Scalable Automatic Buffer Tuning to Provide High Performance and Fair Service for TCP Connections

Takahiro Matsuo† **Go Hasegawa**‡ **Masayuki Murata**†

†Department of Informatics and Mathematical Science
Graduate School of Engineering Science, Osaka University
1-3, Machikaneyama, Toyonaka, Osaka 560-8531, Japan
Phone: +81-6-6850-6616, Fax: +81-6-6850-6589
E-mail: {t-matuo, murata}@ics.es.osaka-u.ac.jp

‡Faculty of Economics, Osaka University
1-7, Machikaneyama, Toyonaka, Osaka 560-0043, Japan
Phone: +81-6-6850-5233
E-mail: hasegawa@econ.osaka-u.ac.jp

Abstract Although many research efforts have been devoted to the network congestion against an increase of network traffic in the Internet, only a few discussions on the performance improvement of the endhosts are recently made. In this paper, we propose a new architecture, which is called Scalable Automatic Buffer Tuning (SABT), to provide high performance and fair service for many TCP connections at the Internet endhosts. In SABT, the sender host estimates 'expected' throughput of the TCP connections through the simple mathematical equations, and assign the send socket buffer to them according to the estimated throughput. If the socket buffer is short, the max-min fairness policy is utilized to allocate the send socket buffer. Since SABT uses the measurement-based approach to estimate the throughput, an initial values of the buffer size cannot be determined. Thus, an initial buffer size of the connection is determined by taking account of the characteristics of WWW traffic. We confirm an effectiveness of our proposed algorithm through both of a simulation technique and an actual experimental system where we implement our algorithm.

1 Introduction

According to a rapid growth of the Internet users, many research efforts have been devoted to how to avoid and dissolve the network congestion against an increase of network traffic. On the other hand, only a few discussions on the performance improvement of the Internet endhosts are recently made in spite of the projection that the bottleneck is now being shifted from the network to endhosts. For example, busy WWW (World Wide Web) servers in the current Internet receive hundreds of requests for document transfer every second at peak time.

Of course, the improvement of protocol processing on the endhosts is not a new subject. Such an example can be found in [1] where the authors propose the 'fbuf' (fast buffer) architecture, which shares the memory space between the system kernel and the user process to avoid redundant memory copies during data exchanges. It is based on an observation that memory copy is a main cause of the bottleneck at endhosts in TCP data transfer. However, the past researches including the above fbuf do not consider the 'fair' treatment of connections, by which we can expect more performance improvement by the following reasons. Suppose that a server host is sending TCP data to two clients of 64Kbps dial-up (say, user A) and 100Mbps LAN (user B). If the server host assigns an equal size of the socket buffer to both users, it is likely that the amount of the assigned buffer is too large for user A, and too small for user B, because of the difference of capacities (more strictly, bandwidth-delay products) of two connections. For an effective buffer usage for both users, a compromise on buffer usage should be taken into account.

Another important example that requires 'fair' buffer treatment can be found in a busy Internet WWW server, which accepts a large number of TCP connections with different bandwidths and round trip times (RTTs) at the same time. In [2], the authors have proposed a buffer tuning algorithm called Automatic Buffer Tuning (ABT), which dynamically adjusts the send socket buffer size according to the change of the TCP sending window size of the connection.

However, it does not provide 'fairness' among connections because the throughput of TCP connections is not proportional to the sending window size as shown in [3].

In this paper, we propose a novel architecture called Scalable Automatic Buffer Tuning (SABT), to provide high performance and fair service for many TCP connections at the sender host. For this purpose, we first derive 'expected' throughput of TCP connections by an analytic approach. It is characterized by packet loss rate, RTT and RTO (Retransmission Time Out) values of the connections, which can easily be monitored by the sender host. The send socket buffer is then assigned to each connection according to the expected throughput, with consideration on a max-min fairness among connections. We validate the effectiveness of our proposed mechanism through both of simulation and implementation experiments.

Another important factor that we should consider is how to determine an initial buffer size of the connection. It cannot be determined a priori since the control parameters such as the packet loss rate, RTT and RTO are not known at the connection setup time. One possible approach is to take into account the characteristics of WWW traffic when our algorithm is applied to the WWW server. Due to recent advancements of researches on Web traffic characterization (see, e.g., [4]), the initial buffer size can be determined based on those results such that the server can send the whole document in one RTT if the document size is small, by which response times of the Web document retrieval can be much improved.

This paper is organized as follows. In Section 2, we first introduce the ABT algorithm briefly for reference purpose, and propose our SABT algorithm. We evaluate the effectiveness of our proposed algorithm through simulation experiments in Section 3, followed by implementation experiments in Section 4. Finally, we present some concluding remarks in Section 5.

2 Automatic Tuning of Send Socket Buffer

As explained in the previous section, when the sender host accepts multiple TCP connections simultaneously, the send socket buffer size of the sender host must be assigned to the connections, by taking care of differences of the connections' characteristics. In this section, we first introduce related researches on the buffer tuning, and point out several problems in Subsection 2.1. We next present our proposed algorithm, called Scalable Automatic Buffer Tuning in Subsection 2.2.

2.1 Related Work; Automatic Buffer Tuning

In [2], the authors have proposed an "Automatic Buffer Tuning" mechanism (referred to as *ABT* in this paper), where the assigned buffer size of each TCP connection is determined according to the current window size of the connection. That is, when the window size becomes large, the sender host tries to assign more buffer to the connection. On the other hand, as the window size becomes small, it decreases the assigned buffer size.

When the total required buffer size of all TCP connections becomes larger than the send socket buffer size prepared at the sender host, the send socket buffer is assigned to each connection according to a max-min fairness policy. More specifically, the sender host first assigns the buffer equally to all TCP connections. Then, if there exists connections which do not require large buffer, the excess buffer is re-assigned to connections which requires larger buffer. Through this mechanism, it is expected to provide dynamic and fair buffer assignment by considering differences of TCP connections.

