Vol. 47 No. 2 IPSJ Journal Feb. 2006

Regular Paper

A Mechanism for TCP Performance Improvement in Asymmetrical Broadband Access Environment

Teruyuki Hasegawa† and Toru Hasegawa†

This paper describes a novel mechanism for achieving sufficient TCP performance in bulk data transfer in some asymmetrical environment without introducing additional functions in customer premises. On today's Internet, several types of broadband access media have an asymmetrical bandwidth characteristic, whereby the downstream link bandwidth is larger than the upstream. However, such an asymmetrical environment may cause TCP performance degradation due to upstream link congestion. To solve this problem, we propose a PEP-based approach to improve TCP file downloading throughput by reducing upstream traffic, using our "oversized MSS with compulsory IP fragmentation" technique. The results of our performance evaluation show that our experimental proxy implementation is capable of accelerating TCP throughput to about three times as fast as without the proxy.

1. Introduction

Recently, with the increasingly widespread use of the Internet, various types of access technology have been developed one after another. In order to meet the demand for broadband access, several types of Internet access (e.g., wireless, ADSL and satellite) have introduced some bandwidth asymmetry, where the downstream link bandwidth is larger than the upstream, in consideration of the implementational limitation of access systems at customer premises and the characteristic of Internet traffic that customers mainly download information from the Internet.

Meanwhile, most Internet applications such as web access use TCP[1] as a transport protocol. For reliable data transfer, TCP should return acknowledgement segments (ACKs) in the opposite direction to the transfer of data segments (DTs). Therefore, TCP file downloading throughput can be limited in an extremely asymmetrical environment, because the upstream link bandwidth is insufficient for tranfer of ACKs[2]. Furthermore, some new applications (e.g., P2P and VoIP) which tend to use upstream bandwidth constantly, are now becoming popular. Such applications can reduce the residual upstream bandwidth for ACK transfer.

In order to solve this problem, various proposals have been made for reducing ACK traffic[2)~6]. However, these proposals require some changes in customer premises such as access routers or host terminals. From the viewpoint of initial and maintenance costs, it is problematic to require every customer to adopt such modifications. In addition, reducing the number of ACKs[5] creates another problem in that the TCP congestion avoidance and retransmission mechanisms[7] can be influenced because the relation between DTs and ACKs is corrupted.

In this paper, we propose a new mechanism for reducing ACK traffic based on a "PEP (performance enhancement proxy)[8] with large MSS (maximum segment size)[2]" approach. This mechanism requires the introduction of a special proxy only on the carrier side, such as the ISP's access network edge. The following is a summary of our mechanism:

(1) The proxy splits a TCP connection established between a server and a client into two individual TCP connections in order to prefetch DTs from the server and accumulate them.

(2) The proxy reassembles several DTs into a large DT whose segment size is several times larger than the path MTU (maximum transfer unit) size (and hence the receiver host's MSS), and then transmits it to the client. This reduces the number of corresponding ACKs.

(3) This oversized DT is compulsorily fragmented into several IP fragments according to the path MTU size, using the IP fragmentation function in the proxy.

(4) In order to prevent TCP throughput degradation due to large propagation delay in a satellite or wireless access

† KDDI R&D Laboratories, Inc.

environment[9], the *TCP gateway approach*[10),11] with its speculative "sending ahead" mechanism can be also applied instead of the TCP window scale option[12].

The above procedures can accelerate TCP in bulk data transfer throughput without requiring the introduction of any additional functions into customer premises. It is also expected that the relation between DTs and ACKs will not be corrupted. This can eliminate the impact on TCP congestion avoidance and retransmission mechanisms. Furthermore, it is possible to eliminate the effort of TCP window size tuning on the customer side by harmonizing with our TCP gateway approach even in the case of an access environment with a large propagation delay. On the other hand, it is necessary to verify the behavior of real TCP/IP implementations, especially toward items 3 and 4, which may cause some unexpected TCP/IP packets. We have therefore verified and evaluated the proposed mechanism through our experimental proxy implementation using a real instance of TCP data transfer, i.e. web-page access and 10 MB binary file downloading based on HTTP.

