Scalable Socket Buffer Tuning for High-performance Web Servers

Go Hasegawa, Tatsuhiko Terai, Takuya Okamoto and Masayuki Murata

Department of Infomatics and Mathematical Science

Graduate School of Engineering Science, Osaka University

1-3, Machikaneyama, Toyonaka, Osaka 560-8531, Japan

Phone: +81-6-850-6616, Fax: +81-6-850-6589

E-mail: {hasegawa, terai, tak-okmt, murata}@ics.es.osaka-u.ac.jp

Abstract

Although many research efforts have been devoted to network congestion in the face of an increase in the Internet traffic, there is little recent discussion on performance improvements for endhosts. In this paper, we propose a new architecture, called Scalable Socket Buffer Tuning (SSBT), to provide high-performance and fair service for many TCP connections at Internet endhosts. SSBT has two major features. One is to reduce the number of memory accesses at the sender host by using some new system calls, called Simple Memory-copy Reduction (SMR) scheme. The other is Equation-based Automatic TCP Buffer Tuning (E-ATBT), where the sender host estimates 'expected' throughput of the TCP connections through a simple mathematical equation, and assigns a send socket buffer to them according to the estimated throughput. If the socket buffer is short, max-min fairness policy is used. We confirm the effectiveness of our proposed algorithm through both simulation technique and an experimental system. From the experimental results, we have found that our SSBT can achieve up to a 30% gain for Web server throughput, and a fair and effective usage of the sender socket buffer can be achieved.

Keywords: TCP (Transmission Control Protocol), Socket Buffer, Fairness, Buffer Tuning, Scalability

1 Introduction

With the rapid growth of Internet users, many research efforts have been directed to avoiding and dissolving network congestion against an increase of network traffic. However, there has been little recent discussion on improvement of the performance of Internet endhosts 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 currently receive hundreds of requests for document transfers every second at peak times.

Of course, proposals for the improvement of protocol processing by endhosts are not new. An early example can be found where the authors proposed the 'fbuf' (fast buffer) architecture, which shares memory space between the system kernel and the user process to avoid redundant memory copies during data exchanges [1]. It is based on the observation that memory copy is a main cause of the bottleneck at endhosts in TCP data transfer. Other approaches can also be found [2, 3]. However, past research including the above approaches do not consider the 'fair' treatment of connections, in which connections receive fair service from the server. By achieving fair service among connections, we can also expect performance improvement for the following reasons. Suppose that a server host is sending TCP data to two clients, one of 64Kbps dial-up (say, client A) and the other of 100Mbps LAN (client B). If the server host assigns equal sizes of the send socket buffer to both clients, it is likely that the amount of the assigned buffer is too large for client A, and too small for client B, because of the difference of capacities (more strictly, bandwidth-delay products) of their connections. For an effective buffer allocation to both clients, 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 simultaneously accepts a large number of TCP connections with different bandwidths and round trip times (RTTs) at the same time. The authors have proposed a buffer tuning algorithm called Automatic TCP Buffer Tuning (referred to as *ATBT* in this paper), that dynamically adjusts the send socket buffer size according to the change of the TCP sending window size of the connection [4]. However, it cannot provide 'fairness' among connections because the throughput of TCP connections is not proportional to the sending window size [5]. We will discuss the problems of ATBT more in the next section.

In this paper, we propose a novel architecture, called Scalable Socket Buffer Tuning (SSBT), to provide high performance and fair service for many TCP connections at the sender host. For this purpose, we developed the following two mechanisms for SSBT. These are the SMR (Simple Memory-copy Reduction scheme) and E-ATBT (Equation-based Automatic TCP Buffer Tuning). The SMR scheme provides a set of socket system calls in order to reduce the number of memory copy operations at the sender host in TCP data transfer. It is a replacement of the existing so-called 'zero-copy' mechanisms. However, it is much simpler and can still achieve the same effect in reducing the overhead at the sender host. However, our main contribution in this paper is E-ATBT, which provides a fair assignment of the socket buffer. In E-ATBT, we consider 'expected' throughput of TCP connections by an analytic approach. It is characterized by the 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 based on its expected throughput, with consideration on max-min fairness among connections.

We validated the effectiveness of our proposed mechanism through both simulation and implementation experiments. In the simulation experiments, we confirmed the fair assignment of the socket buffer using our E-ATBT algorithm under various network situations. In the implementation experiments, we first confirmed the behavior of SSBT at the transport-layer, that is, we used native TCP data transfer in the ex-

periments, followed by experiments on a SSBT-implemented Web server. We showed the effectiveness of SSBT in terms of overall server performance, the number of accommodated connections, and fairness among connections.

We also noted that fairness is recently being paid more attention from various aspects; for TCP enhancements [6, 7, 8] and for router support [9, 10, 11] in order to achieve that fairness. Perhaps, all of the above technologies including our proposed method for endhosts are necessary to provide *end–to–end* fairness to users.

This paper is organized as follows. In Section 2, we first briefly introduce the ATBT algorithm for reference purposes. In Section 3, we propose our SSBT algorithm. We evaluate the effectiveness of our proposed algorithm through simulation experiments in Section 4, followed by implementation experiments in Section 5. Finally, we present some concluding remarks in Section 6.

2 Related Work; Automatic TCP Buffer Tuning (ATBT)

To provide the fairness among multiple TCP connections, the sender host has to assign its send socket buffer to the connections by taking care of differences in the connections' characteristics. In this section, we first introduce related research on buffer tuning, and point out several problems.