However, ABT has several problems. It assigns the send socket buffer to each TCP connection according to its current window size at regular intervals. Therefore, when sender TCP changes its window size suddenly due to, e.g., packet loss, the assigned buffer size sometimes gets smaller than that the connection actually requires. It might be solved by setting the up-

date interval to a smaller value. In that case, however, the assigned buffer size is changed too frequently, which causes the system instability. Furthermore, as the network bandwidth becomes larger, the oscillation of the window size also becomes large, leading to a large oscillation of the assigned buffer size.

Another problem exists in the max-min sharing policy adopted in ABT. Suppose that three TCP connections (connections 1, 2 and 3) are active, and the required buffer sizes calculated from the window size of each connection are 20 [KBytes], 200 [KBytes], and 800 [KBytes], respectively. If the total size of the send socket buffer is 300 [KBytes], the sender host first assigns 100 [KBytes] to each connection. Since connection 1 does not require such a large buffer, the sender re-assigns the excess buffer of 80 [KBytes] of connection 1 *equally* to connections 2 and 3. As a result, the assigned buffer sizes of connections 2 and 3 become 140 [KBytes]. However, it must be better to assign the excess buffer *proportionally* to the required buffer size of connections 2 and 3. In this case, the assigned buffers become 116 [KBytes] for connection 2 and 164 [KBytes] for connection 3 by a *proportional* re-assignment. This assignment is more effective because the throughput improvement of connection 3 becomes larger. The remaining problem is to estimate how much buffer each TCP connection actually requires. ABT cannot re-assign the excess buffer proportionally because it determines the buffer size only by the current window size of the connection.

2.2 Proposed Method; Scalable Automatic Buffer Tuning

Our proposed method called a “Scalable Automatic Buffer Tuning” (SABT) includes the following two mechanisms; the stable and fair buffer assignment and the improved transfer time for small files.

2.2.1 Stable and Fair Buffer Assignment Mechanism

One of undesirable features of ABT is the instability of the buffer assignment. It is because ABT only observes the current window size of the connection, which oscillates largely due to the congestion control mechanism of TCP. In the proposed method, on the other hand, the sender host first estimates an 'expected' throughput of each TCP connection from three parameters, and determines the required buffer size of the connection from the estimated throughput, not from the current window size of TCP.

The estimation method of TCP throughput is based on the analysis result of the previous work [5]. In [5], the average throughput of the TCP connection is derived for the model, where multiple connections with different input link bandwidths share the bottleneck router employing the RED algorithm [6]. The following parameters are used to derive the average throughput;

- p: packet dropping probability of the RED algorithm
- rtt: average value of RTT (Round Trip Time) of the TCP connection
- rto: average value of the retransmission timer of the TCP connection

In the analysis, an average window size of the TCP connection is first calculated from the above three parameters. The average throughput is then obtained by considering the performance degradation caused by TCP's retransmission timeout expiration. The analysis in [5] is easily applied to our case, by viewing the packet dropping probability of the RED algorithm as the observed packet loss rate.

The parameter set (p, rtt, and rto) can be obtained by the sender host as follows. Rtt and rto can be directly obtained from the sender TCP. The packet loss rate p can also be estimated from the number of successfully transmitted packets, and the number of lost packets detected at the sender host via acknowledgement packets. A possible cause of the estimation error on p is related to the stochastic nature of the packet losses since the analysis in [5] assumes the random packet loss. Thus, we need the validation for the case where the packet losses occur at

the drop-tail router, since in that case, the packets tends to be dropped in the bursty nature [6]. We will present results on this aspect in Section 3.

We denote the estimated throughput of connection i by $\overline{\rho_i}$. From $\overline{\rho_i}$, we determine B_i, the required buffer size of connection i, as;

$$B_i = \overline{\rho_i} \times rtt_i \quad (1)$$

where rtt_i is RTT of connection i. By this mechanism, it is expected to provide stable assignment of the send socket buffer to TCP connections, if the parameter set (p, rtt, and rto) used in estimation is stable. In ABT, on the other hand, the assignment is instable even when three parameters are stable, since the window size oscillates more significantly regardless of the stability of the parameters.

As in ABT, our SABT also adopts the max-min fairness policy for re-assigning the excess buffer. Differently from the ABT algorithm, however, SABT employs the proportional re-assignment policy as explained in the previous subsection. That is, when the excess buffer is re-assigned to connections which need more buffer, the buffer is re-assigned to connections in proportion to the required buffer size calculated from the analysis. Note that ABT re-assigns the excess buffer equally, since it has no means to know the expected throughput of the connections.

2.2.2 Improved Transfer Time for Small Files

In the proposed method, the sender estimates the throughput of each TCP connection from observed parameters associated with that connection. Therefore, if the size of transmitted data is small, it is impossible to obtain the accurate and reliable estimation. Especially when the proposed mechanism is applied to the WWW server, this problem becomes an obstacle since the Web documents are comparatively small. Note that it is reported in [4] that the average

size of Web documents at the several Web servers is under 10 [KBytes].

Therefore, an initial setting of the send socket buffer size becomes important. In the proposed mechanism, if a certain TCP connection is going to transmit a small file, the sender host excludes it from the fair buffer assignment described above. Instead, the sender sets both of the initial window size and the send socket buffer size of the connection to the size of transmitted file itself. By this modification, it is expected that TCP connection transfers the file in one RTT.

In our setting, if the file size is smaller than 8 [KBytes], the sender host excludes it from the fair assignment, since 80 % of WWW documents is smaller than 8 [Kbytes] according to [4]. Then, we can expect that the transfer time of small WWW documents can be improved , and SABT provides stable and accurate assignment of the send socket buffer to the remaining connections transmitting larger files.

3 Simulation Results

In this section, we present some simulation results using the network simulator `ns` [7]. We compare the following three algorithms in order to validate the effectiveness of our proposed mechanism.

EQ: All of active TCP connections are assigned equal sizes of the send socket buffer.

ABT: The send socket buffer size is assigned according to the automatic buffer tuning algorithm described in Subsection 2.1.