The reminder of this paper is organized as follows. Section 2 summarizes the impact of an asymmetrical environment on TCP performance, and surveys some related work. Section 3 explains the details of our new mechanism. Section 4 describes our experimental implementation of the proposed mechanism using the Linux kernel and Apache proxy software. Section 5 gives the results of our evaluation, taking account of a large propagation delay. Section 6 offeres some discussion. Section 7 concludes the paper.

2. TCP with Bandwidth Asymmetry

2.1 Overviews

Figure 1 shows an example of a network configuration with asymmetrical access links. A client host on the customer side accesses a server on the Internet, using a broadband downstream link (e.g., 8 Mbps) and a narrowband upstream link (e.g., 64 kbps).

An example of a TCP data communication sequence between the server and the client is shown in **Fig. 2**. Because TCP generally returns an ACK after receiving two DTs, TCP performance can be limited by the transmission speed of ACKs if the upstream bandwidth is insufficient.

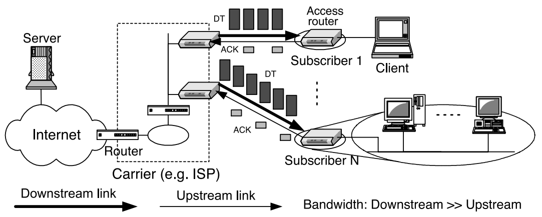

Fig. 1 Network with asymmetrical link.

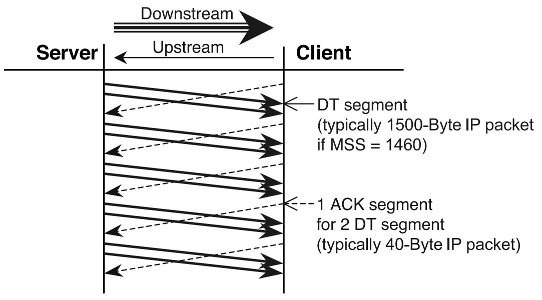
Downstream
Upstream
Server
Client
DT segment
(typically 1500-Byte IP packet if MSS = 1460)
1 ACK segment for 2 DT segment
(typically 40-Byte IP packet)

Fig. 2 TCP data communication sequence.

The following equation is used as an index to decide whether the upstream link bandwidth is large enough[5]:

$$k = \frac{BW_{downstream}/PS_{downstream}}{BW_{upstream}/PS_{upstream}}, \quad (1)$$

where BW and PS denote the link bandwidth and packet size, respectively. Let N be the number of DTs acknowledged by an ACK. Generally, the value of N is equal to 2, because the delayed ACK mechanism[13] is effective. At least $k \geq 1/N$ is required to avoid upstream link congestion caused by ACK transmission. For example, if the IP datagram size of an ACK on the upstream link is 40 bytes and that of a DT on the downstream link is 1,500 bytes, as shown in Fig. 2, more than 110 kbps upstream bandwidth should be arranged for an 8 Mbps downstream bandwidth. In addition, the data link layer overhead (e.g., PPP header/trailer, Ethernet header/trailer, etc.) should be considered.

2.2 Related Work

There have been several studies of how to realize sufficient TCP performance in an asymmetrical environment. The following section surveys some representative ones.

2.2.1 Header Compression

RFC1144[3] and RFC2507[4] specify TCP/UDP/IP header compression mechanisms for point-to-point (e.g., PPP) links. Here, the header information volume is compressed by transmitting only the information on the difference between the present header and the previous one in the same IP flow. This can reduce ACK traffic volume on a byte-count basis.

2.2.2 ACK Filtering and Reconstruction

Balakrishnan, et al.[5] proposed an ACK Filtering (AF) mechanism in which access routers at customer premises select proper ACKs returning in the upstream direction. This can reduce ACK traffic volume on a packet-count basis. The same authors suggested that only the latest ACK in a TCP connection should be left in the transmission queue, and that previous ACKs in the same connection should be discarded when an upstream link is congested.

