1–a)} \\ B \cdot \dfrac{(threshold_i - 2 \cdot 2^{t_i})}{\displaystyle\sum_{j}^{N_{is}} (threshold_j - 2 \cdot 2^{t_j})} & \text{(1–b)} \end{cases}$$

(2) For persistent TCP connections

It is important to consider each network application's characteristic and user's demands for persistent connections to utilize effectively the access link resources. We assume that each persistent TCP connection has a priority value pre–defined according to the user's demands and the application characteristics.

(2–a) When the amount of the receive socket buffer for all TCP connections is sufficient

Since the amount of the receive socket buffer for all TCP connections is larger than that required by all initial connections, the remainder is assigned to the persistent connections according to their priority values and RTTs.

(2–b) When the amount of the receive socket buffer for all TCP connections is insufficient

In this case, we cannot assign enough size of the receive socket buffer for the persistent connections. However, it is necessary to assign at least 1 mss for each connection to avoid the TCP's silly window syndrome [10], [11].

Here, we define the amount of the receive socket buffer for all initial connections as T_{is}, the number of persistent connections as N_{ps}, the RTT of TCP connection i as rtt_i, and the priority value of each TCP connection as p_i. Then, the receive socket buffer size assigned to each persistent connection i, R_i, is described as the following equations:

$$R_i = \begin{cases} (B - T_{is}) \cdot \dfrac{p_i \cdot rtt_i}{\displaystyle\sum_{j}^{N_{ps}} (p_j \cdot rtt_j)} & \text{(2–a)} \\ 1\ mss_i & \text{(2–b)} \end{cases}$$

C. Discussions on Bottleneck Discovery

Our proposed scheme estimates the network congestion level from the changes of the RTTs of all TCP connections at the receiver host. Therefore, if RTTs increase due to a bottleneck link other than the access link, our proposed scheme may fail to estimate the access link congestion. However, when the RTT of most of TCP connections increase, we can consider that these connections are affected by the congestion occurring at the identical link, which corresponds to the access link in this case. Consequently, if the RTTs of all TCP connections increase, our proposed scheme determines the access link bandwidth is a bottleneck. On the other hand, when the RTTs of a few TCP connections increase, we can consider that these connections are affected by the congestion at the link through which only these connections pass. That is to say, this congestion is considered to occur at the link other than the access link. Therefore, if the RTTs of a few TCP connections at the receiver host increase, our proposed scheme determines that the bandwidth of the link other than the access link is a bottleneck.

III. Simulation Experiments

In this section, we compare our proposed scheme with the standard TCP, the scheme proposed in [12] and the scheme proposed in [13] through simulation experiments with ns–2 [14] and evaluate the effectiveness of our proposed scheme. In this section, we denote the scheme proposed in [12] *Spring* and that in [13] *Mehra*.

A. Simulation Setup

Figure 2 shows the simulation model. It consists of four sender hosts A through D and one receiver host. The propagation delays between the sender hosts and the receiver host are: A : 35 msec, B : 45 msec, C : 55 msec and D : 10 msec, respectively. A performance bottleneck in this simulation is the receiver host's access link bandwidth

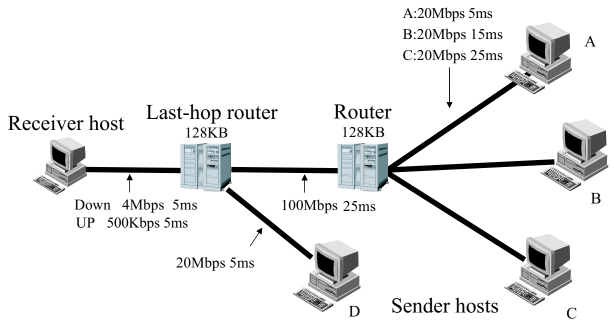


Figure. 2. Simulation Topology

TABLE I
PARAMETERS IN SPRING AND MEHRA ([12], [13])

Spring ([12])		Mehra ([13])	
$Xput_{link}$	4 Mbps	priority	0
$QLength_{Loss}$	64 KBytes	weight	0
$QLength_{Delay}$	5 KBytes	minimal rate	0
Rcv_{short}	2KBytes		
Rcv_{long}	8KBytes		

(4 Mbps). The bandwidths and propagation delays of other links are as described in Figure 2. The router buffer size is set to 128 KBytes, and packet size is 1,500 Bytes. Our proposed scheme (and Spring and Mehra for comparison) is implemented at the receiver host. In this simulation, the bulk data transfers are performed from the sender hosts A through C to the receiver host (long–lived connections) so that the access link bandwidth be fully utilized. At the same time, 100 short–lived connections, each of which transfers 30 KBytes data, start being activated from the sender host D at 100 seconds with random intervals (5 sec average). The interval to check RTTs of TCP connections in our proposed scheme is 5 seconds and we set the parameters in Spring and Mehra as summarized in Table I, except that the parameters of Mehra (priority, minimal rate and weight) is not set. Note that the parameters in Table I are the values recommended in the papers [12], [13] and there is almost no difference in parameter selection caused by the changes in network topology and/or simulation.

