Scalable Socket Buffer Tuning for High-Performance Web Servers

Paper No. 122
Total Number of Pages: 23

Abstract

Although many research efforts have been devoted to the network congestion against an increase of the Internet traffic, only a few discussions are recently made on the performance improvement of the endhosts. In this paper, we propose a new architecture, which is called Scalable Socket Buffer Tuning (SSBT), to provide high performance and fair service for many TCP connections at the Internet endhosts. SSBT has two major features. The one is to reduce the number of memory accesses at the sender host by using some of new system calls, which is called Simple Memory-copy Reduction (SMR) scheme. The other is the 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 the send socket buffer to them according to the estimated throughput. If the socket buffer is short, the max-min fairness policy is used.

We confirm an effectiveness of our proposed algorithm through both of a simulation technique and an experimental system. From the experimental results, we have found that our SSBT can achieve up to 30% gain of the Web server throughput, and the 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 a rapid growth of the Internet users, many research efforts have been made in order to avoid and dissolve the network congestion against an increase of network traffic. However, only a few discussions are recently made on the performance improvement of the 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 in the current Internet receive hundreds of requests for document transfer every second at peak time.

Of course, an improvement of protocol processing on the endhosts is not a new subject. An early 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. Other approaches can be found in, e.g., [2, 3]. However, past researches including the above approaches do not consider the 'fair' treatment of connections, by which connections receive fair service from the server. By fair service, we can also expect performance improvement by the following reasons. Suppose that a server host is sending TCP data to two clients of 64Kbps dial-up (say, client A) and 100Mbps LAN (client B). If the server host assigns an equal size of the send socket buffer to both clients, it is likely that an 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 two connections. For an effective buffer usage for 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 accepts a large number of TCP connections with different bandwidths and round trip times (RTTs) at the same time. In [4], the authors have proposed a buffer tuning algorithm called Automatic TCP Buffer Tuning (referred to as *ATBT* in this paper), which dynamically adjusts the send socket buffer size according to the change of the TCP sending window size of the connection. However, it cannot provide 'fairness' among connections because the throughput of TCP connections is not proportional to the sending window size [5]. We will provide more discussions on the problems of ATBT 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 develop the following two mechanisms in SSBT. Those are 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 simple mechanism, compared with the other so-called 'zero-copy' mechanisms such as the fbuf architecture, but it can achieve the same effect in reducing the overhead at the sender host. Our main contribution in this paper is E-ATBT, which provides a fair assignment of the socket buffer. In E-ATBT, we use '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 based on its expected throughput, with consideration on a max-min fairness among connections.

We validate an effectiveness of our proposed mechanism through both of simulation and implementation experiments. In the simulation experiments, we confirm the fair assignment of the socket buffer by using our E-ATBT algorithm under various network situations. In the implementation experiments, we confirm the behavior of the SSBT at the transport-layer, that is, we use the native TCP data transfer in the experiments. We show the effectiveness of SSBT in terms of overall server performance, the number of accommodated connections, and fairness among connections.

We last note that the fairness is recently paid much attention in the various aspects. See, e.g., [6, 7, 8] for TCP enhancements and [9, 10, 11] for router support in order to achieve the fairness. Perhaps, all of the above technologies including our proposed method for endhosts are necessary to provide the *end–to–end* fairness to the users.

This paper is organized as follows. In Section 2, we first introduce the ATBT algorithm briefly for reference purpose. 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)

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

In [4], the authors have proposed an Automatic TCP Buffer Tuning (ATBT) mechanism, 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. 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, 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. If some connections do not require large buffer, the excess buffer is re-assigned to connections requiring larger buffer. Through this mechanism, it is expected to provide dynamic and fair buffer assignment by considering differences of TCP connections.

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 decreases its window size suddenly due to, e.g., an occasional packet loss, the assigned buffer size sometimes gets smaller than that the connection actually requires. It 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 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 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 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 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 method called a "Scalable Socket Buffer Tuning" (SSBT) includes the following two mechanisms; the Equation-based Automatic TCP Buffer Tuning (E-ATBT) and the Simple Memory-copy Reduction (SMR) scheme. We explain them in the following subsections.

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

The Equation-based Automatic TCP Buffer Tuning (E-ATBT) solves the problems raised in Section 2. In E-ATBT, the sender host estimates an 'expected' throughput of each TCP connection by monitoring three parameters (packet loss probability, round trip time, and retransmit timeout 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 [12]. In [12], the authors have derived the average throughput of the TCP connection for the model in which multiple connections with different input link bandwidths share the bottleneck router employing the RED algorithm [13]. The following parameters are used 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 TCP's retransmission timeout. The analysis developed in [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. 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 a the stochastic nature of the packet losses since the analysis in [12] 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. We will present evaluation results on this aspect in Section 4.

Note that an appropriateness of the estimation method used in the E-ATBT algorithm has been validated in [12]. However, we can apply another estimation result of TCP throughput given in [5] using an additional parameter of W_{max}, the buffer size of the receiver.

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

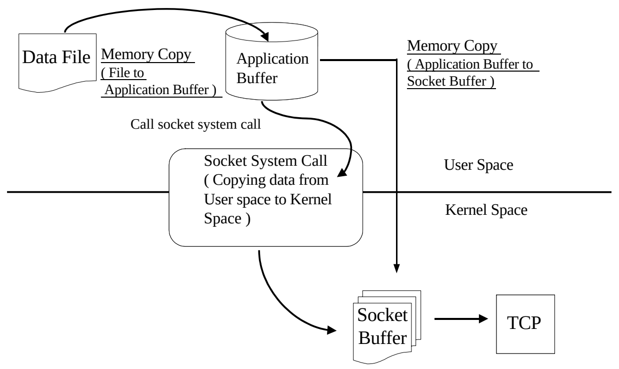
where b is set to 2 if the receiver uses TCP's Delayed ACK option, and 1 if not.