However, since TCP congestion avoidance and retransmission mechanisms work on the basis of the number of ACKs as well as their contents[7], numerical reduction of ACKs caused by AF can produce harmful side effects on these mechanisms. To mitigate such side effects, Balakrishnan, et al.[6] propose ACK Reconstruction (AR), where edge routers in carrier premises re-generate ACKs filtered on the customer side.

2.2.3 Large MSS

RFC3449[2] also mentions the possibility of large MSS to reduce the number of ACKs generated per transmitted byte of DTs. Large MSS can be realized at both ends if individual subnetworks between the sender and receiver support large MTU size. However, the majority of current Internet hosts and routers use a 1,500-byte MTU size, and the corresponding MSS is 1,460 bytes, based on the standard Ethernet frame size. Although RFC3449 also points out that end hosts may use large MSS employing IP fragmentation by routers even if the path MTU is smaller than that of the end hosts, at least enlargement of the IP MTU size is required for almost all the end hosts in customer premises in order to realize large (i.e., over 1,460-byte) MSS.

3. Proposed Mechanism

3.1 Design Principles

We designed a new enhancement mechanism for TCP performance with bandwidth asymmetry according to the following principles[14),15)].

(1) All the existing approaches described in Section 2.2 require some changes in customer premises, in terms of kernel-level protocol implementation or network interface configuration. In contrast, taking account of the initial and maintenance costs, our mechanism aims to improve TCP performance by introducing a small number of additional apparatuses into carrier premises, requiring only a slight (or if possible no) effort such as proxy setting in customer premises.

(2) These apparatuses use an approach based on packet count reduction of ACKs over asymmetrical links. In order to prevent it from having side effects (see Section 2.2.2) on the TCP entity of Internet servers, the apparatuses individually return ACKs according to the TCP standard and conceal the ACK reduction from the Internet servers.

3.2 Communication Sequence

Figure 3 shows a network configuration example in which the proposed mechanism is adopted. As shown in the figure, a special proxy system is installed on the carrier side. In a similar way to that described in Refs. 16) and 17), this proxy splits a TCP connection established between a server and a client into two individual TCP connections: one between the server and the proxy, and the other between the proxy and the client.

An example of a sequence of TCP communication using the proposed mechanism is shown in **Fig. 4**. The proxy accelerates the TCP performance as follows:

(1) When a client accesses a server via the proxy, two individual TCP connections are established: one between server and the proxy, and the other between the proxy and the client.

(2) On establishment of a TCP connection, the TCP maximum segment size (MSS), whose value is decided the basis of the IP MTU size, is advertised by both TCP ends. Between the server and the proxy, both TCP ends use a smaller MSS value according to the TCP standard. In the case of Fig. 4, the MSS value is set to 1,460. The server starts transmission of

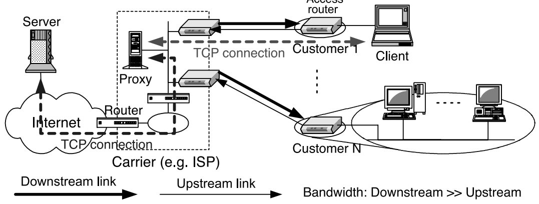

Fig. 3 Network with proposed mechanism.