B. Simulation Results

Figure 3 shows the CDF (cumulative relative frequency) of data transfer time for short–lived connections, the change of utilization of the access link during 500 seconds simulation time, and the change of the average queue length of the router buffer. In this figure, our proposed scheme is labeled as "proposed," the scheme of [12] as "Spring," the scheme of [13] as "Mehra," and the standard TCP as "traditional." From Figures 3(a), we can observe that the traditional scheme, which has no special control, shows the longest data transfer time for short–lived connections. This is because the traditional scheme cannot exactly estimate the access link resources, and many packet losses occur at the buffer of the last–hop router due to congestion at the access link. Although the traditional scheme shows a high enough utilization of the access link as shown in Figure 3(b), most of the bandwidth of the access link is occupied by the long–lived connections, while that of the short–lived connections is very low.

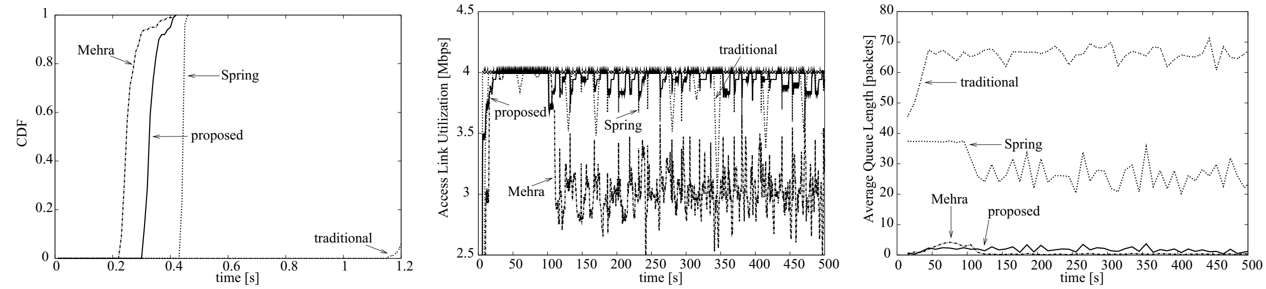
From Figure 3(a), Mehra shows the shortest data transfer time for short–lived connections, but the lowest utilization of the access link from Figure 3(b). Since Mehra tries to assign the same bandwidth for short–lived and long–lived connections, meaning that the access link bandwidth (4 Mbps) is equally shared among three long–lived connections and one short–lived connection in this case. Consequently, the access link bandwidth becomes under–utilized, since the bandwidth assigned to the short–lived connections cannot be fully utilized. The near–zero average queue length of Mehra in Figure 3(c) also confirms the under–utilization of the access link.

From Figure 3(c), Spring shows that the average queue length at the last–hop router is relatively long. This is because Spring assigns a receive socket buffer to each TCP connection so that half of the router buffer is utilized. This results in an increase in the data transfer time for short–lived connections, as shown in Figure 3(a). Note that since the access link bandwidth is not so large, the queuing delay at the last–hop router cannot be ignored. On the other hand, our proposed scheme shows that the average queue length at the last–hop router is small and the data transfer time for short–lived connections are also small, while a high utilization of the access link bandwidth is maintained.

Figure 4 shows the change of the throughput of the long–lived connections in the simulation time. In this figure, we label the throughput of the connection from the sender host A as "flow A," that from the sender host B as "flow B" and that from the sender host C as "flow C," respectively. The label of "best" represents the throughput value when the access link bandwidth is shared most fairly and effectively. From Figure 4, our proposed scheme, Spring and Mehra show positive fairness among the long–lived connections, compared with the traditional scheme. However, Mehra shows the lowest throughput and the largest fluctuation of the throughput. This is because Mehra repeats the adjustment to make all TCP connections share the access link bandwidth equally. On the other hand, as we can see from Figure 4(a) and Figure 4(b), our proposed scheme and Spring show almost the same throughputs as the "best" case.