SABT: The sender host assigns the send socket buffer in accordance with the scalable automatic buffer tuning algorithm described in Subsection 2.2.

In three algorithms, the buffer re-assignment is performed at every second.

In the simulation experiment, we conducted two cases for packet loss occurrence. In the first case, packet loss takes place with a constant rate assuming that the router is equipped with

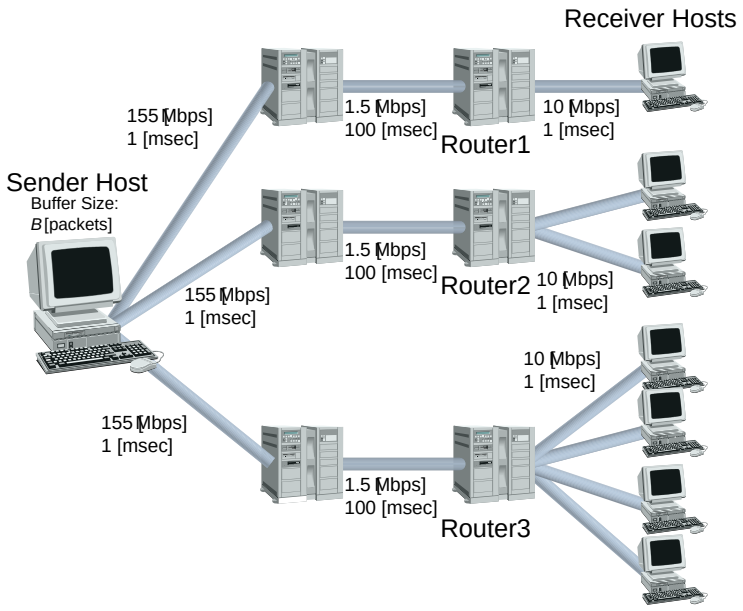


Figure 1: Network Model for Simulation Experiment

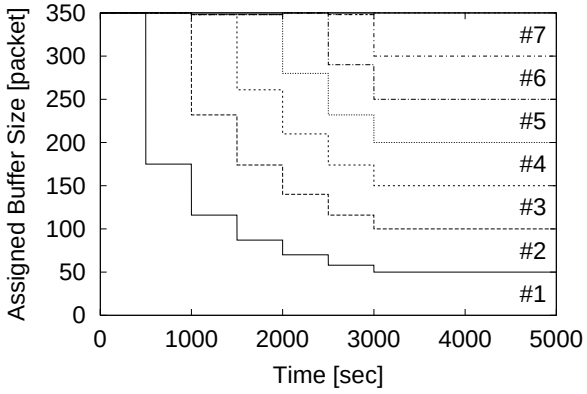
the RED algorithm. It means that the packet loss takes place randomly with given probability. Such an assumption is validated by [3]. However, it is true that random packet dropping is a rather ideal case to our SABT since our method largely relies on adequate estimation of the packet loss rate in determining the throughput of TCP connections. Hence, we also consider the case of the drop-tail router where packet loss occurs by the buffer overflow at the router. In this case, we can never expect random packet loss, and it tends to occur in a bursty fashion. In this section, we only show results of the drop-tail router case due to space limitation. The effectiveness of the SABT algorithm is more apparent in the constant packet loss case, due to the reason above.

Figure 1 depicts the network model used for simulation experiments. The model consists of a sender host, 7 receiver hosts, 6 routers, and links interconnecting routers and sender/receiver hosts. 7 TCP connections are established at the sender host (connections 1 through 7). As shown in the figure, connection 1 occupies router 1, connection 2 and 3 share router 2, and the rest of connections 4 through 7 share router 3. The capacities of links between routers

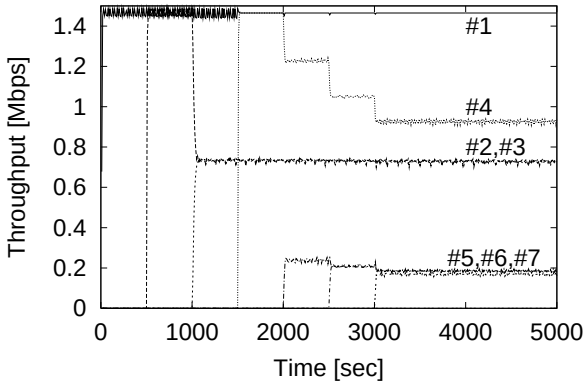
are identically set to be 1.5 [Mbps], and those of links between routers and the sender host, and between routers and the receiver hosts are set to be 155 [Mbps] and 10 [Mbps], respectively. Further, propagation delays of links between routers, and that between routers and sender/receiver hosts are 100 [msec] and 1 [msec], respectively. In this simulation experiment, connection i starts sending packets at time $t = (i - 1) \times 500$ [sec]. The simulation ends at 4,000 [sec]. Therefore, when seven connections join the network, the ideal throughput becomes 1.5 [Mbps] for connection 1, 0.75 [Mbps] for connections 2 and 3, and 0.375 [Mbps] for connections 4 through 7.

Figure 2 shows the results for the throughput and assigned buffer size as a function of time. We set the send socket buffer size $B = 350$ [packets]. As shown in Figures 2(a) and 2(b), connections sharing router 3 (connections 4 through 7) receive unfair throughput values in the EQ case, in spite of the fact that conditions of four connections are completely identical. Perhaps, it is due to an accidental unfairness of the TCP congestion control algorithm, which is well known in the literature. See, e.g., [8]. Figures 2(c) and 2(d) clearly show that ABT cannot provide a stable buffer assignment. It is because ABT assigns the buffer to each TCP connection according to the current window size of the connection, which oscillates dynamically due to an inherent nature of the TCP window mechanism. Therefore, the assigned buffer sizes of connections which do not need large buffer (connection 2 through 7 in the current case) are often inflated, leading to the temporary decrease of the buffer assigned to connections which need large buffer (connection 1). As a result, the throughput of connection 1 is often degraded. On the other hand, our proposed SABT keeps stable and fair buffer assignment as shown in Figure 2(e), leading to fair treatment in terms of throughput (Figure 2(f)).

