A Resource/Connection Management Scheme for HTTP Proxy Servers

Takuya Okamoto[1], Tatsuhiko Terai[1], Go Hasegawa[2], and Masayuki Murata[2]

[1] Graduate School of Engineering Science, Osaka University
1-3 Machikaneyama, Toyonaka, Osaka 560-8531, Japan
{tak-okmt, terai}@ics.es.osaka-u.ac.jp

[2] Cybermedia Center, Osaka University
1-30 Machikaneyama, Toyonaka, Osaka 560-0043, Japan
{hasegawa, murata}@cmc.osaka-u.ac.jp

Abstract. Although many research efforts have been devoted to the network congestion against an increase of the Internet traffic, there has been a little concern on improvement of the performance of Internet hosts in spite of the projection that the bottleneck is now being shifted from the network to hosts. We have proposed SSBT (Scalable Socket Buffer Tuning), which is intended to improve the performance of Web servers by maintaining their resources effectively and fairly, and validated its effectiveness through the simulation and implementation experiments. In the current Internet, however, a significant amount of Web document transfer requests are through HTTP proxy servers. Accordingly, in this paper, we propose a new resource management scheme for proxy servers to improve their performance and to reduce Web document transfer time via the proxy servers. Our proposed scheme has the following two components. One is an enhanced E-ATBT, which is an enhancement version of our previous SSBT for proxy servers by taking account of different characteristics among TCP connections. The other is a scheme that manages persistent TCP connections at proxy servers to avoid newly arriving TCP connections from being rejected due to lack of resources. We validate an effectiveness of our proposed scheme through simulation experiments, and confirm that it can manage proxy server resources effectively.

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 a little concern on improvement of the performance of Internet hosts in spite of the projection that the performance bottleneck is now being shifted from the network to endhosts.

In [1], we have proposed SSBT (Scalable Socket Buffer Tuning) which is intended to improve the performance of Web servers by maintaining their resources effectively and fairly. SSBT has two major components; E-ATBT (Equation-based Automatic TCP Buffer Tuning) and SMR (Simple Memory-copy Reduction) schemes. In E-ATBT, we maintain an 'expected' throughput value of each active TCP connection, which is determined by an analytic estimation [2]. It is characterized by packet loss ratio, RTT (Round

E. Gregori et al. (Eds.): NETWORKING 2002, LNCS 2345, pp. 252–263, 2002.

Trip Time), and RTO (Retransmission Time Out), which are easily monitored by a sender host. Then, the send socket buffer is assigned to each connection based on its expected throughput with consideration on a max-min fairness among connections. 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. The SMR scheme is alike as other schemes [3,4], but it is simpler to implement.

In the current Internet, there are many requests for Web documents transfer via HTTP proxy servers [5]. Since the proxy servers are usually prepared by ISPs (Internet Service Providers) for their customers, such proxy servers must accommodate a large number of the customers' HTTP accesses simultaneously. Furthermore, the proxy servers should handle both of upward TCP connections (from the proxy server to Web servers) and downward TCP connections (from the client hosts to the proxy server). Therefore, it is likely that the proxy server becomes the bottleneck in Web document transfer, even when both of the network bandwidth and the Web server performance are large enough. That is, to reduce the Web document transfer time, a performance enhancement of the proxy servers should next be considered.

In this paper, we first point out several problems in handling TCP connections at the HTTP proxy server. The one is the assignment of the socket buffer for TCP connections at the proxy server. When a TCP connection is not assigned the proper size of send/receive socket buffer according to its throughput, the assigned socket buffer may be left unused or insufficient, which results in waste of the socket buffer. Another problem is the management of persistent TCP connections, which tends to waste the resource of the busy proxy server. When a proxy server accommodates many persistent TCP connections without any effective management, its resources are kept assigned to those connections whether those connections are actually 'active' or not. Then new TCP connections cannot be established since the server resources are short.