An Automatic TCP Buffer Tuning (ATBT) mechanism has been proposed, where the assigned buffer size of each TCP connection is determined according to the current window size of the connection [4]. That is, when the window size becomes large, the sender host tries to assign more buffer to the connection. It decreases the assigned buffer size as the window size becomes small. When the total required buffer size of all TCP connections becomes larger than the send socket buffer size prepared at the sender host, 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. If some connections do not require a large buffer, the excess buffer is re-assigned to connections requiring a larger buffer. Through this mechanism, it is expected that a dynamic and fair buffer assignment can be provided by considering different characteristics.

However, ATBT has several problems. It assigns the send socket buffer to each TCP connection according to its current window size at regular intervals. Therefore, when the sender TCP suddenly decreases its window size due to, e.g., an occasional packet loss, the assigned buffer size sometimes gets smaller than that the connection actually requires. This might be resolved by setting the update interval to a smaller value. In that case, however, the assigned buffer size is changed too frequently, which causes the system instability. Furthermore, as 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 ATBT. Suppose that three TCP connections (connections 1, 2 and 3) are active, and the required buffer sizes calculated from the window sizes of connections 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 both connections 2 and 3 become 140 [KBytes]. However, it would 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 how to estimate the buffer size each TCP connection actually requires. ATBT cannot re-assign the excess buffer proportionally because it determines the buffer size only by the current window size of the connection.

3 Scalable Socket Buffer Tuning (SSBT)

Our proposed "Scalable Socket Buffer Tuning" (SSBT) method includes two mechanisms, (1) the Equation-based Automatic TCP Buffer Tuning (E-ATBT) and (2) the Simple Memory-copy Reduction (SMR) scheme, as explained below.

3.1 Equation-based Automatic TCP Buffer Tuning (E-ATBT)

E-ATBT solves the problems raised in Section 2. In E-ATBT, the sender host estimates an 'expected' throughput for each TCP connection by monitoring three parameters (packet loss probability, RTT, and RTO values. It then 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 in Hasegawa et al [12]. There the authors derived the average throughput of a TCP connection for a model in which multiple connections with different input link bandwidths share a bottleneck router employing the RED algorithm [13]. The following parameters are necessary to derive the throughput;

- p: packet dropping probability of the RED algorithm
- rtt: the RTT value of the TCP connection
- rto: the RTO value of the TCP connection

In the analysis, the 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 the TCP's retransmission timeout. The analysis developed in Hasegawa et al [12] 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) is obtained at the sender host as follows. Rtt and rto can be directly obtained from the sender TCP and 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 estimation error in p is due to the stochastic nature of packet losses since the analysis in Hasegawa et al [12] assumes random packet loss. Thus, we need a validation for the case where the packet losses occur at the drop-tail router, since in that case, packets tend to be dropped in a bursty nature. We present evaluation results on this aspect in Section 4.

Note that the appropriateness of the estimation method used in the E-ATBT algorithm has been validated [12]. However, we can apply another estimation result of TCP throughput given in Padhye et al [5] using the additional parameter of

W_{max}, the buffer size of the receiver.

$$\lambda = max(W_{max}/RTT, \; 1/(RTT\sqrt{\frac{2bp}{3}} + T_0\, min(1, 3\sqrt{\frac{3bp}{8}})p(1+32p^2))$$

where b is set to 2 if the receiver uses TCP's Delayed ACK option, and 1 if not.

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

$$B_i = \overline{\rho_i} \times rtt_i$$

where rtt_i is RTT of connection i. By this mechanism, a stable assignment of the send socket buffer to TCP connections is expected to be provided if the parameter set (p, rtt, and rto) used in the estimation is stable. However, in ATBT, the assignment is inherently unstable even when three parameters are stable, since the window size oscillates more significantly regardless of the stability of the parameters.

As in ATBT, our E-ATBT also adopts a max-min fairness policy for re-assigning the excess buffer. Differently to the ATBT algorithm, however, E-ATBT employs a *proportional* re-assignment policy as explained in the previous subsection. That is, when excess buffer is re-assigned to connections needing more buffer, the buffer is re-assigned proportionally to the required buffer size calculated from the analysis. Whereas ATBT re-assigns excess buffer equally, since it has no means to know the expected throughput of the connections.

3.2 Simple Memory-copy Reduction (SMR) Scheme

Another mechanism we implemented is the Simple Memory-copy Reduction (SMR) scheme. In TCP data transfer, two memory-copy operations are necessary at the sender host in TCP data transfer [14]. One is from the file system to the application buffer, and the other is from the application buffer to the socket buffer, as shown in Figure 1(a). The problem is that memory access is the largest obstacle in improving TCP data transfer [15], so reducing the number of memory copy operations in TCP data transfer is a key to improving the server's protocol processing speed. Reducing the number of memory accesses in TCP data transfer is not a new subject [1, 2, 3]. In Drushel and Peterson [1], the authors proposed a 'fbuf' (fast buffer) architecture, which shares the memory space between the system kernel and user processes to avoid redundant memory copies during data transfers. However, it is difficult to implement fbuf on actual systems, because its complicated architecture requires many modifications in the memory management mechanism of operating systems. In this paper, we propose a Simple Memory-copy Reduction (SMR) scheme, which is a new and simpler mechanism that can avoid memory copying from the file system to the application buffer.

A detailed mechanism at the sender host in TCP data transfer of the original BSD UNIX system is illustrated in Figure 1(a). When the user application transfers a file by using TCP, the file is first copied from the file system (on disk or memory) to the application buffer located in the user memory space by one memory copy operation. Then, the data in the application buffer is copied to the socket buffer which is located in the kernel memory space. It is performed by a couple of system calls provided by the socket interface, and so two memory copy operations are necessary. It is redundant, however, to copy the file in the file system to the socket buffer via the application buffer, as it can be directly copied from the file system to the socket buffer in the file transfer. Of course, the memory space should be clearly divided into user and kernel spaces by the UNIX memory management system, so that the kernel memory space is protected from illegal accesses caused by user applications. When we consider the performance improvement of TCP data transfer, we can avoid memory copying from the file system to the application buffer.