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;

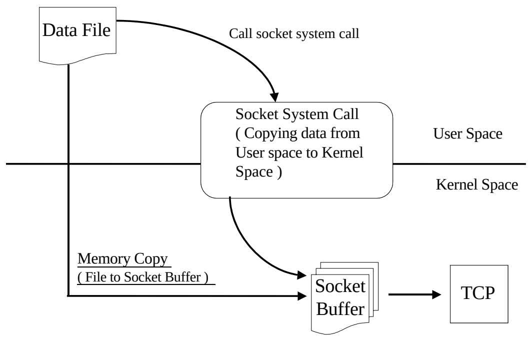
$$B_i = \overline{\rho_i} \times rtt_i$$

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

As in ATBT, our E-ATBT also adopts the max-min fairness policy for re-assigning the excess buffer. Differently from the ATBT algorithm, however, E-ATBT 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


Data File
Memory Copy (File to Application Buffer)
Application Buffer
Memory Copy (Application Buffer to Socket Buffer)
Call socket system call
Socket System Call (Copying data from User space to Kernel Space)
User Space
Kernel Space
Socket Buffer
TCP

(a) Original Mechanism


Data File
Call socket system call
Socket System Call (Copying data from User space to Kernel Space)
User Space
Kernel Space
Memory Copy (File to Socket Buffer)
Socket Buffer
TCP

(b) Proposed Mechanism of SMR scheme

Figure 1: Memory Copy Operations in TCP data Transfer

buffer, the buffer is re-assigned to connections in proportion to the required buffer size calculated from the analysis. Note that ATBT re-assigns the 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 implement is a Simple Memory-copy Reduction (SMR) scheme for further performance improvement. 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 the memory access is the largest obstacle in improving TCP data transfer [15], and 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 issue. Such an example can be found in [1, 2, 3]. In [1], the authors proposed the '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. It is a new and simpler mechanism that can avoid the memory copy 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. Two memory copy operations are thus necessary. It is redundant, however, to copy the file in the file system to the socket buffer via the application buffer, and it can be directly copied from the file system to the socket buffer in file transfer. Of course, the memory space should be clearly divided into the user and kernel spaces by the UNIX memory management system, so that the kernel memory space is protected from illegal accesses by user applications. When we consider performance improvement of TCP data transfer, however, we can avoid the memory copy 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 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. In the proposed mechanism, on the contrary, the 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 copy procedure can be avoided. In what follows, we explain 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 ker-

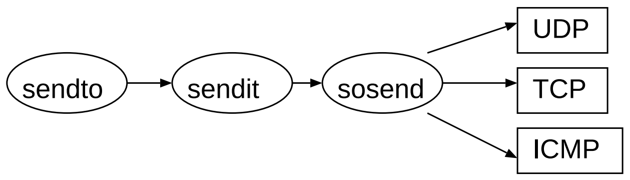


Figure 2: Flow Chart of Socket System Call

nel 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. Furthermore, we modify `sendto` and `sendit` system calls to handle the file descriptor according to the changes of 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 transmitted data is small, it is impossible to obtain the accurate and reliable estimation. The 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 transferring 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 our proposed mechanism to be applied to Web servers since Web documents are comparatively small. It is reported in [17] that the average size of Web documents at several Web servers is under 10 [KBytes]. More importantly, the authors in [18] report that the Web document size exhibits the heavy-tailed nature. It means that there exist very long documents with certain probabilities, but at the same time small-sized documents exist with large probability. Especially, if there are many connections on the Web server which is requesting small documents, the buffer assignment of the E-ATBT mechanism may become unstable since the number of connections fluctuates largely.

One possible way to overcome this problem is to exclude connections requesting the small-sized documents from the fair buffer assignment described above. By this modification, the stable assignment of the buffer is expected to be achieved. However, it leads another difficulty that the threshold value of the document size cannot be determined a priori because the appropriate setting of the threshold is dependent on 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 re-uses the status when a new connection is established on the same session. Then, E-ATBT is able to utilize the status information for the effective buffer assignment even when most of the connections request small documents.

4 Simulation Results of E-ATBT

In this section, we present simulation results of E-ATBT using the network simulator `ns` [19]. For the SMR scheme, we performed implementation experiments, which will be reported in Section 5. For evaluating an effectiveness of E-ATBT, we compare the following three algorithms in order to validate the effectiveness of our proposed mechanism.

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 three algorithms, the 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 takes place with a constant rate implicitly assuming that the router is equipped with the RED algorithm. It means that the packet loss takes place randomly with given probability. Such an assumption is validated by [5]. However, random packet dropping is a rather ideal case to 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 consider the case of the drop-tail router where packet loss occurs 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.

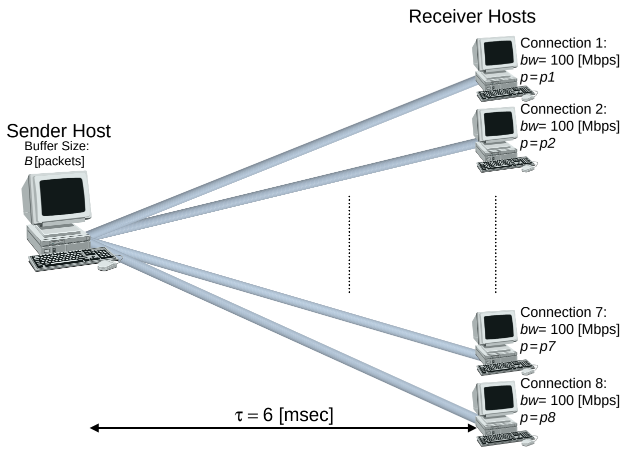


Figure 3: Network Model for Simulation Experiment 1

4.1 Simulation Experiment 1: Constant Packet Loss Rate Case

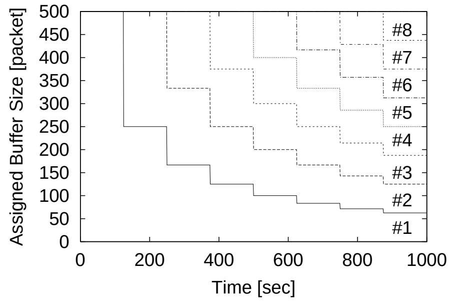
We first investigate the case where each TCP connection experiences random packet losses with constant rate. The network model used in this simulation experiment is depicted in Figure 3. It consists of a sender host, and eight receiver hosts (from receiver host 1 to receiver host 8), which are connected by the link of 100 [Mbps] to the sender host. The propagation delays between the sender host and receiver hosts (denoted by τ) are 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 has B [packets] of the socket buffer size in total. Each of simulation experiments starts at time 0. TCP connection i, which is established from the sender host to the receiver host i, starts packet transmission at time $t = (i-1) \times 125$ [sec]. All connections stop 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 is set to be 500 [packets]. Expected throughput values obtained by the receiver hosts then become 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 value of connections obtained by three mechanisms; EQ, ATBT, and E-ATBT. In the figure, labels "#1" – "#8" represent for connections 1 through 8. In the EQ mechanism (Figures 4(a) and 4(d)), the throughput of connection 1 is degraded during $200 < t < 500$ [sec] and $600 < t$ [sec] of the simulation time. The first degradation occurs because the assigned buffer size of connection 1 is suddenly decreased at the time when the second and third connections starts packet transmission. It results in temporal degradation of the throughput of connection 1. At $t = 600$ [sec], the number of active connections exceeds six. However, connection 1 requires about 110 [packets] to achieve its estimated throughput for $p_1 = 0.0001$. If the active number of connections exceeds six, its assigned buffer is below 110 [packets] in the EQ mechanism. It is the reason why the throughput of connection 1 is decreased. On the other hand, the other connections only require about 10 [packets] of the send socket buffer and therefore, those connections attain the constant throughput even if the assigned buffer size gets smaller.