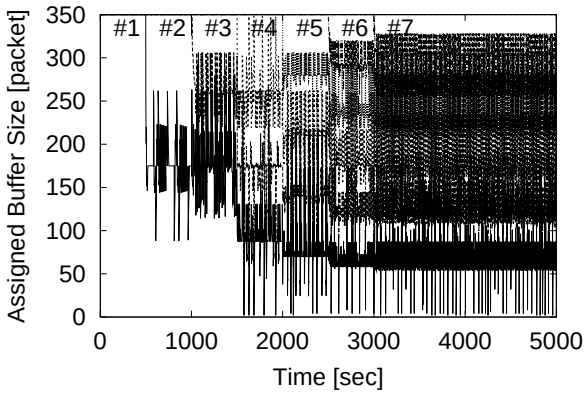
Even when the buffer size is smaller, SABT keeps its effectiveness. Results are shown in Figure 3, where we decrease the buffer size B to 200 [KBytes]. Figures 3(b) and 3(d) show that EQ and ABT mechanisms cannot provide fairness among connections. In particu-



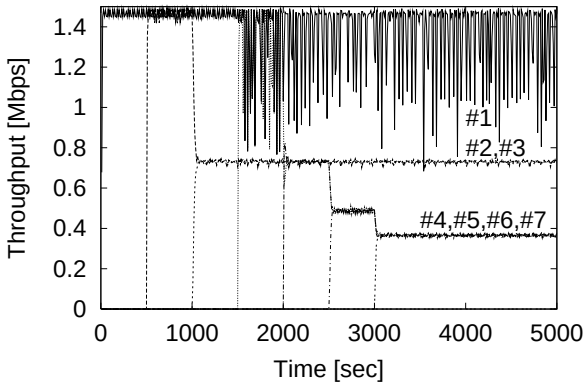
(a) EQ Case: Changes of Assigned Buffer Size



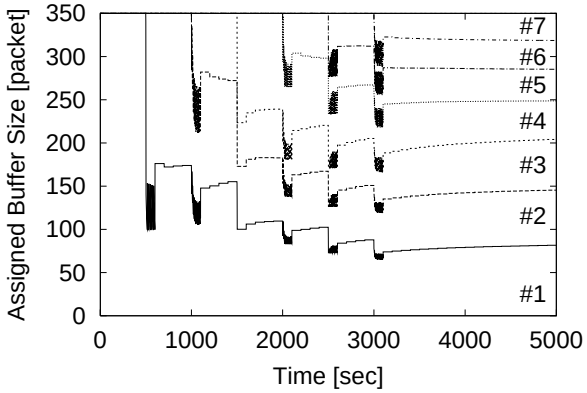
(b) EQ Case: Changes of Throughput



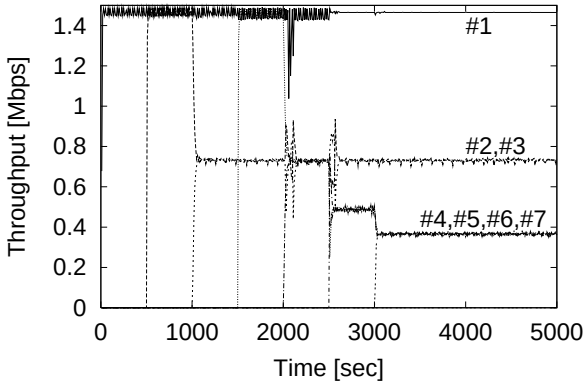
(c) ABT Case: Changes of Assigned Buffer Size



(d) ABT Case: Changes of Throughput



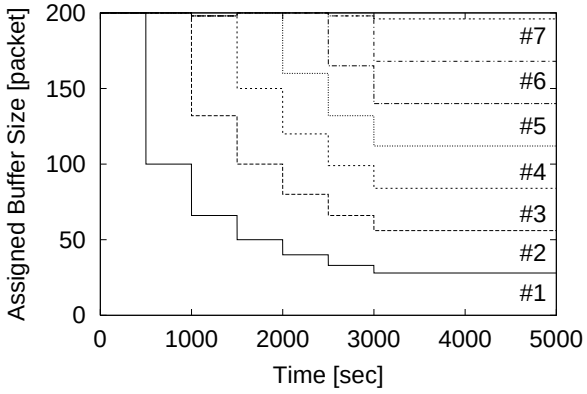
(e) SABT Case: Changes of Assigned Buffer Size



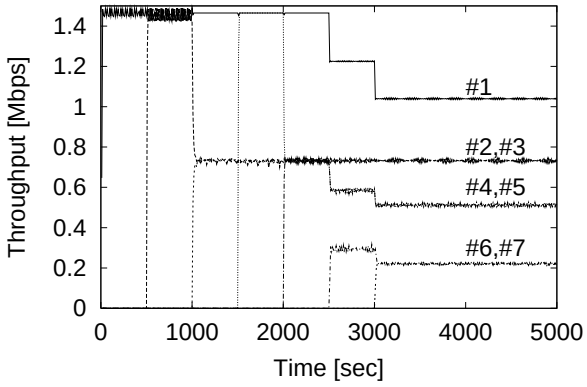
(f) SABT Case: Changes of Throughput

Figure 2: Comparative Results of Simulation Experiment: $B = 350$ [packets]

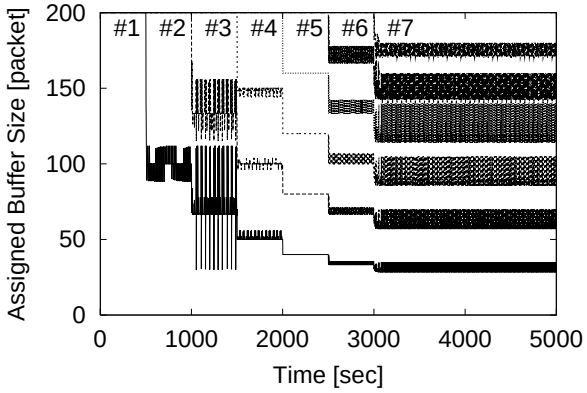
lar, throughput of connection 1 is significantly degraded in those cases, and connections connected to router 3 receive different throughput values. It is because in assigning the buffer,