Figure 1(b) illustrates the proposed mechanism of the TCP data transfer. In the proposed mechanism, a transferring file in the file system is directly copied to the socket buffer by the new socket system calls. The socket system calls of the original mechanism require the application buffer as one of the arguments to copy the data from the application buffer to the socket buffer. However, in the proposed mechanism, socket system calls are modified to specify a file descriptor of a transferring file as an argument, by which a direct copy of the data from the file system to the socket buffer can be accomplished. Then, the redundant memory copying procedure can be avoided. In what follows, we explain the new socket system calls implemented in our proposed mechanism. See Figure 2 for the flow chart of the socket system call of the FreeBSD system.

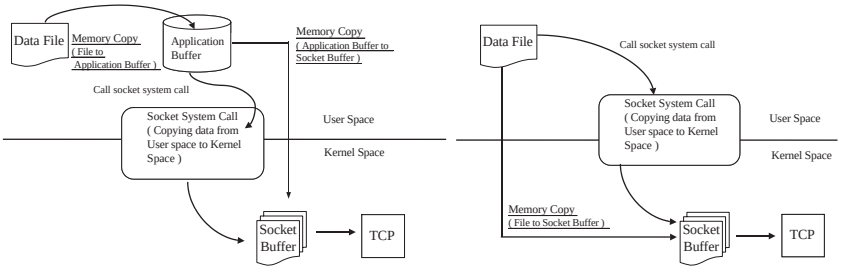
We implemented the proposed mechanism by modifying three system calls, and adding two new arguments to the *mbuf* structure [16]. All system calls for TCP data transfer call a `sosend` system call, in order to copy the transferring data from the user memory space to the kernel memory space and pass the data for further protocol processing. In the proposed mechanism, we change one of the arguments of `sosend` from the application buffer to the file descriptor, so that `sosend` can copy the data directly from the file system to the socket buffer. We also modify `sendto` and `sendit` system calls to handle the file descriptor according to the changes to the `sosend` system call.

3.3 Transfer of Small Documents

In E-ATBT, the sender estimates the throughput of each TCP connection from observed parameters associated with that connection. Therefore, if the size of the transmitted data is small, it is impossible to obtain an accurate and reliable estimation. A similar discussion is also applied to the SMR scheme, where the effect of avoiding a memory-copy operation is likely to be limited if the document size is small. One of the reasons is that the connection set-up time (TCP's three-way handshake) is likely to dominate the document transfer time.

The above problem becomes an obstacle for the application of our proposed mechanism to Web servers since Web documents are comparatively small. It is reported in Nabe et al [17] that the average size of Web documents at several Web servers was under 10 [KBytes]. More important, Crovella and Cestavos [18] report that Web document size exhibits a heavy-tailed nature, meaning that very long documents exist with certain probabilities, but at the same time small-sized documents exist with large probabilities. Thus, when there are many connections on the Web server requesting small documents, the buffer assignment of the E-ATBT mechanism may become unstable as the number of connections fluctuates greatly.

One possible way to overcome this problem is to exclude connections requesting small-sized documents from the fair buffer assignment described above. By this modification, a stable assignment of buffer is expected to be achieved. However, it leads another difficulty that the threshold value of



(a) Original Mechanism

(b) Proposed Mechanism of SMR scheme

Figure 1: Memory Copy Operations in TCP data Transfer

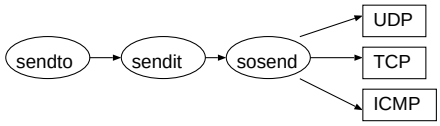


Figure 2: Flow Chart of Socket System Call

the document size cannot be determined *a priori* because the appropriate setting of the threshold is dependent on the access frequency of the Web server, processing power of the Web server, the network condition, and so on. Fortunately, the 'persistent connection' mechanism recently proposed and adopted in HTTP/1.1 can alleviate the above-mentioned problem. In HTTP/1.1, the server preserves the status of the TCP connection when it finishes the document transfer, and reuses the status when a new connection is established in the same session. Then, E-ATBT is able to utilize the status information for an effective buffer assignment even when most of connections request small documents.

4 Simulation Results of E-ATBT

In this section, we present simulation results of E-ATBT using a network simulator [19]. For evaluating the effectiveness of E-ATBT, we compared the following three algorithms.

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

ATBT: The send socket buffer size is assigned according to the automatic TCP buffer tuning algorithm described in Section 2.

E-ATBT: The sender host assigns the send socket buffer in accordance with the equation-based automatic TCP buffer tuning algorithm described in Subsection 3.1.

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

In the simulation experiment, we conducted two cases for packet loss occurrence. In the first case, packet loss took place at a constant rate, implicitly assuming that the router was equipped with the RED algorithm. It meant that packet loss took place randomly with a given probability. Such an assumption has been validated [5]. However, random packet dropping is a rather ideal case for our E-ATBT since our method largely relies on an adequate estimation of the packet loss rate in determining the throughput of TCP connections. Hence, we also considered the case of the drop-tail router where packet loss occurred due to buffer overflow at the router. In this case, we can never expect random packet losses, and they tend to occur in a bursty fashion. Due to space limitation, we only show the case of constant packet loss rate case. However, we have obtained the results that the estimation error of the packet loss rate make little effect on the performance of E-ATBT in drop-tail case.