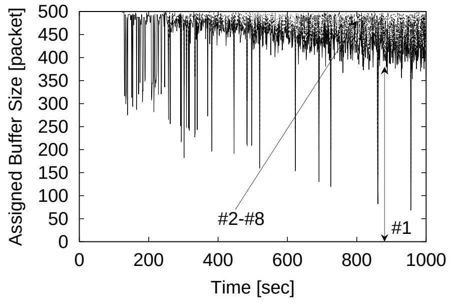
In both of ATBT and E-ATBT cases, on the other hand, the throughput of connection 1 can keep a high value even when the number of connections becomes large as shown in Figures 4(e) and 4(f). It is because the send socket buffer is assigned appropriately, that is, the connection 1 is assigned the larger buffer size than other connections (see Figures 4(b) and 4(c)) even after other connections join. 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 two cases are not different.

We next decrease the total buffer size, B, to 100 [packets]. See Figure 5. In all of three mechanisms, the throughput of connection 1 is degraded as the number of connections increases, but E-ATBT provides the highest throughput to connection 1. E-ATBT estimates the required buffer size for all connections, and assigns the small buffer to connections 2 through 8 since those do not require the large buffer due to its high packet loss rate. Then, 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 assigned buffer by E-ATBT is very stable when comparing with the ATBT case. In ATBT, connections 2 through 8 temporarily inflate their window size according to the congestion algorithm of TCP, and it results in the decrease of the assigned buffer size of connection 1 (Figure 5(e)). It leads to throughput degradation of connection 1 in ATBT.

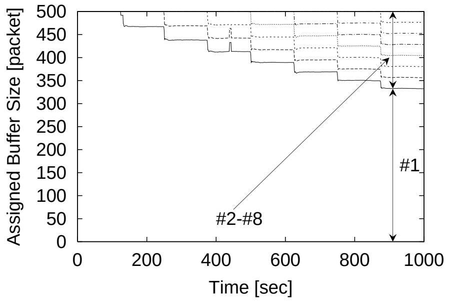
For another experiment using the network model depicted in Figure 3, we set p_i's as follows; p_1 = p_2 = 0.02, p_3 = p_4 = 0.01, p_5 = p_6 = 0.002, p_7 = p_8 = 0.001. That is, four kinds of connections exist at the sender host. We set the buffer size B to 200 [Kbytes], which is relatively small compared with the total value of required buffer sizes of 8 connections. Results are shown in Figure 6. Connections 7 and 8 start the packet transmis-



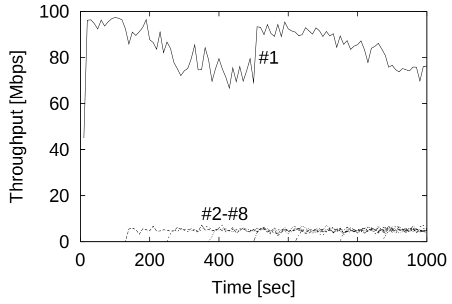
(a) EQ: Assigned Buffer Size



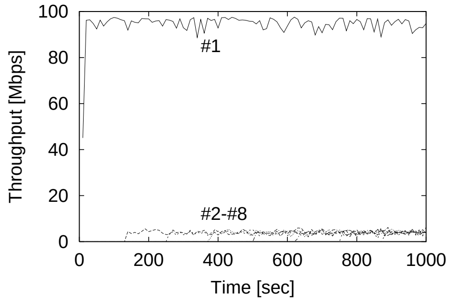
(b) ATBT: Assigned Buffer Size



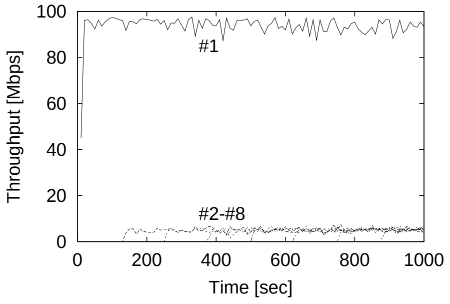
(c) E-ATBT: Assigned Buffer Size



(d) EQ: Throughput

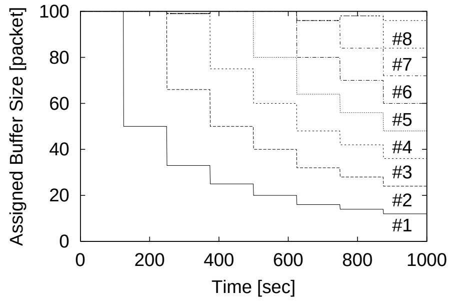


(e) ATBT: Throughput

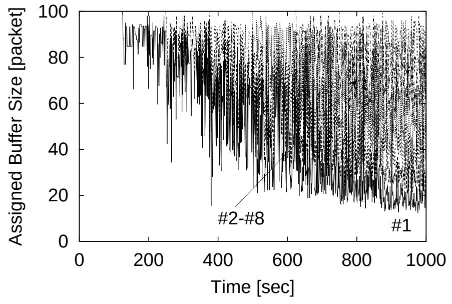


(f) E-ATBT: Throughput

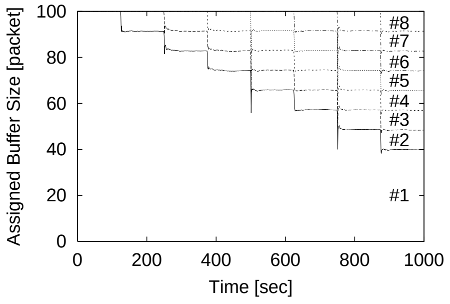
Figure 4: Result of Simulation Experiment 1: $p_1 = 0.0001$, $p_2 = \ldots = p_8 = 0.02$, $B = 500$ [packets]