IV. CONCLUDING REMARKS

In this paper, we have proposed a receiver-based access link resource management scheme at the receiver host. Our proposed scheme adjusts virtually the amount of the receive socket buffer for all TCP connections at the user hosts in order to avoid congestion at the access link, and assigns it to each TCP connection so that for a short-lived connection, the packets be treated with high priority,

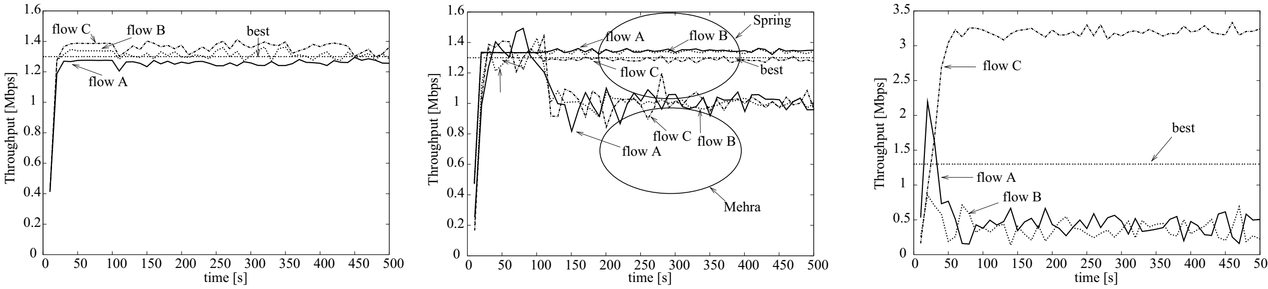


(a) Data Transfer Time of Short–Lived Connections

(b) Access Link Utilization

(c) Average Queue Length

Figure. 3. Simulation Results (1)



(a) Proposed Scheme

(b) Spring and Mehra

(c) Traditional Scheme

Figure. 4. Simulation Results (2)

and for a long-lived connection, the upper-layer application's QoS and the user's demands be reflected. We have evaluated the performance of the our proposed scheme through extensive simulation experiments, and confirmed that it can utilize effectively the access link resources, that is, it can improve the performance of short-lived TCP connections, and maintain the throughput of long-lived connections as expected, while keeping the utilization of the access link bandwidth. Moreover, we have compared our proposed scheme with the schemes in [12], [13] and confirmed the advantages of our proposed scheme.

As for future work, we plan to implement the proposed scheme to the actual receiver host, and to evaluate it through experiments using the actual network.

REFERENCES

[1] Proxy Survey, available at *http://www.delegate.org/survey/proxy.cgi*.

[2] T. Okamoto, T. Terai, G. Hasegawa, and M. Murata, "A resource/connection management scheme for HTTP proxy servers," in *Proceedings of Second International IFIP-TC6 Networking Conference*, pp. 252–263, May 2002.

[3] Broadband networking report in japanese, available at *http://www.musen-lan.com/speed/htmldata/*.

[4] L. Guo and I. Matta, "The war between mice and elephants," in *Proccedings of the 9th IEEE International Conference on Network Protocols*, no. 2001-005, Nov. 2001.

[5] W3C Recommendations Reduce 'World Wide Wait', available at *http://www.w3.org/Protocols/NL-PrefNote.html*.

[6] H. Balakrishnan, H. S. Rahul, and S. Seshan, "An integrated congestion management architecture for Internet hosts," in *SIGCOMM 1999*, pp. 175–187, Sept. 1999.

[7] J. Touch, "TCP control block interdependence," *Request for Comments (RFC) 2140*, Apr. 1997.

[8] J. Postel, "Transmission control protocol (TCP)," *Request for Comments (RFC) 793*, Sept. 1981.

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

[10] D. D. Clark, "Window and acknowledgement strategy in TCP," *Request for Comments (RFC) 813*, July 1982.

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

[12] N. T. Spring, M. Chesire, M. Berryman, V. Sahasranaman, T. Anderson, and B. N. Bershad, "Receiver based management of low bandwidth access links," in *Proccedings of IEEE INFOCOM 2000*, pp. 245–254, 2000.

[13] P. Mehra, A. Zakhor, and C. D. Vleeschouwer, "Receiver-driven bandwidth sharing for TCP," in *Proceedings of IEEE INFOCOM 2003*, Mar. 2003.

[14] The VINT Project, UCB/LBNL/VINT network simulator - ns (version 2), available at *http://www.isi.edu/nsnam/ns/*.