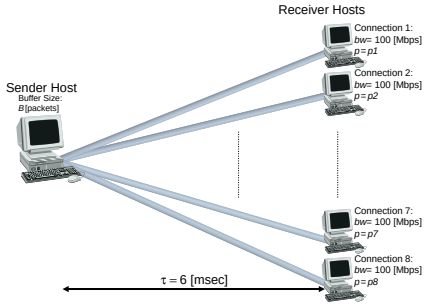
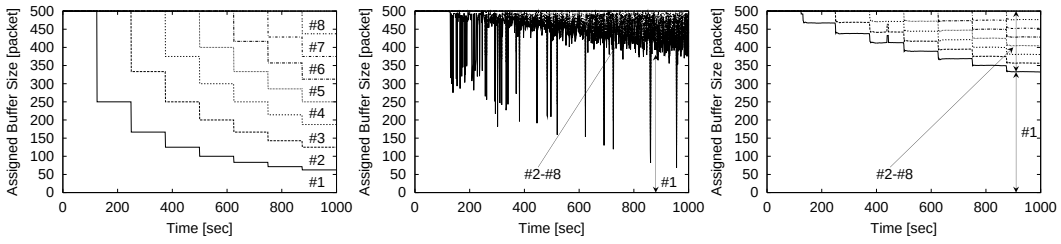


Figure 3: Network Model for Simulation Experiment 1

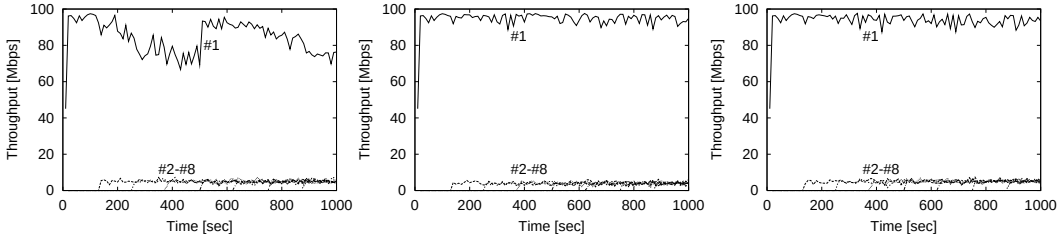
4.1 Simulation Experiment 1: Constant Packet Loss Rate Case

Here we investigated the case where each TCP connection experiences random packet losses with constant rate. The network model used in this simulation experiment is depicted in Figure 3. It consisted of a sender host, and eight receiver hosts (from receiver host 1 to 8), each of which was connected by a link of 100 [Mbps] to the sender host. The propagation delays between the sender host and receiver hosts (denoted by τ) were equally set to 6 [msec]. We set the packet loss probability on each link between the sender host and the receiver host i to p_i. The sender host had B [packets] of the socket buffer size in total. Each of simulation experiments started at time 0. TCP connection i, which was established from the sender host to the receiver host i, started packet transmission at time $t = (i-1) \times 125$ [sec]. All connections stopped packet transmission at $t = 1000$ [sec].

We first set the packet dropping probability p_i's as p_1 = 0.0001 and p_2 = ... = p_8 = 0.02. The socket buffer size of the sender B was set to be 500 [packets], and the packet size was fixed at 1000 [bytes]. Expected throughput values obtained by the receiver hosts then became about 95 [Mbps] for the receiver host i, and about 7 [Mbps] for the rest of receiver hosts. Figure 4 plots the time-dependent behaviors of the assigned buffer size and throughput values of connections obtained by three mechanisms; EQ, ATBT, and E-ATBT. In the figure, labels "#1" – "#8" represent connections 1 through 8. In the EQ mechanism (Figures 4(a) and 4(d)), the throughput of connection 1 was degraded during $200 < t < 500$ [sec] and $600 < t$ [sec] of the simulation time. The first degradation occurred because the assigned buffer size of connection 1 was suddenly decreased at the time when the second and third connections started packet transmission. It resulted in temporal degradation of the throughput of connection 1. At $t = 600$ [sec], the number of active connections exceeded

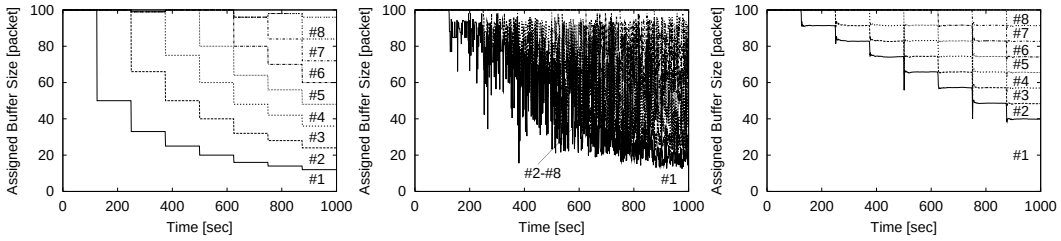


(a) EQ: Assigned Buffer Size (b) ATBT: Assigned Buffer Size (c) E-ATBT: Assigned Buffer Size

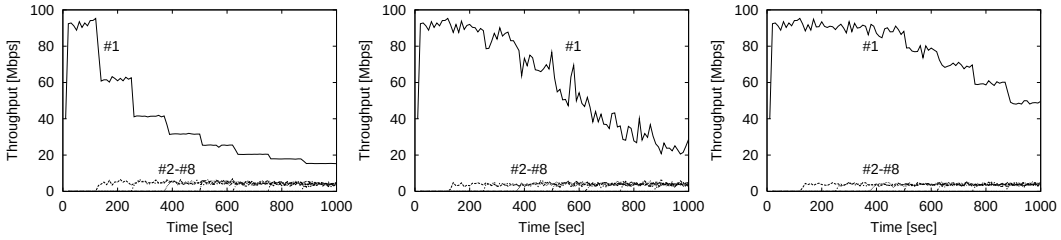


(d) EQ: Throughput (e) ATBT: Throughput (f) E-ATBT: Throughput

Figure 4: Result of Simulation Experiment 1: $p_1 = 0.0001$, $p_2 = \ldots = p_8 = 0.02$, $B = 500$ [packets]