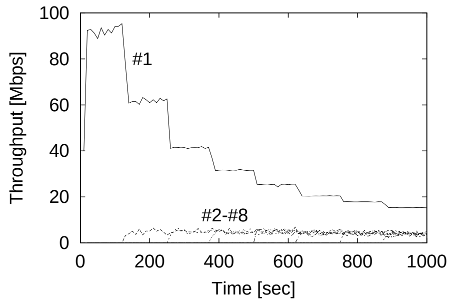
(a) EQ: Assigned Buffer Size



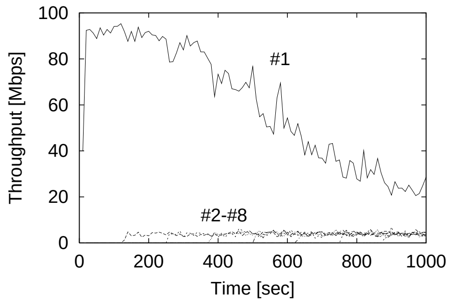
(b) ATBT: Assigned Buffer Size



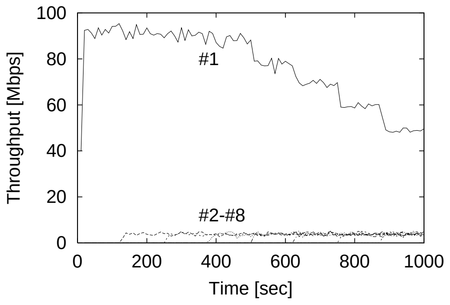
(c) E-ATBT: Assigned Buffer Size



(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]

sion at 750 [sec] and 875 [sec], respectively. In the EQ mechanism, those connections receive as same through-

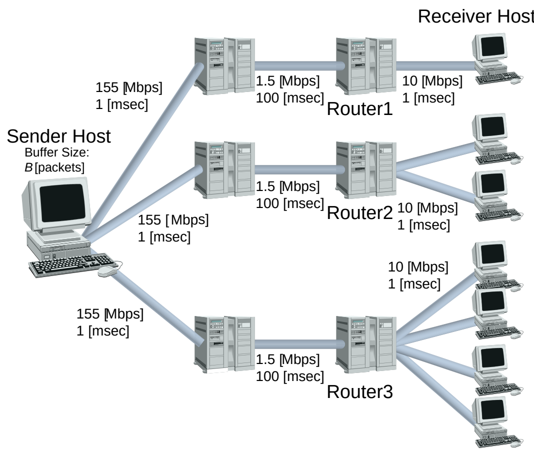


Figure 7: Network Model for Simulation Experiment 2

put as connections 5 and 6 as shown in Figure 6(d), even though connections 7 and 8 have lower packet loss rate than connections 5 and 6. In ATBT, throughputs of connections 7 and 8 are slightly higher than connections 5 and 6 (Figure 6(e)), but the buffer assignment has large oscillation (Figure 6(b)). On the other hand, Figure 6(c) shows that E-ATBT keeps stable buffer assignment. It can also provide the highest throughput to connections 7 and 8 without throughput degradation of other connections (Figure 6(f)).

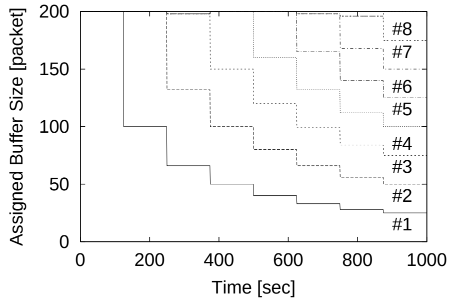
4.2 Simulation Experiment 2: Drop-tail Case

We next use the network model depicted in Figure 7 to examine the case where the packet loss occurs by packet buffer overflow at the network router. Differently from the constant packet loss rate case in the previous subsection, the packet loss rate of each TCP connection oscillates, which may affect the behavior of the E-ATBT mechanism.

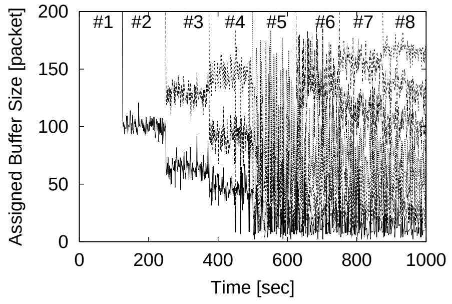
The simulation model consists of a sender host, seven receiver hosts, six routers, and links connecting routers and sender/receiver hosts. Seven TCP connections are established at the sender host (connections 1 through 7). As shown in the figure, connection 1 occupies router 1, connection 2 and 3 share 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 the routers and the receiver hosts are set to be 155 [Mbps] and 10 [Mbps], respectively. Further, propagation delays of links between the routers, and that between the routers and the 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]. 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. In this case, packet loss tends to occur in a bursty fashion due to buffer overflow at drop–tail routers. In the E-ATBT algorithm, the packet loss rates of the connections are monitored and be used for throughput estimation.

Figure 8 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 8(a) and 8(d), connections sharing router 3 (connections 4 through 7) have unfair throughput values in the EQ case, in spite of the fact that conditions of four connections are completely identical. It is due to an accidental unfairness of the TCP congestion control algorithm, which is well known in the literature. See, e.g., [20, 21]. Figures 8(b) and 8(e) clearly show that ATBT cannot provide a stable buffer assignment. It is because ATBT 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 not requiring large buffer (connection 2 through 7 in the current case) are often inflated. It leads to the temporary decrease of the buffer assigned to connections that need large buffer (connection 1), as in the constant packet loss rate case of the previous subsection. As a result, the throughput of connection 1 is often degraded. On the other hand, our proposed E-ATBT keeps a stable and fair buffer assignment as shown in Figure 8(c), leading to fair treatment in terms of throughput (Figure 8(f)).