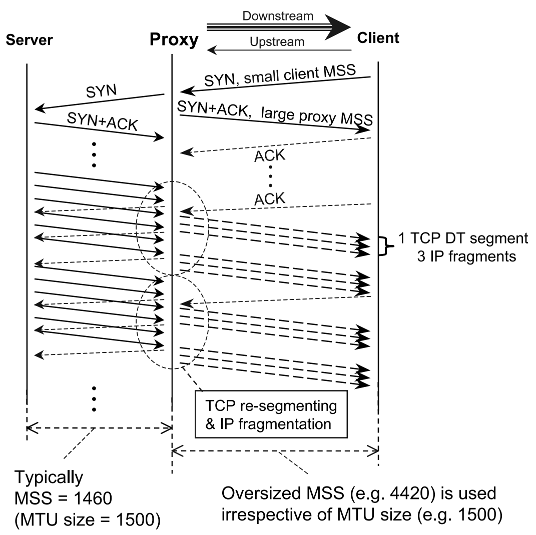
Downstream
Upstream
Server
Proxy
Client
SYN
SYN+ACK
SYN, small client MSS
SYN+ACK, large proxy MSS
ACK
ACK
1 TCP DT segment
3 IP fragments
TCP re-segmenting & IP fragmentation
Typically MSS = 1460 (MTU size = 1500)
Oversized MSS (e.g. 4420) is used irrespective of MTU size (e.g. 1500)

Fig. 4 Communication sequence example.

DTs whose segment sizes (excluding the TCP/IP header) are equal to or less than 1,460 bytes.

(3) On the other hand, between the proxy and the client, the proxy reports an oversized MSS value (e.g., 4,420) to the client irrespective of the proxy's MTU size. Moreover, the proxy compulsorily uses this oversized MSS to generate DTs, even if the client advertises a smaller MSS (e.g., 1,460).

(4) Instead of the client, the proxy returns ACKs to the server according to the TCP standard so that the proxy can prefetch DTs from the server. These prefetched DTs are re-segmented into user data and accumulated in the proxy. In order to accumulate sufficient user data for generating an oversized TCP DT, the Nagle algorithm [18] is used in the proxy's TCP on the client side, which prohibits the consecutive transmission of small (non-oversized) TCP DTs.

(5) The TCP layer in the proxy generates a large TCP DT in accordance with the oversized MSS (see item (3)) and feeds this down to the IP layer. The IP layer in the proxy fragments this oversized TCP DT into multiple IP fragments according to the path MTU (e.g., 1,500) [19] between the proxy and the client, and then transmits these fragments to the client. In Fig. 4, one TCP DT is transmitted by using three IP fragments.

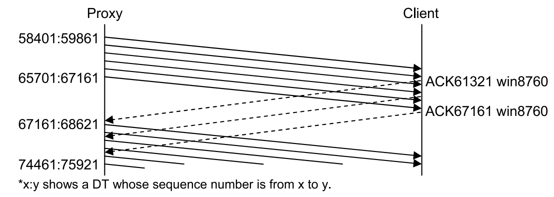

Fig. 5 Communication sequence with insufficient window size.

(6) The IP layer in the client defragments these IP fragments into an oversized TCP DT. Since this TCP DT will be processed as one DT in the client, the number of ACKs can be reduced, but the relation between DTs and ACKs is not corrupted.

(7) It should be noted that this mechanism assumes that the following conditions are satisfied:

- The client side TCP implementation makes it possible to process a DT whose size is larger than the MSS advertised by the client itself.
- The IP defragmentation process is not a very heavy load in the client.

In addition, our mechanism cannot suppress the number of ACKs efficiently in the case of short-lived TCP sessions, where 1 or 2 segments related to a TCP 3-way handshake transmitted on the upstream link (e.g., SYN and the first ACK in Fig. 4) are not negligible compared with the number of DTs.

3.3 Consideration of Access Environment

As described in item (4) of Section 1, the large propagation delay in satellite links is another performance bottleneck in addition to the bandwidth asymmetry, because originally the maximum TCP window size is limited to 65,535 bytes. **Figure 5** shows a typical TCP communication sequence with an insufficient TCP window size, where the server suspends DT transmission until an ACK with window update returns from the client.

In order to cope with this problem, we adopt either of the following two approaches:

(1) The *TCP window scale option* [12], which enlarges the maximum window size to 1 GB and has been implemented in today's major OSes. However, some configuration effort is required in the customer premises as well as the carrier-side proxy.