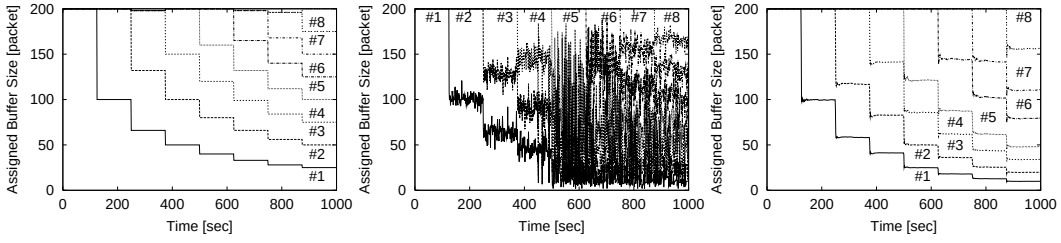


(a) EQ: Assigned Buffer Size (b) ATBT: Assigned Buffer Size (c) E-ATBT: Assigned Buffer Size

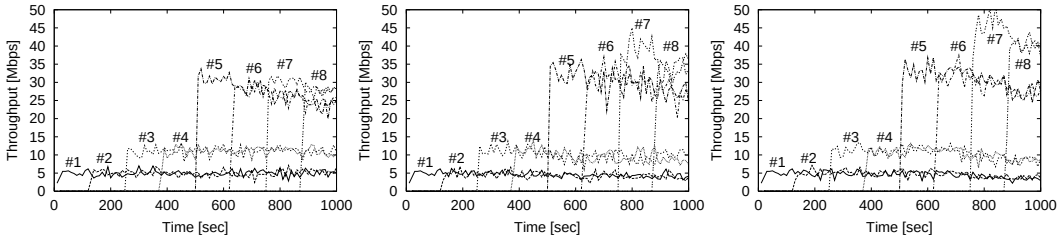


(d) EQ: Throughput (e) ATBT: Throughput (f) E-ATBT: Throughput

Figure 5: Result of Simulation Experiment 1: $p_1 = 0.0001$, $p_2 = \ldots = p_8 = 0.02$, $B = 100$ [packets]



(a) EQ: Assigned Buffer Size (b) ATBT: Assigned Buffer Size (c) E-ATBT: Assigned Buffer Size



(d) EQ: Throughput (e) ATBT: Throughput (f) E-ATBT: Throughput

Figure 6: Result of Simulation Experiment 1: $p_1 = p_2 = 0.02$, $p_3 = p_4 = 0.01$, $p_5 = p_6 = 0.002$, $p_7 = p_8 = 0.001$, $B = 200$ [packets]

six. However, connection 1 required about 110 [packets] to achieve its estimated throughput for $p_1 = 0.0001$. If the active number of connections exceeds six, its assigned buffer would be below 110 [packets] in the EQ mechanism. This was the reason the throughput of connection 1 was decreased. On the other hand, the other connections only required about 10 [packets] of the send socket buffer and therefore, those connections attained a constant throughput even though the assigned buffer size got smaller.

In both ATBT and E-ATBT cases, on the other hand, the throughput of connection 1 could retain a high value even when the number of connections became large as shown in Figures 4(e) and 4(f). This is because the send socket buffer was assigned appropriately, that is, the connection 1 was assigned a larger buffer size than other connections (see Figures 4(b) and 4(c)) even after other connections joined. When comparing ATBT and E-ATBT mechanisms, E-ATBT can offer stable buffer assignment as shown in Figure 4(b). However, it can also be observed by comparing Figures 4(e) and 4(f) that the obtained throughput values of the two cases are not different.

We next decreased the total buffer size, B, to 100 [packets]. See Figure 5. In all of three mechanisms, the throughput of connection 1 was degraded as the number of connections increased, but E-ATBT provided the highest throughput to connection 1. E-ATBT estimates the required buffer size for all connections, and then assigns a small buffer to connections 2 through 8 since those do not require a large buffer due to a high packet loss rate. Thus, the excess buffer can be utilized by connection 1 so that connection 1 can enjoy high throughput as shown in Figure 5(c). Further, the buffer assigned by E-ATBT was very stable compared with the ATBT case. In ATBT, connections 2 through 8 temporarily inflated their window size according to the congestion algorithm of TCP, which resulted in a decrease of the assigned buffer size of connection 1 (Figure 5(e)), and led to the throughput degradation of connection 1 in ATBT.

For another experiment using the network model depicted in Figure 3, we set p_i's as follows; $p_1 = p_2 = 0.02$, $p_3 = p_4 = 0.01$, $p_5 = p_6 = 0.002$, $p_7 = p_8 = 0.001$. That is, four kinds of connections were set up at the sender host. We set the buffer size B to 200 [Kbytes], which was relatively small compared with the total value of the required buffer sizes of the 8 connections. Results are shown in Figure 6. Connections 7 and 8 started the packet transmission at 750 [sec] and 875 [sec], respectively. In the EQ mechanism, those connections received the same throughput as connections 5 and 6 as shown in Figure 6(d), even though connections 7 and 8 had a lower packet loss rate than connections 5 and 6. In ATBT, throughputs of connections 7 and 8 were slightly higher than connections 5 and 6 (Figure 6(e)), but the buffer assignment had a large oscillation (Figure 6(b)). On the other hand, Figure 6(c) shows that E-ATBT kept a stable buffer assignment. It could also provide the highest throughput to connections 7 and 8 without throughput degradation of other connections (Figure 6(f)).