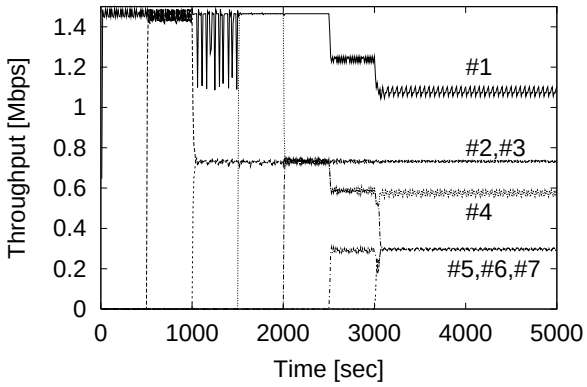
(a) EQ Case: Changes of Assigned Buffer Size



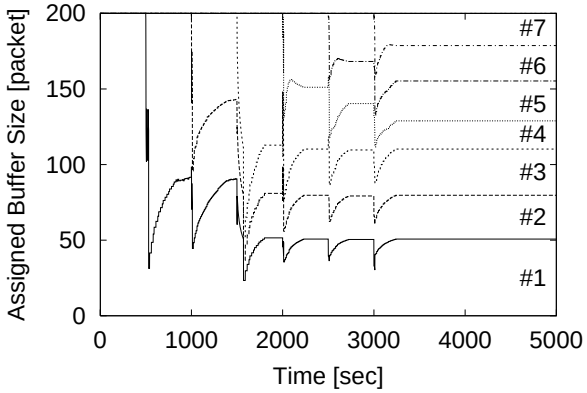
(b) EQ Case: Changes of Throughput



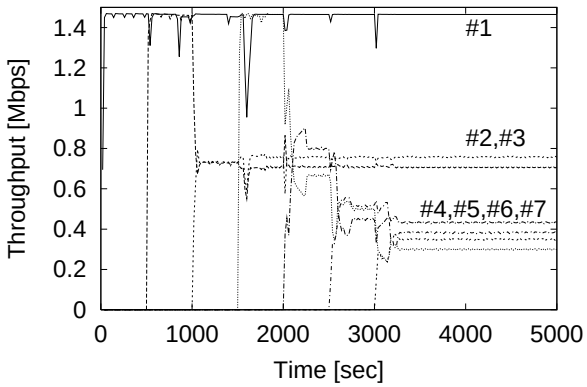
(c) ABT Case: Changes of Assigned Buffer Size



(d) ABT Case: Changes of Throughput



(e) SABT Case: Changes of Assigned Buffer Size



(f) SABT Case: Changes of Throughput

Figure 3: Comparative Results of Simulation Experiment: $B = 200$ [packets]

the EQ method does not take care of the connections' characteristics at all. See Figure 3(a). It is also true for the ABT mechanism (Figure 3(c)). It is mainly due to the oscillation of

the assigned buffer to each connection as shown in Figures 3(c). From the figure, it is also verified that ABT's re-assignment algorithm, which equally re-assigns the excess buffer to connections, does not work well. On the other hand, in the SABT algorithm, throughput of connection 1 is kept to be a high value, and much better fairness can be achieved (Figures 3(e) and 3(f)). Note that a slight difference of the throughput values of connections 2 and 3 (and connections 4 through 7) is due to an estimation error of the required buffer sizes. In the case of the drop-tail router, the parameters (p, rtt, and rto) observed for the throughput estimation changes more largely than the constant packet loss rate case because of the bursty nature of packet losses, leading to the difference of the assigned buffer size.

4 Implementation Experiments

In this section, we present the results obtained by our experimental system. We implemented EQ, ABT and SABT mechanisms on Intel Pentium-III 700 [MHz] PC running FreeBSD 2.2.8, and the two machines are directly connected. Through the experiments, we focus on the following three subjects;

1. fair buffer assignment among different connections
2. scalability against the number of connections
3. improvement of transfer time for small files

4.1 Experiment 1: Fair Buffer Assignment among Different Connections

We first evaluate the fairness property of three buffer assignment algorithms. In this experiment, we consider the situation where three TCP connections are established at the sender host through the 622 [Mbps] MAPOS (Multiple Access Protocol Over SONET/SDH) [9] link. Three connections have different packet loss rates; 0.005 for connection 1, 0.01 for connec-