(2) The *TCP gateway approach* [10),11], where

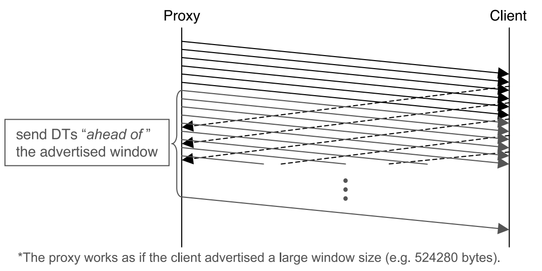
Proxy
Client
send DTs "*ahead of*" the advertised window
*The proxy works as if the client advertised a large window size (e.g. 524280 bytes).

Fig. 6 Communication sequence with TCP gateway approach.

the proxy speculatively sends DTs ahead of the advertised window from the client, as shown in **Fig. 6** for bulk data transfer. Although this is quite an aggressive approach, which can ignore receiver-initiated flow control, TCP window size tuning can be omitted in the customer premises. The merits and demerits (including some workarounds) of our TCP gateway approach are discussed in Refs. 10) and 11).

4. Implementation

4.1 Implementation Principles

Since our mechanism assumes that some conditions described in Section 3.2 item (7) are satisfied, it is essential to verify the applicability of our mechanism to today's major operating systems (OSes). We therefore experimentally implemented a proxy system according to the following principles:

(1) The system is constructed by adopting existing proxy and TCP/IP implementations wherever possible. The communication sequences described in Section 3.2 are realized by adding the minimum modifications to these implementations.

(2) There are two methods whereby a client can access an Internet server via a proxy. One is for the client to explicitly set proxy configurations in its application level. The other is for some intermediate node (e.g., the carrier edge router or the proxy itself) to redirect the packets automatically to the proxy. The latter method (e.g., transparent proxy[20]) is better in terms of ease of use. However, we adopt the former method, because it is easier to implement and there is no difference between these two methods from the viewpoint of our implementation objectives, i.e., applicability and performance evaluation.

(3) The path MTU value, which is used in IP fragmentation of the proxy, can be various for each client. For example, some clients may access the proxy via a link with a small MTU size (e.g., 552 bytes). Strictly, the path MTU discovery mechanism[19] should be introduced to decide the path MTU value. On the other hand, such a mechanism is not crucial to our objectives. Thus, for ease of implementation, the proxy simply uses the downstream link MTU size as the value of the path MTU.

4.2 Implementation Details

According to the above principles, we have constructed the proxy system on the basis of the Linux 2.2.14 kernel and *Apache 1.3.12*[21] software. We added the following configuration and modifications into the TCP/IP stack in the Linux kernel. For the proxy server, we executed *Apache* unchanged on this modified kernel. Note that this implementation does not support some TCP options such as T/TCP[22].

(1) By executing the *route*[23] command with the *mss* option in the proxy system, it is possible to change the path MTU size (and hence the TCP MSS) for the downstream link. If the proxy wants to use a 4,460-byte path MTU size (i.e., a 4,420-byte MSS), the following command should be executed:

$$route\ add\ \text{-}net\ NW_{addr}/NW_{pref}\ gw\ GW\ mss\ 4460$$

Here, NW_{addr}/NW_{pref} and GW indicate the network address/prefix of the client and the IP address of the next hop router, respectively. It should be noted that the mss option corresponds to the path MTU size, and therefore its value needs to be set 40 bytes larger than the requested MSS.

(2) Some TCP kernel routines (tcp_ipv4.c: tcp_v4_rcv()) are modified so that the MSS field in a SYN segment received from the client is overwritten to the MSS value given in item (1) (e.g., 4,420). As a result, the TCP layer in the proxy is spoofed and considers that the large MSS value is successfully exchanged.

(3) According to the above procedures, the proxy's TCP layer generates a oversized DT and delivers it to the IP layer. Mean-

Table 1 Hardware and software specifications.