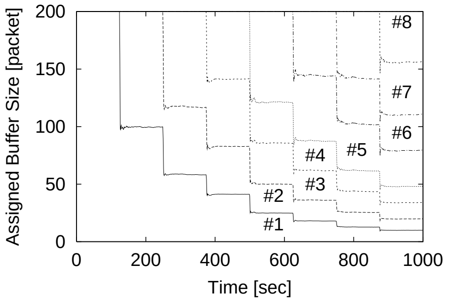
Even when the buffer size is smaller, E-ATBT keeps its effectiveness. Results are shown in Figure 9, where the buffer size B is decreased to 200 [KBytes]. Figures 9(d) and 9(e) show that EQ and ATBT mechanisms cannot provide fairness among connections. In particular, throughput of connection 1 is significantly degraded, and connections going through router 3 receive different throughput values. It is because in assigning the buffer, the EQ method does not take care of the connections' characteristics at all. See Figure 9(a). It is also true for the ATBT mechanism (Figure 9(b)). It is mainly due to the oscillation of the assigned buffer to each connection as shown in Figures 9(b). From the figure, it is also verified that ATBT, which equally re-assigns the excess buffer to connections, does not work well. On the other hand, in the E-ATBT algorithm, throughput of connec-



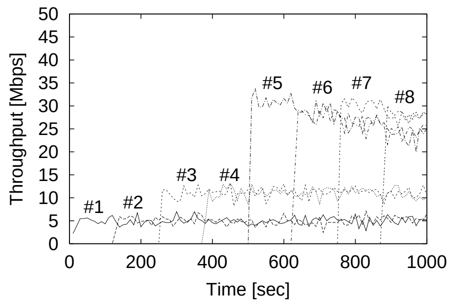
(a) EQ: Assigned Buffer Size



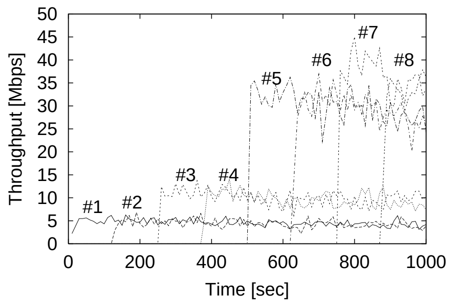
(b) ATBT: Assigned Buffer Size



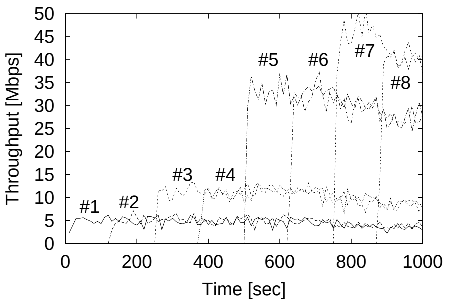
(c) E-ATBT: Assigned Buffer Size



(d) EQ: Throughput

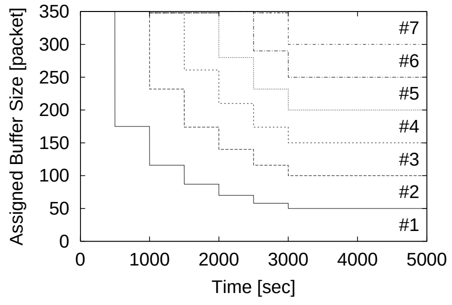


(e) ATBT: Throughput

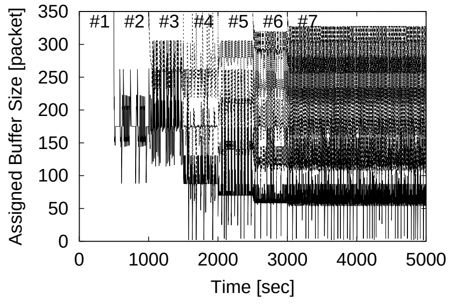


(f) E-ATBT: Throughput

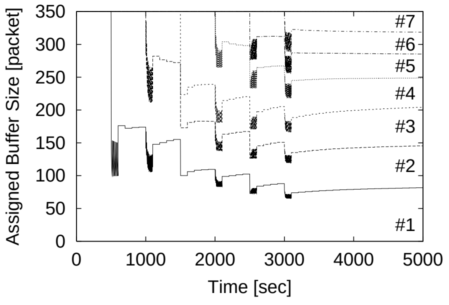
Figure 6: Result of Simulation Experiment 1: $p_1 = p_2 = 0.02$, $p_3 = p_4 = 0.01$, $p_5 = p_6 = 0.002$, $p_7 = p_8 = 0.001$, $B = 200$ [packets]



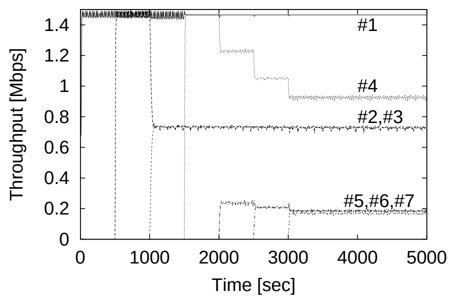
(a) EQ: Assigned Buffer Size



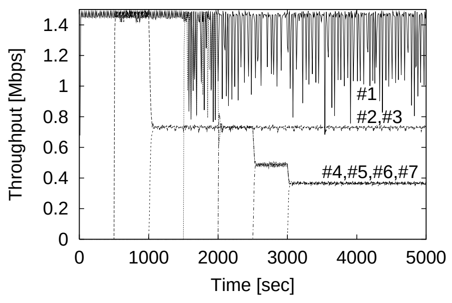
(b) ATBT: Assigned Buffer Size



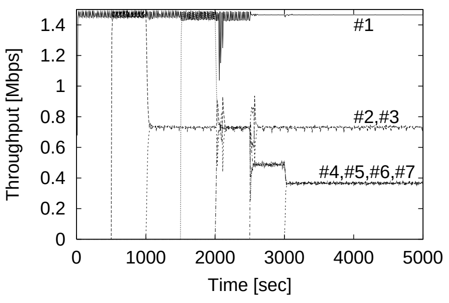
(c) E-ATBT: Assigned Buffer Size



(d) EQ: Throughput

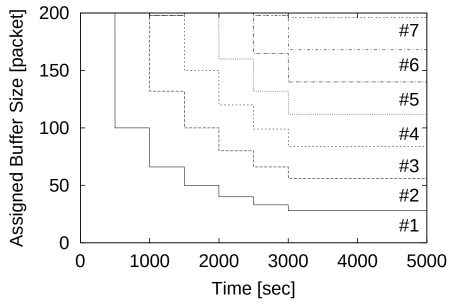


(e) ATBT: Throughput

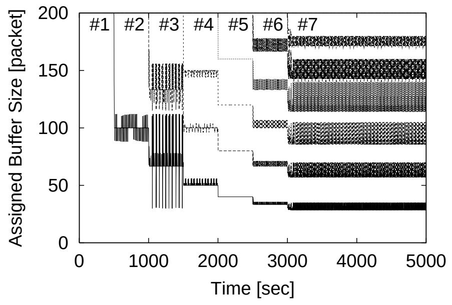


(f) E-ATBT: Throughput

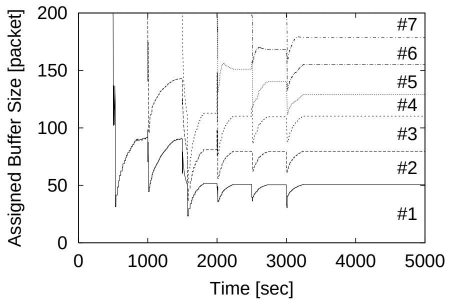
Figure 8: Comparative Results of Simulation Experiment 2: $B = 350$ [packets]