We propose a new resource management scheme for proxy servers to resolve such problems, and then to reduce Web document transfer time via the proxy servers. Our proposed scheme has following two features. One is an enhanced E-ATBT, which is an enhancement version of our previous E-ATBT for proxy servers. Differently from the Web servers, the proxy server should handle both upward and downward TCP connections and behave as a client host to obtain Web documents and in-line images from Web servers. We therefore enhance E-ATBT to effectively handle a dependency between upward and downward TCP connections and to assign its receive socket buffer size dynamically. The other is a resource management scheme that can avoid newly arriving TCP connections from being rejected due to lack of resources for establishing them on the proxy server. It involves the management of persistent TCP connections provided by the HTTP/1.1. The persistent connection can omit the overhead of TCP's three-way handshake and then reduce the document transfer time by HTTP. However, when the persistent TCP connection is unused until it is closed by timeout mechanism, the resources assigned for the TCP connection are wasted. The proposed scheme intentionally tries to close the persistent connections when the resources of the proxy server are shorthanded.

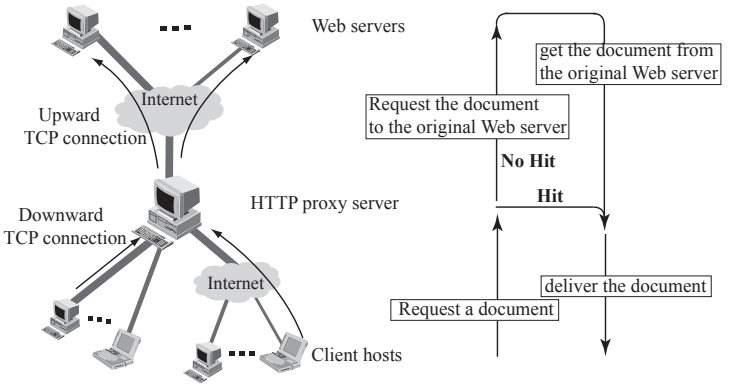


Fig. 1. HTTP Proxy Server

2 Background

2.1 Proxy Server

An HTTP proxy server works as an agent for Web client hosts that request Web documents. When it receives Web document transfer requests from the Web client host, it obtains the requested document from the original Web servers on behalf of the client host and delivers it to the client. It also caches obtained Web documents. When other client hosts request the same documents, it transfers the cached documents, which results in that the document transfer time is much reduced. For example, it is reported in [6] that using Web proxy servers reduces document transfer time by up to 30%. Also, when the cache is hit, the document transfer is performed without any connection establishment to Web servers. Thus, the congestion within the network and at Web servers can also be reduced.

The proxy server accommodates a large number of connections, which are connected from Web client hosts and to Web servers as depicted in Figure 1. It is a different point from Web servers. The proxy server behaves as a sender host for the downward TCP connection (between the client host and the proxy server) and as a receiver host for the upward TCP connection (between the proxy server and the Web server). Therefore, if the resource management is not appropriately configured at the proxy server, the document transferring time increases even when the network is not congested or the load at the Web server is not high. That is, careful and effective resource management is a critical issue for improving the performance of the proxy server. In the current Internet, however, most proxy servers including those in [7,8] are lack of such considerations.

Resources of HTTP proxy servers that we focus in this paper are *mbuf*, *file descriptor*, *control blocks*, and *socket buffer*. Those are closely related to the performance of TCP connections in transferring Web documents. Mbuf, file descriptor, and control blocks are resources for TCP connections. The amount of those resources cannot be changed dynamically according to the requirement of the proxy server, since it is determined when the system kernel is booted or when the proxy server is activated [9]. When at least one of the resources lacks, therefore, newly arriving TCP connections for Web document

transfer have to wait for other connections to be closed and their assigned resources to be released.

The socket buffer is used for data transfer operations between user applications and the sender/receiver TCP. When the user application transmits data using TCP, the data is copied to the send socket buffer and subsequently it is copied to the mbufs (or mbuf clusters). The size of the assigned socket buffer is a key issue for the effective data transfer by TCP. Suppose that a server host is sending TCP data to two client hosts; one a 64 Kbps dial-up (say, client A) and the other a 100 Mbps LAN (client B). If the server host assigns equal size of send socket buffers to both client hosts, 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 differences of capacity (more strictly, bandwidth-delay products) of their connections. For an effective buffer allocation to both client hosts, a compromise of the buffer usage should be taken into account.

We proposed an E-ATBT scheme [1], which assigns the receive socket buffer to each TCP connection dynamically according to its throughput estimated from the observed network parameters, such as packet loss ratio, RTT, and RTO. That is, a sender host calculates the average window size of its TCP connection based on the analysis result in [10] from the above three parameters. The throughput of the TCP connection is then obtained by considering the performance degradation caused by TCP's retransmission timeout. Finally, we estimate the required receive socket buffer size as multiplication of the estimated throughput and RTT of the TCP connection. By taking into account the observed network parameters, the resource at the Web server is appropriately allocated to connections in various network environments.