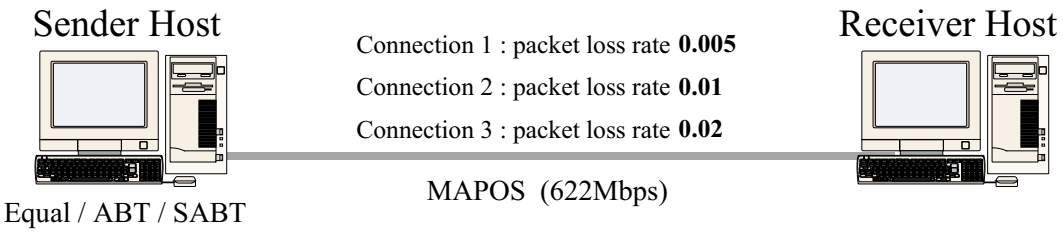
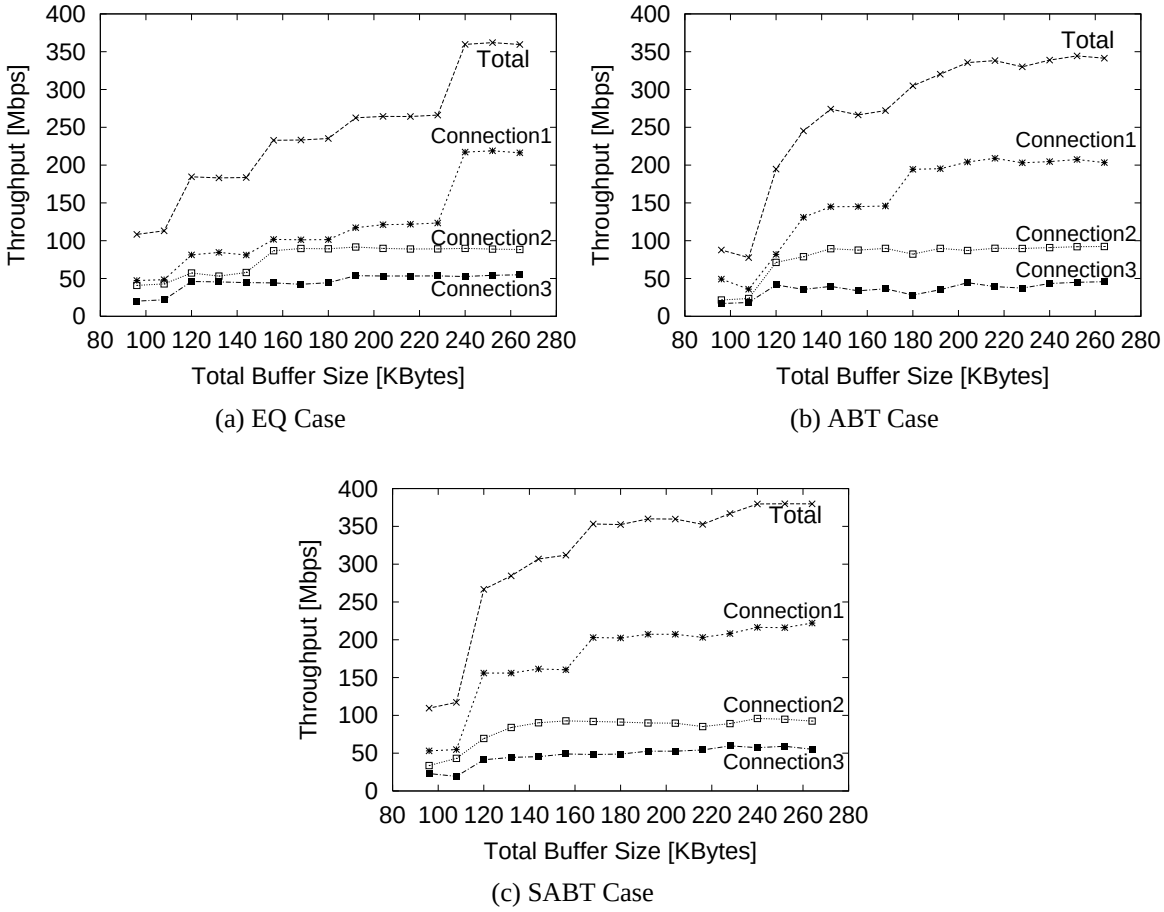


Figure 4: Network Environment for Experiment 1

tion 2, and 0.02 for connection 3, as shown in Figure 4. In the experiment, packets are intentionally dropped at the receiver host. Note that with those packet loss rates, three connections can achieve throughput values of about 55, 90, and 220 [Mbps], respectively, if an enough amount of the send socket buffer is assigned to each of connections.

Figure 5 compares the throughput values of three connections and total throughput. The horizontal axis is the total size of the send socket buffer. In the EQ case (Figure 5(a)), three connections can achieve their maximum throughputs when the total buffer size is larger than 240 [KBytes], while ABT and SABT need only about 200 [KBytes] and 170 [KBytes], respectively. It is due to the same reason explained in Section 3. Connections 2 and 3 do not need as much buffer size as connection 1 does, but the EQ algorithm cannot re-allocate the excess buffer of connections 2 and 3 to connection 1. On the contrary, ABT and SABT algorithms work adequately for the buffer assignment.

We look at results of around 120 [KBytes] of the total buffer size in the figure. By comparing ABT and SABT algorithms, it is clear that SABT can provide much higher throughput for connection 1. The difference is due to re-assignment methods of excess buffer to connections. In the ABT algorithm, the excess buffer of connection 3 is re-assigned to connections 1 and 2 equally. Then, only connection 3 is assigned a sufficient size of the buffer while connections 1 and 2 need more buffer. On the other hand, SABT re-assigns the excess buffer in proportion to the required buffer size of connections 1 and 2. Accordingly, SABT gives more buffer to connection 1 than connection 2 because the required buffer size of connection 1 is



(a) EQ Case

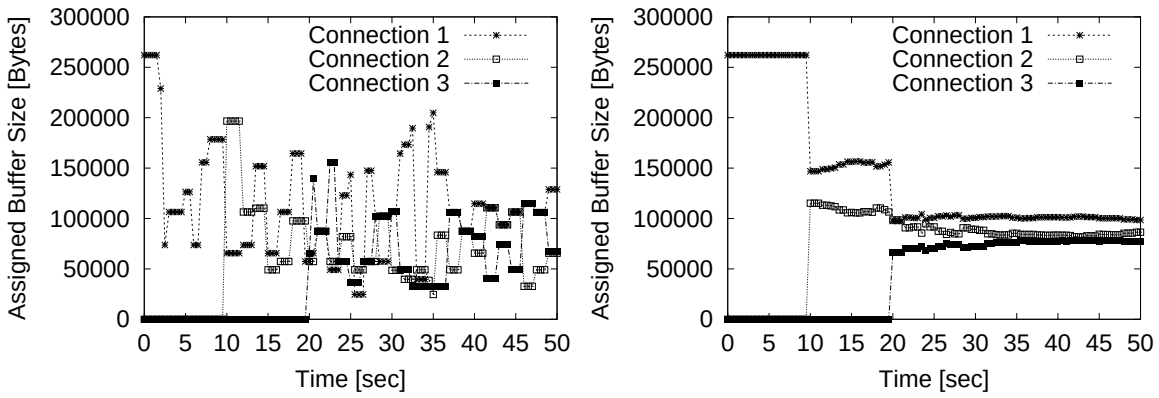
(b) ABT Case

(c) SABT Case

Figure 5: Fairness among Three Connections

larger than that of connection 2. Consequently, connection 1 can achieve higher throughput at expense of small throughput degradation of connection 2. As a result, the total throughput of SABT is highest among three algorithms regardless of the total buffer size as shown in Figure 5.