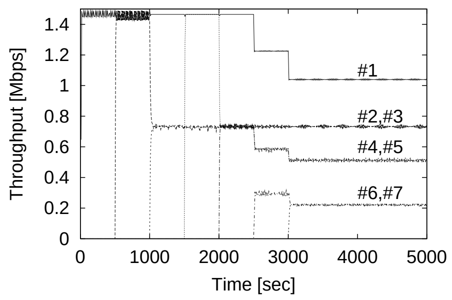
(a) EQ: Assigned Buffer Size



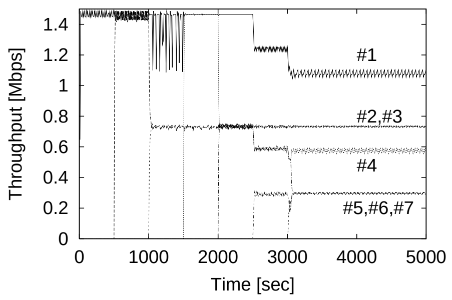
(b) ATBT: Assigned Buffer Size



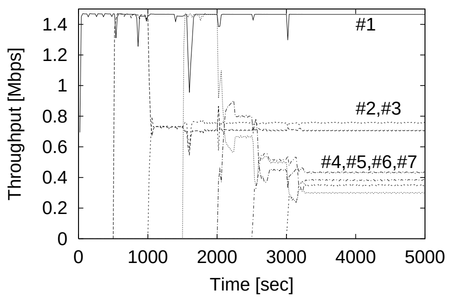
(c) E-ATBT: Assigned Buffer Size



(d) EQ: Throughput



(e) ATBT: Throughput



(f) E-ATBT: Throughput

Figure 9: Comparative Results of Simulation Experiment 2: $B = 200$ [packets]

tion 1 is kept to be a high value, and much better fairness can be achieved (Figures 9(c) and 9(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 change 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.

5 Implementation Experiments of SSBT

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

1. Server performance improvement by the SMR scheme
2. Fair buffer assignment among different connections
3. Scalability against the number of connections
4. Web server performance evaluation

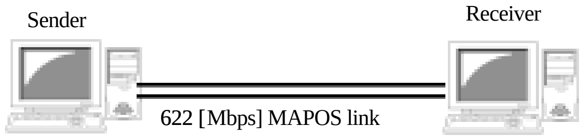


Figure 10: Network Environment (1)

The experimental results are shown in the following subsections in turn.

5.1 Experiment 1: Server Performance Improvement by SMR

We implemented the SMR scheme explained in Subsection 3.2 on Pentium-III Xeon 550MHz machine with 512MB memory, running FreeBSD 4.0-RELEASE. The two machines are directly connected by the the 622 [Mbps] MAPOS (Multiple Access Protocol Over SONET/SDH) [22] link, as shown in Figure 10. We did not change the TCP/IP implementation, except that the delayed ACK option [16] is not used. The sender host transmits 500 [MB] data to the receiver host with a single TCP connection through the MAPOS link. We then calculate the mean TCP throughput by measuring the transfer time of the data.

Figure 11 shows the mean TCP throughput values

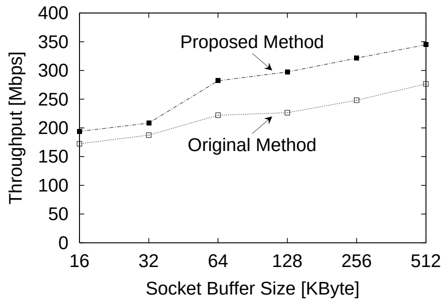


Figure 11: TCP Throughput using MAPOS Link

of the original and proposed mechanisms, as a function of the socket buffer size of the sender host. The socket buffer size of the receiver host is identically set to that of the sender socket buffer size. The result shows that the proposed mechanism can improve up to about 30% of the throughput compared with the original mechanism. In this case, the bottleneck for performance improvement is located at endhosts since 622 [Mbps] bandwidth of the MAPOS link is large enough. That is, the sender host cannot inject data packets to fully utilize the link bandwidth in the original system, because its processing performance in TCP data transfer is slow compared with the network bandwidth. However, our SMR scheme can utilize the bandwidth of the MAPOS link largely compared with the original systems, and therefore the effect of the proposed mechanism becomes significant.

In the following experiments, we use the server machine which is equipped with SMR scheme.

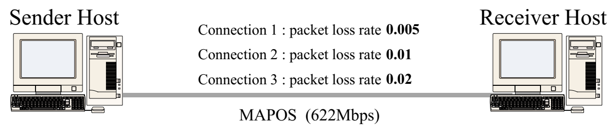
5.2 Experiment 2: Fair Buffer Assignment among Different Connections

We next evaluate the E-ATBT mechanism of SSBT. First, we investigate the fairness property of three buffer assignment algorithms described in Section 4. In this experiment, we consider the situation where three TCP connections are established at the sender host through 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 12. 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, when an enough amount of the send socket buffer is assigned to each connection.

Figure 13 compares 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 13(a)), the three connections can achieve their maximum throughputs when the total buffer size is larger than 240 [KBytes], while ATBT and E-ATBT need only about 200 [KBytes] and 170 [KBytes], respectively. It is due to the same reason explained in Section 4. 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, ATBT and E-ATBT algorithms work well for the buffer assignment.

By comparing ATBT and E-ATBT algorithms in Figures 13(b) and (c), it is clear that E-ATBT can provide much higher throughput for connection 1. The difference is due to re-assignment policies of excess buffer to connections. In the ATBT algorithm, the excess buffer of connection 3 is equally re-assigned to connections 1 and 2. Then, only connection 3 is assigned a sufficient size of the buffer while connections 1 and 2 need more buffer. On the other hand, E-ATBT re-assigns the excess buffer in proportion to the required buffer size of connections 1 and 2. Accordingly, E-ATBT gives more buffer to connection 1 than connection 2 because the required buffer size of connection 1 is larger than that of connection 2. Consequently, connection 1 can achieve higher throughput at an expense of small throughput degradation of connection 2. Then, the total throughput of E-ATBT is highest among three algorithms regardless of the total buffer size as shown in Figure 13.