E-ATBT is applicable to HTTP proxy servers, since the proxy servers also accommodate many TCP connections issued by clients in various environments. However, since proxy servers have a *dependency* between upward and downward TCP connections, a straightforward application of E-ATBT is insufficient. Furthermore, the proxy server behaves as a receiver host for the upward TCP connection to the Web server, we have to consider the management scheme for the receive socket buffer, which was not considered in the original E-ATBT.

2.2 Persistent TCP Connection of HTTP/1.1

In recent years, many Web servers and client hosts (namely, Web browsers) support a *persistent connection* option, which is one of the important functions of HTTP/1.1 [11]. In the older version of HTTP (HTTP/1.0), the TCP connection between server and client hosts is immediately closed when the document transfer is completed. However, since Web documents have many in-line images, it is necessary to establish TCP connections many times to download them in HTTP/1.0. It results in a significant increase of document transfer time since the average size of Web documents at several Web servers is about 10 [KBytes] [12,13]. The three-way handshake in each TCP connection establishment makes the situation worse.

In the persistent connection of HTTP/1.1, on the other hand, the server preserves the status of the TCP connection, which includes the congestion window size, RTT, RTO, ssthresh, and so on, when it finishes the document transfer, and re-uses the connection and its status when other documents are transferred by using the same HTTP session. Then, the three-way handshake can be omitted. However, since it keeps the TCP connection

established whether the connection is active (in use for packet transfer) or not, the resources at the server are wasted when the TCP connection is inactive. Therefore, the significant portion of the resources may be wasted in order to keep the persistent TCP connections at the proxy server accommodating many TCP connections.

One solution against this problem is simply to discard HTTP/1.1 and to use HTTP/1.0, since HTTP/1.0 closes the TCP connection when the document transfer is finished. However, HTTP/1.1 has other elegant mechanisms such as the pipelining and the contents negotiation [11]. We should therefore develop an effective resource management scheme under HTTP/1.1. Our solution is that the proxy server aggressively closes the persistent TCP connections that are unnecessarily wasting the proxy resources, as the resources become short.

3 Algorithm and Implementation Issues

In this section, we propose a new resource management scheme suitable to the HTTP proxy server, which consists of a new management scheme of send/receive socket buffer, and a handling algorithm of persistent TCP connections.

3.1 New Socket Buffer Management Method

Handling the Relation of Upward and Downward Connections

A HTTP proxy server relays a document transfer request to a Web server for a Web client host. Thus, there is a close relation between an upward TCP connection (from the proxy server to the Web server) and a downward TCP connection (from the client to the proxy server). That is, the difference of the throughput of both connections should be taken into account when socket buffers are assigned to them. For example, when the throughput of a certain downward TCP connection is larger than that of other concurrent downward TCP connections, the larger size of socket buffer should be assigned to the TCP connection by using E-ATBT. However, if the throughput of the upward TCP connection corresponding to the downward TCP connection is low, the send socket buffer assigned to the downward TCP connection is likely not to be fully utilized. In this case, the unused send socket buffer should be assigned to the other concurrent TCP connections having smaller socket buffers, hence, that the throughputs of those TCP connections would be improved.

There is one problem to realize the above-mentioned method. The TCP connection is identified with the control blocks, `tcpcb`, by the kernel. However, the relation between the upward and downward connections cannot be known by the kernel. Two possible ways to overcome this problem are considered as follows:

- The proxy server monitors the utilization of the send socket buffer of downward TCP connections. Then, it decreases the assigned buffer size of connections whose send socket buffers are not fully utilized.
- When the proxy server sends the document transfer request to the Web server, the proxy server attaches an information of the relation to the packet header.

The former algorithm can be done only by the modification of the proxy server. On the other hand, the latter algorithm needs the interaction of the HTTP protocol. In the higher abstract model, the above two algorithms have a same effect. However, the latter has a implementation difficulty while it can achieve a precise control.