5 Implementation Experiments of SSBT

In this section, we present the results obtained from our experimental system. We implemented EQ, ATBT and SSBT mechanisms on an Intel Pentium PC running FreeBSD, with the two machines directly connected. In the experiments, we focused on the following four subjects;

1. Fair buffer assignment among different connections

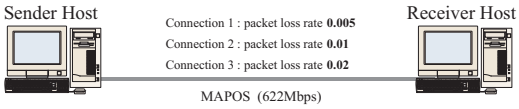


Figure 7: Network Environment (1)

2. Scalability against the number of connections
3. Performance improvement of the Web server

The experimental results concerning these four points are shown in turn in the following subsections.

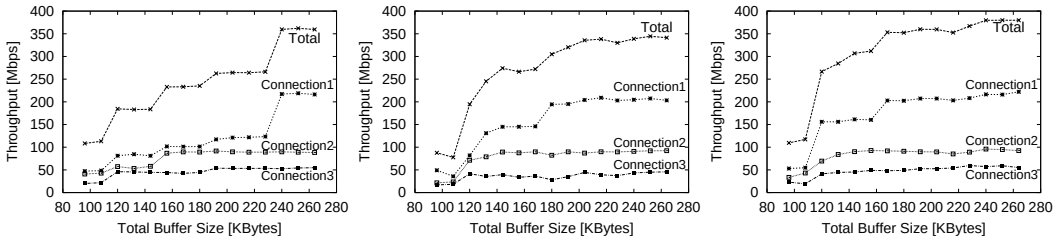
5.1 Experiment 1: Fair Buffer Assignment among Different Connections

We next evaluated the E-ATBT mechanism of SSBT. First, we investigated the fairness property of the three buffer assignment algorithms described in Section 4. In this experiment, we considered the situation where three TCP connections were established at the sender host through a 622 [Mbps] MAPOS link. Three connections have different packet loss rates; 0.005 for connection 1, 0.01 for connection 2, and 0.02 for connection 3, as shown in Figure 7. In the experiment, packets were intentionally dropped at the receiver host. Note that with those packet loss rates, three connections could achieve throughput values of about 55, 90, and 220 [Mbps], respectively, when enough of the send socket buffer was assigned to each connection.

Figure 8 compares throughput values of the three connections and total throughput. The horizontal axis shows the total size of the send socket buffer. In the EQ case (Figure 8(a)), the three connections could achieve their maximum throughputs when the total buffer size was larger than 240 [KBytes], while ATBT and E-ATBT needed only about 200 [KBytes] and 170 [KBytes], respectively. This due to the same reason as set out in Section 4. Connections 2 and 3 did not need as much buffer size as connection 1, but the EQ algorithm cannot re-allocate the excess buffer of connections 2 and 3 to connection 1. Contrarily, ATBT and E-ATBT algorithms worked well for the buffer assignment.

By comparing the ATBT and E-ATBT algorithms in Figures 8(b) and (c), it is clear that E-ATBT can provide a much higher throughput for connection 1. The difference due to the re-assignment policies of the excess buffer to the connections. In the ATBT algorithm, the excess buffer of connection 3 was equally re-assigned to connections 1 and 2. Then, only connection 3 was assigned a sufficient buffer size while connections 1 and 2 needed more buffer. On the other hand, E-ATBT re-assigned the excess buffer in proportion to the required buffer sizes for connections 1 and 2. Accordingly, E-ATBT gave more buffer to connection 1 than connection 2 because the required buffer size of connection 1 was larger than that of connection 2. Consequently, connection 1 could achieve higher throughput at the expense of a small throughput degradation of connection 2. Thus, the total throughput of E-ATBT was highest among the three algorithms regardless of the total buffer size, as shown in Figure 8.

We next present the time dependent behavior of the buffer size assigned to each connection in Figure 10 using the network topology depicted in Figure 9. For routers 1 and 2 in the figure, we used a PC-based router called ALTQ [20] to realize the RED router. We established connections 1, 2, and 4 between the server host and client host 1, and connection 3 between the server host and client host 2. In this experiment,



(a) EQ Case (b) ATBT Case (c) E-ATBT Case

Figure 8: Fairness among Three Connections

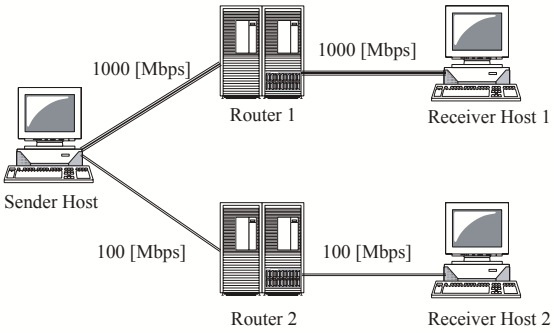


Figure 9: Network Environment (2)

connections 1 and 2 first started packet transmission at time 0 [sec]. Then, connections 3 and 4 joined the network at times of 5 [sec] and 10 [sec], respectively. Here, we set the total buffer size to 512 [KBytes]. In ATBT (Figure 10(a)), the assigned buffer sizes heavily oscillated because the ATBT algorithm adapted the buffer assignment according to the window size. Another and more important reason was that when retransmission timeout expiration occurred at a certain connection, that connection reset the window size to 1 [packet] according to the TCP retransmission mechanism. Then, the assigned buffer size of that connection became very low. Once the assigned buffer got small, the connection could not inflate its window size for a while because of the throttled buffer size. This was the reason why the assigned buffer size kept low for a while in the ATBT algorithm. On the other hand, E-ATBT can provide a stable and fair buffer assignment as shown in Figure 10(b). It brings higher throughput to each TCP connection as shown in Figure 11 in Subsection 5.2.