We next present a time–dependent behavior of the



Sender Host
Receiver Host
Connection 1 : packet loss rate 0.005
Connection 2 : packet loss rate 0.01
Connection 3 : packet loss rate 0.02
MAPOS (622Mbps)

Figure 12: Network Environment (2)

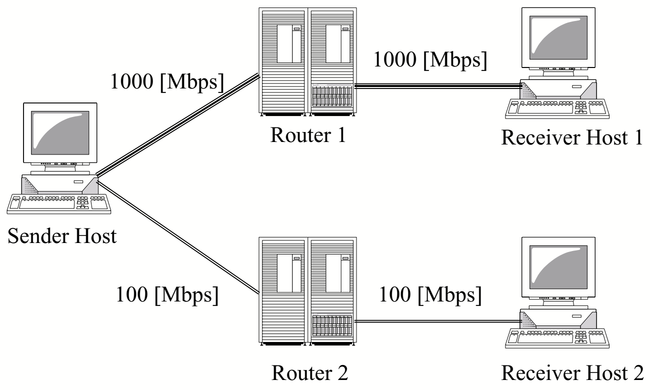
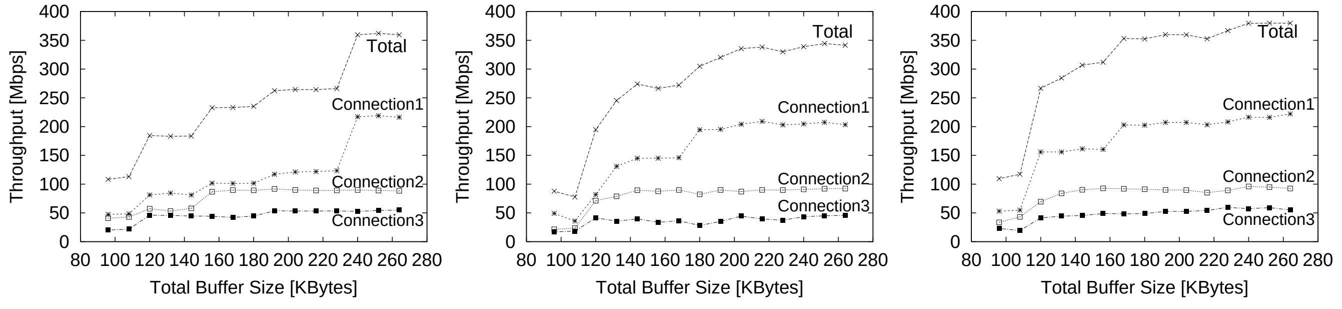


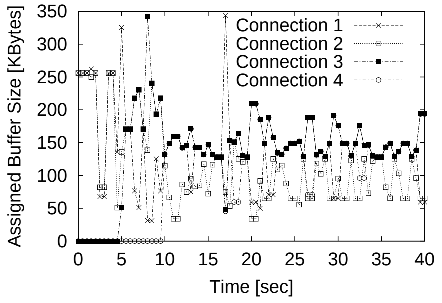
Figure 14: Network Environment (3)



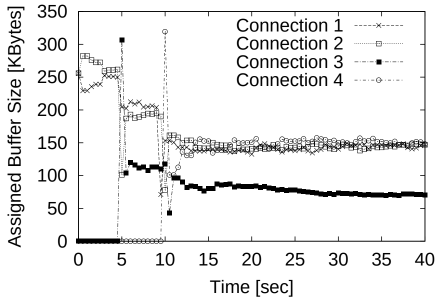
(a) EQ Case (b) ATBT Case (c) E-ATBT Case

Figure 13: Fairness among Three Connections

buffer size assigned to each connection in Figure 15. We use the network topology depicted in Figure 14. In the two routers (Router 1 and 2) in the figure, we use the PC-based router called ALTQ [23] to realize the RED router. We establish Connections 1, 2, and 4 between the server host and the client host 1, and connection 3 between the server host and the client host 2. In this experiment, connection 1 and 2 first start packet transmission at time 0 [sec]. Then, connection 3 and 4 joins the network in time 5 [sec] and 10 [sec], respectively. Here, we set the total buffer size to be 512 [KBytes]. In ATBT (Figure 15(a)), the assigned buffer sizes are heavily oscillated because the ATBT algorithm adapts the buffer assignment according to the window size. Another 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 during a while in the ATBT algorithm. On the other hand, E-ATBT can provide the stable and fair buffer assignment as shown in Figure 15(b). It brings higher throughput to each TCP connection as having been shown in Figure 17.



(a) ATBT Case



(b) E-ATBT Case

Figure 15: Changes of Assigned Buffer Sizes

5.3 Experiment 3: 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 16 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*). Through this experiment, we intend to investigate the effect of the number of Ethernet connections on the performance of the MAPOS connection.

Figure 17 shows the throughput values of the MAPOS connection dependent on the number of Ethernet connections. In addition to the results of EQ, ATBT and E-ATBT 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 connec-

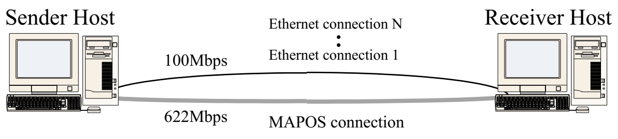


Figure 16: Network Environment (4)

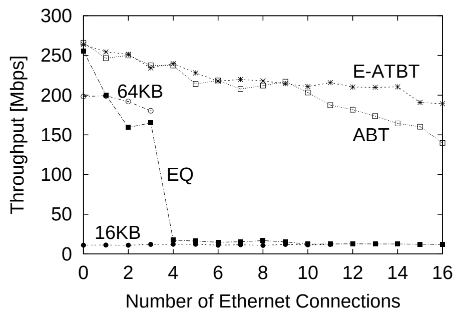


Figure 17: Throughput of MAPOS Connection vs. the Number of Ethernet Connections

tion. 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.