Control of Receive Socket Buffer

In most of past researches, it was assumed that a receiver host has enough large size of receive socket buffer, considering that the performance bottleneck of the data transfer is not at the endhosts, but within the network. Therefore, many OSs assign a small size of the receive socket buffer to each TCP connection. For example, the default size of the receive socket buffer in the FreeBSD system is 16 [KBytes]. Now it is very small [14] because the network bandwidth is dramatically increased in the current Internet, and the performance of the Internet servers becomes higher and higher.

To avoid the performance limit by the receive socket buffer, the receiver host should adjust its receive socket buffer size to the congestion window size of the sender host. This can be done by monitoring the utilization of the receive socket buffer, or by adding information about the window size to data packet header, as described above. In the simulation in the next section, we suppose that the proxy server can obtain complete information about required sizes of the receive socket buffer of upward TCP connections and control them according to the required size.

3.2 Connection Management

As explained in Subsection 2.2, a careful treatment of persistent TCP connections on the proxy server is necessary for an effective usage of the resources of the proxy server. We propose a new management scheme of persistent TCP connections at the proxy server by considering the amount of the remaining resources. The key idea is as follows. When the load at the proxy server is low and the remaining resources are much enough, it tries to keep as many TCP connections as possible. On the other hand, when the resources at the proxy server are going to be short, the proxy server tries to close the persistent TCP connections and free the resources, such that the released resources can be used for new TCP connections.

The remaining resources of proxy servers should be monitored for realizing the above-mentioned control method. The resources for establishing TCP connections at the proxy server include *mbuf*, *file descriptor*, and *control blocks*. The total amount of these resources cannot be changed dynamically after the kernel is booted. However, the total and remaining amounts of these resources can be observed in kernel system [9]. Therefore, we introduce threshold values of the utilization for these resources, and if one of utilization level of those resources, calculated by the kernel system at regular intervals, reaches the threshold, the proxy server starts closing the persistent TCP connections and releasing the resources assigned to the connections.

The proxy server maintains persistent TCP connections as follows. See also Figure 2. When a TCP connection becomes idle, the proxy server records the connection and the current time. For fast lookup of the record, we use the hashing algorithms, of which key is the combination of source/destination IP addresses and the port number of the TCP connection. We also introduce a list, called a *time scheduling list*, to put the persistent connections in order of the time length that they are persistent. When a new persistent TCP connection is registered hash table, it is added at the end of the time scheduling list, so that the proxy server can select the older persistent TCP connections to be closed.

Each entry in the hash table has the socket file descriptor of the corresponding to the TCP connection, which is used later to identify the connection. When the proxy

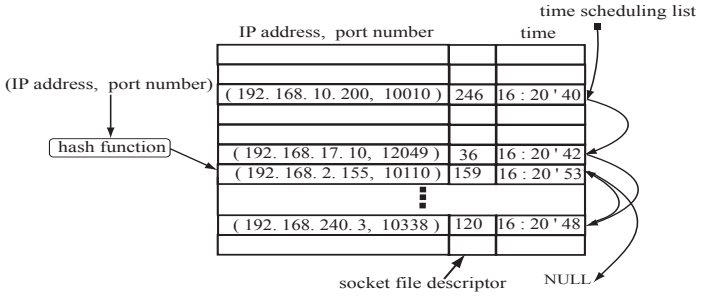


Fig. 2. Management Scheme of Persistent TCP Connections

server closes some of persistent TCP connections, it selects them from the top of the time scheduling list, by which the proxy server can close the older persistent connections. When a certain persistent TCP connection in the hash table becomes active before closed, or when it is closed by persistent timer expiration, the proxy server removes the corresponding entry from the hash table and the time scheduling list. All operations on the persistent TCP connections can be performed by simple pointer manupilations and hash operations.

For the further effective resource usage, we also add the mechanism that the amount of resources assigned to the persistent TCP connections is decreased gradually after the connection is inactive. The socket buffer is not necessary at all when the TCP connection becomes idle. However, we gradually decrease the send socket/receive buffer size of persistent TCP connections by taking account of the fact that as the connection idle time continues, the possibility that the TCP connection is ceased becomes large.