5.2 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 12 depicts the experimental setting for this purpose. One TCP connection was established using the MAPOS link (which is referred to below as the *MAPOS connection*). Several numbers of TCP connections were simultaneously established through the Ethernet link (*Ethernet connections*). In this experiment, we intended to investigate the effects of the number of Ethernet connections on the performance of the MAPOS connection.

Figure 11 shows the throughput values of the MAPOS connection dependent on the number of Ethernet connections. In

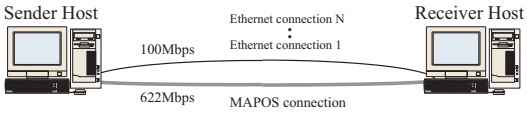
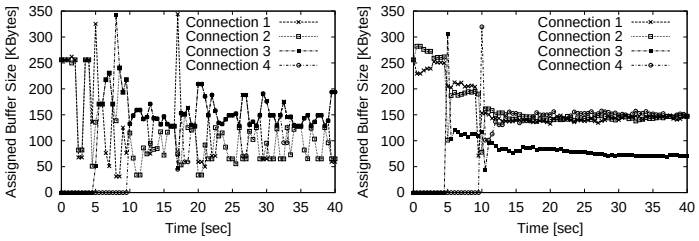


Figure 12: Network Environment (3)

addition to the results of EQ, ATBT and E-ATBT algorithms, we also present the results for the situation 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 the 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 would allow up to four connections at the same time.

From Figure 11 several important observation can be made. First, we can see that the constant assignment algorithm, which is widely employed in current Operating Systems, has the following drawback. If a 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 higher as shown in Figure 11. 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]. This is because the EQ algorithm does not distinguish the MAPOS connection from the Ethernet connections, and assigns equal sizes 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 ATBT and E-ATBT algorithms, 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 exceeds 11, the throughput values of ATBT and E-ATBT algorithms become distinguishable, as shown in Figure 11. This is again caused by the instability of the assigned buffer size and by the poor re-assignment algorithm of ATBT, as explained in the previous subsection. Even in the E-ATBT algorithm, the throughput of the MAPOS connection is slightly degraded by the larger number of Ethernet connections because the total required buffer size of Ethernet connections is increased, even though the required buffer of each Ethernet



(a) ATBT Case (b) E-ATBT Case

Figure 10: Changes of Assigned Buffer Sizes

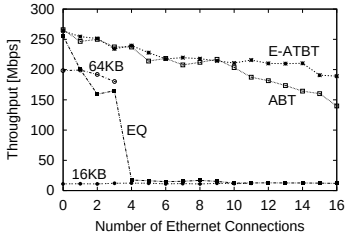


Figure 11: Throughput of the MAPOS Connection vs. the Number of Ethernet Connections

connection is small. Thus, 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 E-ATBT algorithm.

5.3 Experiment 3: Web Server Performance Evaluation

In this experiment, the effectiveness of the SSBT scheme is investigated using a machine running an Apache Web server [21]. We used the network topology in Figure 9, and ran the SSBT-enabled Web server at the sender host. At the receiver host, we ran httperf [22] to generate document transfer requests to the Web server by HTTP. Furthermore, to to make it a realistic scenario for traffic arriving at the server, we followed Barfort and Crovella [23] in determining the characteristics of the Web access, which included document size distribution, idle time distribution of the requests, and distribution for the number of embedded documents. We used HTTP/1.1 for Web access to the server. The send/receive socket buffer size of each TCP connection was set to 64 [KBytes]. We ran each experiment for 30 minutes, which corresponded to the generation of about 25,000 document transfer requests from each client.

Figure 13 shows the average performance gain of the SSBT scheme as a function of the document size. Here, the average performance gain for the documents with size i [Bytes] was determined as follows. Letting the number of document requests during the experiments be $M_{\mathrm{SSBT,i}}$ and $M_{orig,i}$ for cases with and without the SSBT scheme, respectively. Similarly, the observed transfer time of the jth document with size i was denoted as $T_{\mathrm{SSBT,i,j}}$ and $T_{orig,i,j}$, respectively. Then, the average performance gain G_i for documents with size i was defined as;

$$G_i\ [\%] = 100 \times \left(\frac{\sum_{j=1}^{M_{orig,i}} \frac{i}{T_{orig,i,j}}}{M_{orig,i}} \right) \Bigg/ \left(\frac{\sum_{j=1}^{M_{\mathrm{SSBT,i}}} \frac{i}{T_{\mathrm{SSBT,i,j}}}}{M_{\mathrm{SSBT,i}}} \right)$$

Note that since we generated the requests according to the probability functions for the document sizes, idle time, and the number of embedded documents, $M_{orig,i}$ and $M_{\mathrm{SSBT,i}}$ may be different in the experiments with or without SSBT. If $G(i)$ was over 100 [%], it meant that our SSBT outperformed the original mechanism.

The cases of 10, 100, and 600 clients are shown in Figures 13(a), 13(b) and 13(c), respectively. It appears that SSBT degraded the performance of the WWW server, especially when the document size was small. This is caused by TCP's retransmission timeouts, which were explainable as follows. When a requested document size was small, the window size of the TCP connection carrying the document did not become so large. Therefore, if packet loss occurred, the TCP connection tended to wait retransmission timeout since the window size was too small to invoke a fast retransmit algorithm. It resulted in the timeout duration occupying the largest part of the document transfer. That is, the transfer time of the small document was dominated by whether the timeout occurred or not, not by whether the SSBT algorithm was used or not.