We next a present time–dependent behavior of the buffer size assigned to each of three connections in Figure 6. In this experiment, connections 1, 2, and 3 start packet transmission at time 0 [sec], 10 [sec], and 20 [sec], respectively. Here, we set the total buffer size to be 256 [KBytes]. In ABT (Figure 6(a)), the assigned buffer sizes are heavily oscillated because the ABT algorithm adapts the buffer assignment according to the window size. An-



(a) ABT Case (b) SABT Case

Figure 6: Changes of Assigned Buffer Sizes

other and more important reason is that when retransmission timeout expiration occurs at a certain connection, that connection resets the window size to 1 [packet] according to the TCP retransmission mechanism. Then, the assigned buffer size of that connection becomes very low. Once the assigned buffer gets small, the connection cannot inflate its window size during a while because of the throttled buffer size. It is the reason why the assigned buffer size is kept low in the ABT algorithm. On the other hand, SABT can provide the stable and fair buffer assignment as shown in Figure 6(b). It brings high throughput to each TCP connection as having been shown in Figure 5(c).

4.2 Experiment 2: Scalability against the Number of Connections

We next turn our attention to the scalability of the buffer assignment algorithms against the number of connections. Figure 7 depicts the experimental setting for this purpose. One TCP connection is established using the MAPOS link (which is referred to as *MAPOS connection* below). The several numbers of TCP connections are simultaneously established through the Ethernet link (*Ethernet connections*). By this experiment, we investigate the effect of the

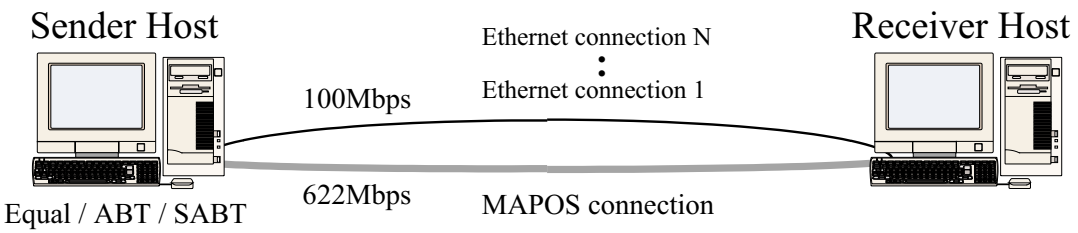


Figure 7: Network Environment for Experiment 2

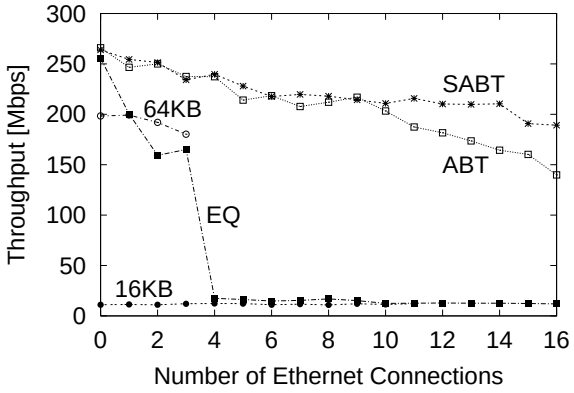


Figure 8: Throughput of MAPOS Connection vs. the Number of Ethernet Connections

number of Ethernet connections on the performance of the MAPOS connection.

Figure 8 shows the throughput values of the MAPOS connection dependent on the number of Ethernet connections. In addition to the result of EQ, ABT and SABT algorithms, we also present the results for the cases where the constant size (16 [KBytes] or 64 [KBytes]) of the send socket buffer is assigned to each TCP connection. Such a constant assignment is a default mechanism of the current TCP/IP implementation in major operating systems. Results are plotted in the figure with labels “16KB” and “64KB.” In this figure, we set the total of the send socket buffer to 256 [KBytes]. Therefore, if we assign the constant buffer size of 64 [Kbytes] to each TCP connection, the sender allows up to four connections at the same time.

We can make several important observations from this figure. First, we can see that the constant assignment algorithm, which is widely employed in the current OS, has drawbacks.

If 16 [KBytes] send socket buffer is assigned to each TCP connection, the MAPOS connection suffers from very low throughput because it is too small for the 622 [Mbps] MAPOS link. When each connection is given 64 [KBytes] buffer size, on the other hand, the throughput of the MAPOS connection becomes considerably high as shown in Figure 8. However, the number of connections which can be simultaneously established is severely limited.

In the EQ algorithm, when the number of Ethernet connections exceeds four, the throughput of the MAPOS connection is suddenly decreased to about 11 [Mbps]. It is because the EQ algorithm does not distinguish the MAPOS connection and Ethernet connections, and assigns an equal size of the send socket buffer to all connections. Therefore, as the number of Ethernet connections is increased, the assigned buffer size to the MAPOS connection is decreased, leading to throughput degradation of the MAPOS connection.

When we employ the ABT or SABT algorithm, the throughput degradation of the MAPOS connection can be limited even when the number of Ethernet connections is increased. However, when the number of Ethernet connections is beyond twelve, throughput values of ABT and SABT algorithms becomes distinguishable as shown in Figure 8. It is again caused by the instability of the assigned buffer size and by the poor re-assignment algorithm of ABT, as having been explained in the previous subsection. Even in the SABT algorithm, the throughput of the MAPOS connection is slightly degraded by the larger number of Ethernet connections. It is because the total of the required buffer size of Ethernet connections is increased, Even though the required buffer of Ethernet connections is small. Then, the buffer assigned to the MAPOS connection is decreased because the excess buffer becomes small. However, the degree of the throughput degradation of the MAPOS connection can be limited in the SABT algorithm.