4 Simulation Experiments

In this section, we investigate the effectiveness of our proposed scheme through simulation experiments using ns-2 [15]. Figure 3 shows the simulation model. In this figure, the bandwidths of the links between client hosts and an HTTP proxy server and those between the proxy server and Web servers are all set to 100 Mbps. To see the effect of various network conditions, we set the packet loss probability on each link to be 0.0001, 0.0005, 0.001, 0.005 or 0.01. That is, one-fifth of the links is assigned one of the above values. The propagation delay of each link between the client hosts and the proxy server is also varied as ranged from 10 msec and 100 msec, and that between the proxy server and the Web servers is from 10 msec and 200 msec. The propagation delays of each link is determined randomly from the above ranges. The number of Web servers is fixed at 50, and that of the client hosts is changed as 50, 100, 200 and 500. We ran 300 sec simulation in each experiment.

In the simulation experiments, each client host selects one of the Web servers at random and generates a document transfer request via the proxy server. The distribution of the requested document size is obtained from [12], which is given by the combination of a log-normal distribution for small documents and a Pareto distribution for large

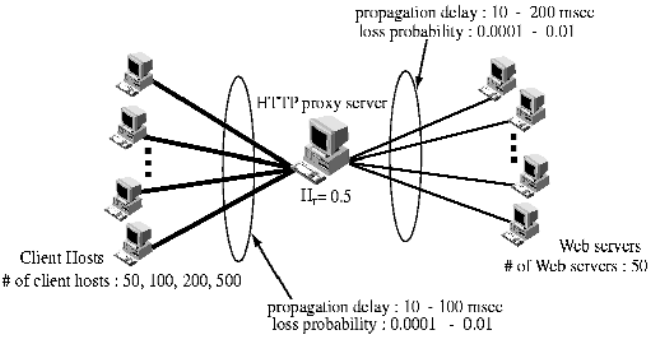


Fig. 3. Simulation Model

ones. Note that since we focus on the resource and connection management of proxy servers, we have not considered detailed algorithms of the caching behavior, including the cache replacement algorithms. Instead, we set the hit ratio, H_r, to 0.5. Using H_r, the proxy server decides either to transfer the requested document to the client directly, or to deliver it to the client after downloading it from the Web server. The proxy server has 3200 KBytes of socket buffer and assigns it as send/receive socket buffer to TCP connections. It means that the original scheme can establish at most 200 TCP connections concurrently, since it fixedly assigns 16 KBytes of send/receive socket buffer to each TCP connection.

In what follows, we compare the performance of the following 4 schemes; scheme (1) which does not use any enhanced algorithms in this paper, scheme (2) which uses E^2-ATBT, scheme (3) which uses E^2-ATBT and the connection management scheme described in Subsection 3.2, and scheme (4) which uses E^2-ATBT and the connection management scheme with the algorithm that gradually decreases the socket buffer assigned to the persistent TCP connections. Note that for scheme (3) and (4), we do not explicitly consider the amount and the threshold value of each resource, as explained in Subsection 3.2. Instead, we introduce N_{max}, the maximum number of connections which can be established simultaneously, to simulate the limitation of the proxy server resources. In scheme (1) and (2), newly arrived requests are rejected when the number of TCP connections in the proxy server is N_{max}. On the other hand, scheme (3) and (4) forcibly terminate some of persistent TCP connections that are unused for the document transfer, and establish the new TCP connections. For scheme (4), we exclude persistent TCP connections from calculation process of E^2-ATBT algorithm, and halve the assigned size of socket buffer every 3 sec. The minimum size of the socket buffer is 1 KByte.

4.1 Evaluation of Proxy Server Performance

We first investigate the performance of the proxy server. Here we define the performance of proxy server as the total size of the documents transferred in both directions by the proxy server during 300 sec simulation time. In Figure 4, we plot the performance of