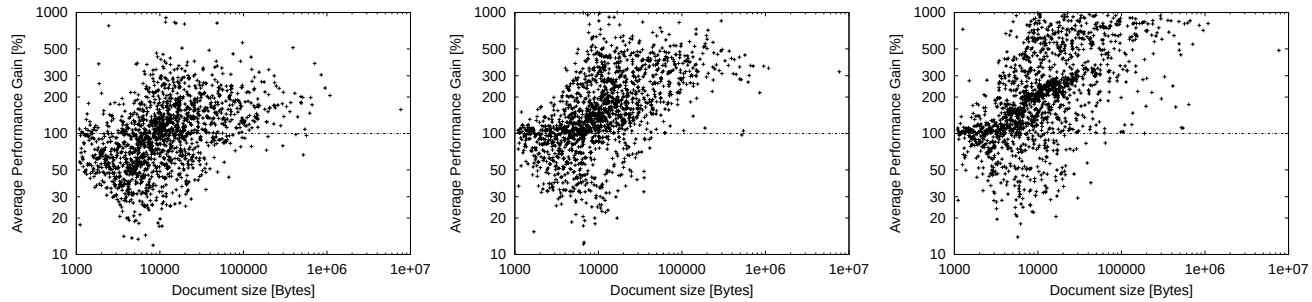
On the other hand, when the document size was large, the effectiveness of the SSBT became apparent as was expected. Especially when the number of clients increased, it greatly reduced the document transfer times (see Figure 13(c)). It meant that the SSBT algorithm shared the Web server resource in an effective and fair manner, compared with the original mechanism. These results show that the SSBT algorithm is effective at actual Internet servers

6 Conclusion

In this paper, we have proposed SSBT (Scalable Socket Buffer Tuning), a novel architecture for effectively and fairly utilizing the send socket buffer of a busy Internet server. SSBT consists of two algorithms, the SMR and E-ATBT schemes. The SMR scheme can reduce the number of memory-copy operations when the data packet is sent by TCP, and improve the overall performance of the server host. The E-ATBT algorithm assigns send socket buffers to the connections according to the estimated throughput of those connections. We have confirmed the effectiveness of the SSBT algorithm through both of simulation and implementation experiments, and have shown that SSBT can improve the overall performance of a server, as well as providing a fair assignment of the send socket buffer to the heterogeneous TCP connections.

Acknowledgements

This work was partly supported by the Research for the Future Program of the Japan Society for the Promotion of Science under the Project "Integrated Network Architecture for Advanced Multimedia Application Systems," a Grant-in-Aid for Encouragement of Young Scientists (A) 13750349 from The Ministry of Education, Culture, Sports and Science and Technology of Japan, and by the "Research on High-performance WWW server for the Next-Generation Internet" program of from the Telecommunications Advancement Foundation.



(a) 10 Clients (b) 100 Clients (c) 600 Clients

Figure 13: Document Size vs. Average Performance Gain by the SSBT Scheme

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, Dec. 1993.

[2] J. Chase, A. Gallatin, and K. Yocum, "End-system optimizations for high-speed TCP," *appear in IEEE Communications, special issue on high-speed TCP*, June 2000.

[3] A. Gallatin, J. Chase, and K. Yocum, "Trapeze/IP: TCP/IP at near-gigabit speeds," in *Proceedings of 1999 USENIX Technical Conference*, June 1999.

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

[5] 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, Aug. 1998.

[6] S. Floyd and K. Fall, "Promoting the use of end-to-end congestion control in the Internet," *IEEE/ACM Transactions on Networking*, vol. 6, Aug. 1999.

[7] J. Martin, A. Nilsson, and I. Rhee, "The incremental deployability of RTT-based congestion avoidance for high speed TCP Internet connections," in *Proceedings of ACM SIGMETRICS 2000*, pp. 134–144, June 2000.

[8] G. Hasegawa, K. Kurata, and M. Murata, "Analysis and improvement of fairness between TCP Reno and Vegas for deployment of TCP Vegas to the Internet," in *Proceedings of IEEE ICNP 2000*, Nov. 2000.

[9] D. Lin and R. Morris, "Dynamics of random early detection," *ACM Computer Communication Review*, vol. 27, pp. 127–137, Oct. 1997.

[10] M. Shreedhar and G. Varghese, "Efficient fair queuing using deficit round robin," *IEEE/ACM Transactions on Networking*, vol. 4, pp. 375–385, June 1996.

[11] T. J. Ott, T. V. Lakshman, and L. Wong, "SRED: Stabilized RED," in *Proceedings of IEEE INFOCOM'99*, Mar. 1999.

[12] G. Hasegawa, T. Matsuo, M. Murata, and H. Miyahara, "Comparisons of packet scheduling algorithms for fair service among connections on the Internet," in *Proceedings of IEEE INFOCOM 2000*, Mar. 2000.

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

[14] X. Xiao, L. Ni, and W. Tang, "Benchmarking and analysis of the user-perceived performance of TCP/UDP over Myrinet," *Tech. Rep., Michigan State Univ.*, Dec. 1997.

[15] D. D. Clark, V. Jacobson, J. Romkey, and H. Salwen, "An analysis of TCP processing overhead," *IEEE Communications Magazine*, vol. 27, pp. 23–29, June 1989.

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

[17] 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, Dec. 1999.

[18] M. E. Crovella and A. Bestavros, "Self-similarity in World Wide Web traffic: Evidence and possible causes," *IEEE/ACM Transactions on Networking*, vol. 6, pp. 835–846, Dec. 1997.

[19] "Network simulator - `ns` (version 2).." available from `http://www.isi.edu/nsnam/ns/`.

[20] ALTQ (Alternate Queueing for BSD UNIX) , *available from* `http://www.csl.sony.co.jp/~kjc/software.html`.

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

[22] D. Mosberger and T. Jin, "httperf – a tool for measuring Web server performance," *Technical Report, Hewlett-Packard Laboratories, HPL-98-61*, Mar. 1998.

[23] P. Barford and M. Crovella, "Generating representative Web workloads for network and server performance evaluation," in *Proceedings of ACM SIGMETRICS '98*, 1998.