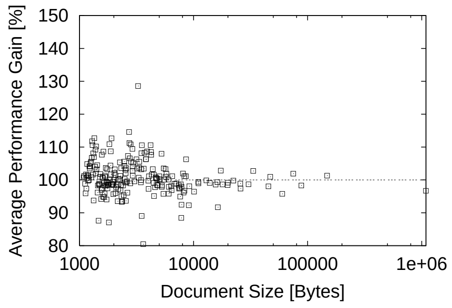
We can make several important observations from this figure. First, we can see that the constant assignment algorithm, which is widely employed in the current OS, has the following drawback. If 16 [KBytes] send socket buffer is assigned to each TCP connection, the MAPOS connection suffers from very low 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 17. 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 the 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 ATBT or E-ATBT 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 more than 11, throughput values of ATBT and E-ATBT algorithms becomes distinguishable as shown in Figure 17. It is again caused by the instability of the assigned buffer size and by the poor re-assignment algorithm of ATBT, as having been 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. It is because the total of the required buffer size of Ethernet connections is increased, even though the required buffer of the 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 by the E-ATBT algorithm.

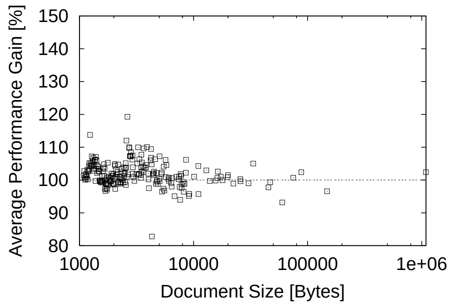
5.4 Experiment 4: Web Server Performance Evaluation

For the last experiment, the effectiveness of the SMR scheme is presented using the machine running the Apache Web server [24]. We use the same network topology as shown in Figure 10, and run the SMR-enabled Web server on the sender host. At the receiver host, we use httperf [25] to generate document transfer requests to the Web server by HTTP, by which we consider more realistic scenario for the traffic arriving at the server. For this purpose, we follow [26] in determining the characteristics of the Web access, which includes the document size distribution, the idle time distribution of requests, and the distribution for the number of embedded documents. For comparison purpose, we use both of HTTP/1.0 and HTTP/1.1 for Web access to the server. The send/receive socket buffer size of each TCP connection is set to 64 [KBytes]. We run each experiment during 30 minutes, which corresponds to about 25,000 document transfer requests from each client.

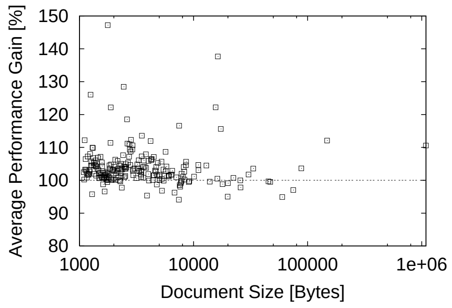
Figure 18 shows the average performance gain by the SMR scheme as a function of the document size. Here, the average performance gain for the documents with size i [Bytes] was determined as follows. Let the number of document requests during the experiments be $M_{smr,i}$ and $M_{orig,i}$ for the case with and without the SMR scheme. Similarly, the observed transfer time of jth document with size i is denoted as $T_{smr,i,j}$ and $T_{orig,i,j}$.



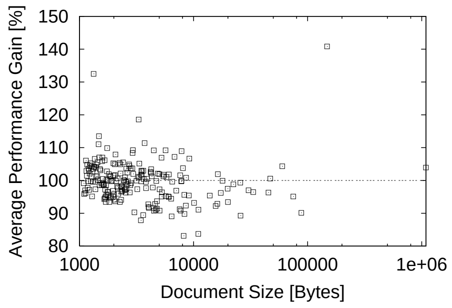
(a) HTTP/1.1, 10 Clients



(b) HTTP/1.1, 60 Clients



(c) HTTP/1.1, 600 Clients



(d) HTTP/1.0, 600 Clients

Figure 18: Document Size vs. Average Performance Gain by SMR Scheme

Then, the average performance gain G_i for documents with size i is defined as;

$$G_i = \left(\frac{\sum_{j=1}^{M_{orig,i}} \frac{i}{T_{orig,i,j}}}{M_{orig,i}} \right) \Bigg/ \left(\frac{\sum_{j=1}^{M_{smr,i}} \frac{i}{T_{smr,i,j}}}{M_{smr,i}} \right) \quad (2)$$

Note that since we generated the requests according to the probability functions for the document size, idle time, and the number of embedded documents, $M_{orig,i}$ and $M_{smr,i}$ may be different.

The cases of 10, 60, and 600 clients with HTTP/1.1, and the case of 600 clients with HTTP/1.0 are shown in Figures 18(a), 18(b), 18(c), and 18(d), respectively. Recall that in Subsection 5.1, we have found that the SMR scheme can improve up to 30% of the server performance (Figure 11). However, it can be observed in the figures that the performance gain by the SMR scheme is limited when the number of concurrent clients is small. In the case of ten clients (Figure 18(a)), there are some points where the performance of the SMR scheme is even lower than the original scheme. It is due to the small sizes of the documents, as having been explained in Subsection 3.3. However, as the number of clients is increased, the performance gain becomes stable and significant. Compare Figures 18(a) through 18(c). As the number of concurrent clients increases, the server load gets larger, and the effect of the memory copy reduction becomes significant. When we compare HTTP/1.0 and HTTP/1.1 (Figures 18(c) and 18(d)), the effect of the SMR scheme in HTTP/1.1 is larger than that in HTTP/1.0 as expected. The SMR scheme does not help improve the performance in the HTTP/1.0 case. It is because TCP's three-way handshake is necessary in every document transfer in HTTP/1.0, and it occupies the most part of the total transfer time, as we have explained in Subsection 3.3.

We are now investigating the effect of our E-ATBT on the Web server using the same traffic model as above. As partly observed in the experimental results of the current subsection, HTTP/1.1 alleviates an ill–effect of the small-sized documents, which is one main problem of E-ATBT as described in Subsection 3.3. We can therefore expect that E-ATBT exhibits good performance even in a realistic traffic scenario.

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 the busy Internet server. SSBT consists of the 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 the send socket buffer to the connections according to the estimated throughput of 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 the server, and provide the fair assignment of the send socket buffer to the heterogeneous TCP connections.

In Subsection 5.4, we have only shown the effectiveness of the SMR scheme under the realistic traffic scenario based on the statistical model proposed in [26]. We are now conducting the experiment to investigate the effectiveness of the E-ATBT algorithm using the same traffic model.

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," in *Proceedings of ACM SIGCOMM '97*, 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-mash.cs.berkeley.edu/ns/`*.

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

[21] A. Veres and M. Boda, "The chaotic nature of TCP congestion control," in *Proceedings of IEEE INFOCOM 2000*, Mar. 2000.

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

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

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

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

[26] P. Barford and M. Crovella, “Generating representative Web workloads for network and server performance evaluation,” in *Proceedings of ACM SIGMETRICS ’98*, 1998.