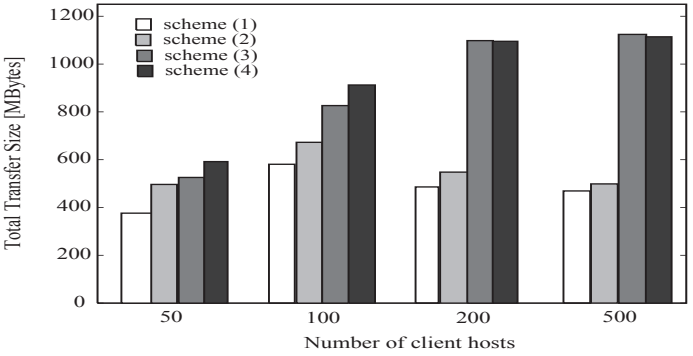
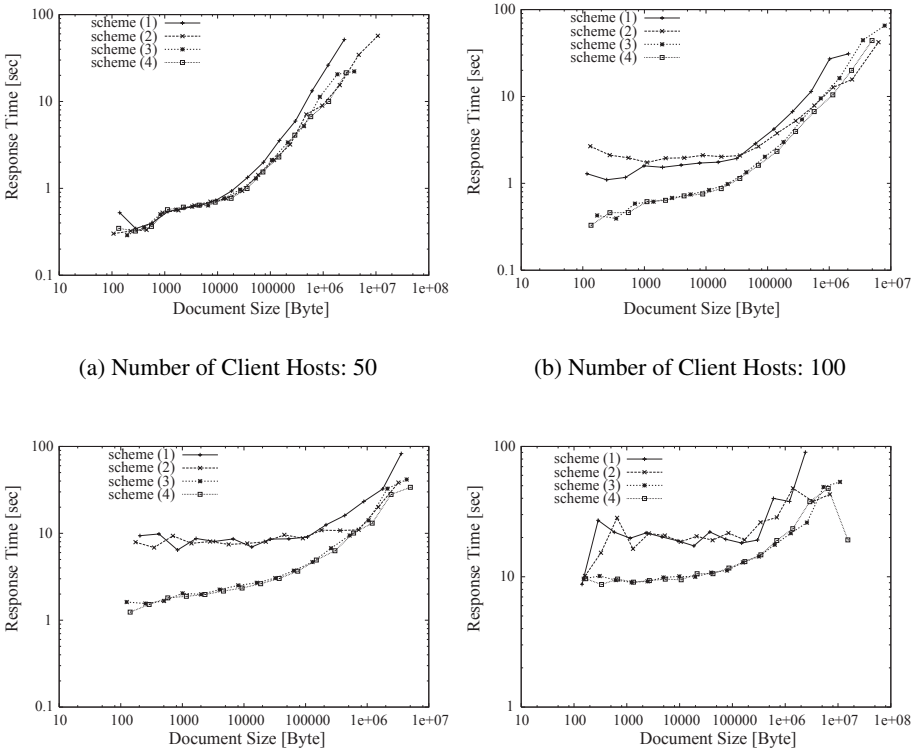


Fig. 4. Simulation Result: Proxy Server Performance

the proxy server as a function of the number of client hosts. Here, we set N_{max} to 200. It is clear from this figure that the performance of the original scheme (scheme (1)) is decreased in the case of the larger number of client hosts. It is because when the number of client hosts is larger than N_{max}, the proxy server rejects some of document transfer requests, although most of N_{max} TCP connections are idle, which means that they do nothing but waste the resources of the proxy server. The results of scheme (2) in Figure 4 shows that E^2-ATBT can improve the proxy server performance regardless of the number of client hosts. However, it also shows that the performance also degrades when the number of client hosts increases. This means that E^2-ATBT cannot solve the problem of 'idle' persistent TCP connections, and that it is necessary to introduce a connection management scheme to overcome this problem.

We can also see that scheme (3) can significantly improve the performance of the proxy server, especially when the number of client hosts is large. It is since when the proxy server cannot accept all connections from the client hosts, which corresponds to the case where the number of client hosts is larger than 200 in Figure 4, scheme (3) would close idle TCP connections for newly arriving TCP connections to be established. It results in that the number of TCP connections which actually transfer documents increases largely. Scheme (4) can also improve the performance of the proxy server, especially when the number of client hosts is small, as shown in Figure 4. In the case of larger number of client hosts, however, there is little performance improvement. It can be explained as follows. When the number of client hosts is small, most of the persistent TCP connections at the proxy server are kept established, since the proxy server has enough resources to accommodate 50 client hosts. Therefore, the socket buffer assigned to the persistent TCP connections can be effectively re-assigned to other active TCP connections by scheme (4). When the number of client hosts is large, on the other hand, the persistent TCP connections are likely to be closed before scheme (4) begins to decrease the assigned socket buffer. It results in that scheme (4) can do nothing against the persistent TCP connections.