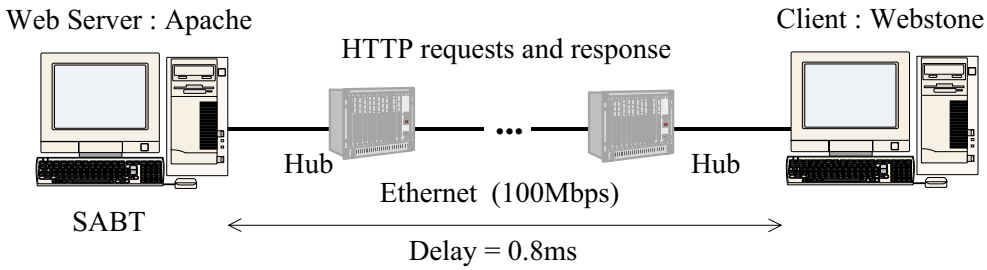


Figure 9: Network Environment for Experiment 3

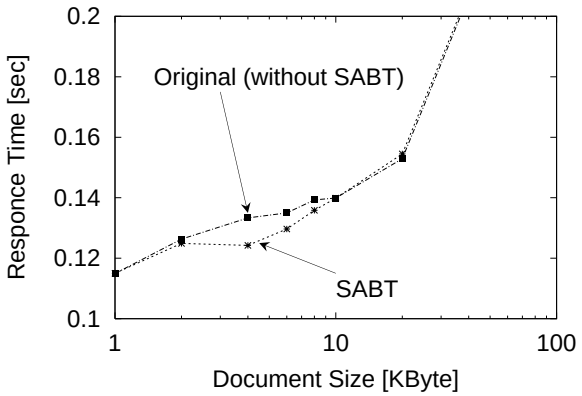


Figure 10: Response Time vs. Document Size

4.3 Experiment 3: Improvement of Transfer Time of Small Files

We last evaluate the effectiveness of our algorithm to improve transfer times of small files, which was explained in Subsection 2.2.2. The experimental environment is depicted in Figure 9. We implement the SABT algorithm on the Web server host, running apache WWW server version 1.3.3 [10]. On the client host, we use Webstone [11], a benchmark tool to automatically send the document transfer requests to the server host and to obtain the several statistics including the document transfer time. We set the distribution of requested document size according to the results in [4], which is drawn by analyzing the log at the Web site. Then, the client randomly chooses the size of the requested document according to the prescribed probability distribution. The propagation delay between the server and client hosts is about 0.8 [msec].

Figure 10 shows the average response time of the document transfer requests against the document size. Here, the response time is defined as the time duration from when the client transmits the document transfer request to the time when the client receives the requested document. In the figure, we plot the results of the SABT algorithm and the original mechanism. By the original mechanism, we mean that no buffer assignment policy is utilized and a constant value of the buffer size (16 [KBytes]) is assigned to the connection. We can see from the figure that when the document size is smaller than 8 [KBytes], the response time of SABT becomes smaller than the original mechanism as expected. That is, the SABT algorithm sets the initial window size and the buffer size of the connection to be the size of the requested document if the document size is smaller than 8 [KBytes]. Then, such connections can transfer the file in one RTT, leading to an improvement of the transfer time. Therefore, we can expect that as the propagation delay between the server and the client hosts becomes large, the effect on reduction of the document transfer time also gets large. Also, Figure 10 clearly shows that SABT algorithm can reduce transmission time of small files, without any increase of transmission times of large files.

5 Conclusion

In this paper, we have proposed SABT (Scalable Automatic Buffer Tuning), a novel architecture for assigning the send socket buffer of the busy Internet server to TCP connections with different characteristics. In SABT, the "expected" throughput of each TCP connection is derived from three parameters, which can be easily estimated at the sender host. The sender host then assigns the buffer to the connections according to the estimated throughput, taking care of the max-min fairness among active TCP connections. Further, we have proposed how to determine the initial buffer size and the initial window size of TCP connections, according to the typical distribution of the WWW document size. We have confirmed the effectiveness

of SABT algorithm through both of simulation and implementation experiments, and have shown that SABT can assign the send socket buffer to each TCP connection in a fair and effective way than ABT (Automatic Buffer Tuning) and other algorithms.

References

[1] P. Druschel and L. L. Peterson, "Fbufs: a high-bandwidth cross-domain transfer facility," in *Proceedings of the Fourteenth ACM symposium on Operating Systems Principles*, pp. 189–202, December 1993.

[2] J. Semke, J. Mahdavi, and M. Mathis, "Automatic TCP buffer tuning," in *Proceedings of ACM SIGCOMM'98*, pp. 315–323, August 1998.

[3] 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*, pp. 303–314, August 1998.

[4] M. Nabe, M. Murata, and H. Miyahara, "Analysis and modeling of world wide web traffic for capacity dimensioning of internet access lines," *Performance Evaluation*, vol. 34, pp. 249–271, December 1999.

[5] T. Matsuo, G. Hasegawa, M. Murata, and H. Miyahara, "Comparisons of packet scheduling algorithms for fair service among connections," in *Proceedings of Internet Workshop '99*, pp. 193–200, February 1999.

[6] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, pp. 397–413, August 1993.

[7] Network Simulator - `ns` (version 2), available from *`http://www-mash.cs.berkeley.edu/ns/`*.

[8] S. Floyd and V. Jacobson, "On traffic phase effects in packet-switched gateways," *Internetworking: Research and Experience*, vol. 3, pp. 397–413, August 1992.

[9] K. Murakami and M. Maruyama, "MAPOS - multiple access protocol over SONET/SDH, version 1," *Request for Comments 2171*, June 1997.

[10] Apache Home Page, available from *`http://www.apache.org/`*.

[11] Silicon Graphics Inc., "WebStone: World wide web server benchmarking," available from *`http://www.sgi.com/Products/WebFORCE/Resources/res_webstone.html`*, 1996.