(a) Number of Client Hosts: 50

(b) Number of Client Hosts: 100

(c) Number of Client Hosts: 200

(d) Number of Client Hosts: 500

Fig. 5. Simulation Result: Response Time

4.2 Evaluation of Response Time

We next show the evaluation results of response time of document transfer, which corresponds to the user-perceived performance. We define the response time as the time from when a client host sends a document transfer request to when it receives the requested document. It also includes the waiting time for connection establishment. Figure 5 shows the simulation results. We plot the response time as a function of document size for the four schemes. From this figure, we can clearly observe that the response time is much improved when our proposed scheme is applied especially when the number of connections is large (Figure 5 (b)-(d)). However, when the number of client hosts is 50, the proposed scheme does not help improving the response time. For this, the server resources are enough to accommodate 50 client hosts and all TCP connections are soon established at the proxy server. Therefore, response time can not be improved so much. Note that since E^2-ATBT can improve the throughput of TCP data transfer to some degree, the proxy server performance can be improved as shown in the previous subsection.

Although schemes (3) and (4) can improve the response time largely, there is little difference between the two schemes. This can be explained as follows. Scheme (4) decreases the assigned socket buffer to persistent TCP connections and re-assign it to other active TCP connections. Although the throughput of the active TCP connections becomes improved, its effect on the response time is very small compared with the effect of introducing scheme (3). However, scheme (4) is worth to be used at the proxy server, since scheme (4) can give a good effect on the proxy server performance as shown in Figure 3.

From all of the above simulation results, we can say that scheme (4), which has all enhanced mechanisms proposed in this paper, is the best one to improve both the performance of the proxy server and response time of client hosts, regardless of the number of client hosts.

5 Conclusion

In this paper, we have proposed a new resource management scheme for HTTP proxy servers. Our proposal scheme has two algorithms. One is an enhanced E-ATBT, the scheme for managing the socket buffer considering about the relation between the upward and downward TCP connections, which is one of the characteristics of the proxy servers. It also manages the receiver socket buffer, which is not considered in the original E-ATBT. The other is the scheme for managing TCP connections at the proxy servers. It maintains persistent TCP connections, and it aggressively closes them when the resources lack. We have evaluated our scheme through some simulation experiments, and confirmed that our scheme can improve the performance of the proxy servers, and reduce document transfer time experienced by client hosts.

We are now implementing the proposed scheme to the actual proxy server, and to evaluate it through experiments using the actual network. We also plan to introduce other kinds of the resources of Web servers and proxy servers to our resource management scheme. For example, a CPU processing time should be considered for executing CGI programs, which is one of the bottleneck of the busy Web servers.

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," Telecommunication Advancement Organization of Japan under the Project "Global Experimental Networks for Information Society Project," and the "Research on High-performance WWW server for the Next-Generation Internet" program of from the Telecommunications Advancement Foundation.

References

1. G. Hasegawa, T. Terai, T. Okamoto, and M. Murata, "Scalable socket buffer tuning for high-performance Web servers," in *Proceedings of IEEE ICNP 2001*, Nov. 2001.
2. 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.

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. P. Druschel and 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.
5. Proxy Survey, available at *http://www.delegate.org/survey/proxy.cgi*.
6. A. Feldmann, R. Caceres, F. Douglis, G. Glass, and M. Rabinovich, "Performance of Web proxy caching in heterogeneous bandwidth environments," in *Proceedings of IEEE INFOCOM '99*, pp. 107–116, 1999.
7. Squid Home Page, available at *http://www.squid-cache.org/*.
8. Apache proxy `mod_proxy`, available at *http://httpd.apache.org/docs/mod/mod_proxy.html*.
9. M. K. McKusick, K. Bostic, M. J. Karels, and J. S. Quarterman, *The Design and Implementation of the 4.4 BSD Operating System*. Reading, Massachusetts: Addison-Wesley, 1999.
10. J. Padhye, V. Firoiu, D. Towsley, and J. Krusoe, "Modeling TCP throughput: A simple model and its empirical validation," in *Proceedings of ACM SIGCOMM '98*, pp. 303–314, Aug. 1998.
11. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee, "Hypertext transfer protocol – HTTP/1.1," *Request for Comments (RFC) 2068*, Jan. 1997.
12. P. Barford and M. Crovella, "Generating representative Web workloads for network and server performance evaluation," in *Proceedings of the 1998 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pp. 151–160, July 1998.
13. 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.
14. M. Allman, "A Web server's view of the transport layer," *ACM Computer Communication Review*, vol. 30, pp. 10–20, Oct. 2000.
15. The VINT Project, "UCB/LBNL/VINT network simulator - ns (version 2)." available at `http://www.isi.edu/nsnam/ns/`.