Type	CPU, Memory	OS	Application
Web server	P2-300 MHz, 128 MB	Linux 2.4.2	Apache 1.3.19
Proxy server	P2-450 MHz, 64 MB	Linux 2.2.14	Apache 1.3.12
Router	P3-1 GHz×2, 1.5 GB	FreeBSD 4.7	dummynet
Packet monitor	P2-233 MHz, 128 MB	Linux 2.4.2	tcpdump 3.4
Client #1	Ppro-200 MHz×2, 128 MB	FreeBSD 4.5	wget 1.8.1
Client #2	P2-233 MHz, 128 MB	Linux 2.4.2	Netscape 4.76, wget 1.6
Client #3	P3-600 MHz, 256 MB	Windows 2000*	IE 5.5, wget 1.5.3
Client #4	P3-600 MHz, 128 MB	Windows 98*	IE 5
Client #5	P3-800 AMHz-M, 256 MB	Windows XP*	IE 6

*No service pack is applied.

while, the IP layer considers that such oversized DTs can be transmitted without fragmentation, because of the configuration described in item (1). In order to fragment this DT into multiple IP fragments suited to the downstream link MTU size, some IP kernel routines (ip_output.c: ip_queue_xmit(), ip_fragment()) are rewritten.

(4) To realize it sending ahead mechanism for the TCP gateway, we also arranged another Linux kernel image in which one of the TCP kernel routines (tcp_input.c: tcp_ack()) is changed so that the proxy's TCP layer always recognizes the advertised TCP window size as 524,280 ($= 65535 \times 2^3$) bytes, taking account of our evaluation network configuration (see Section 5.1). Note that the TCP segment loss recovery mechanism taking account of the sending ahead mechanism [10),11)] is not implemented.

5. Evaluation

5.1 Network Configuration

We have evaluated our experimental implementation not only from the viewpoint of performance but also from that of conformance with today's major OSes. **Figure 7** shows the network configuration for our evaluation. We arranged several PCs, whose hardware and software specifications are shown in **Table 1**.

In this configuration, both web and proxy servers and a router are accommodated in a FastEthernet switch. The proxy can communicate with all the clients via the router. The TCP MSS values in the proxy are decided as follows:

- For the client side, either 4,420 or 1,460 bytes is selected. IP fragmentation is forced in the former case only.
- For the server side, only 1,460-byte MSS is used.

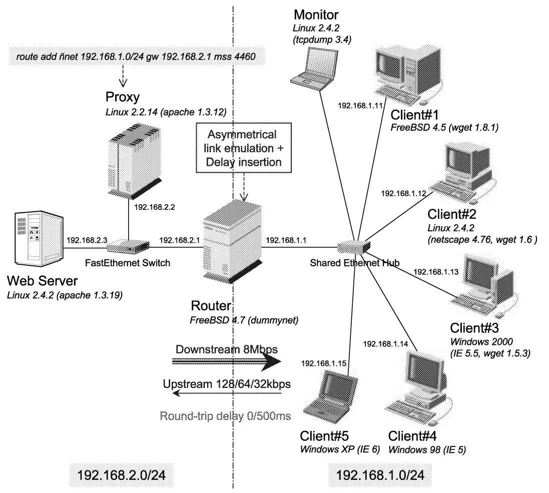

Fig. 7 Evaluation network configuration.

On the other hand, the router and the clients are connected via a shared Ethernet hub so that the monitor can capture and analyze packets exchanged between the proxy and the clients. Note that no packet loss occurs in this network configuration.

In addition, the dummynet [24)] function is activated in the router in order to emulate an asymmetrical environment with some propagation delay between the proxy and the clients. The bandwidth and delay assignments are as follows:

- 8 Mbps downstream bandwidth is assigned.
- On the other hand, one of the following upstream bandwidth is assigned: 128, 64, or 32 kbps.
- A propagation delay of 0 or 250 ms is inserted in each direction, i.e., Rtd (Round-trip delay) is 0 or 500 ms.

5.2 Conformance Verification

In conformance verification, all the clients (from #1 to #5) illustrated in Fig. 7 individually accessed the web server via the proxy and

Table 2 TCP parameters.

Rtd	Type	Rxwin	Txbuf	Note
0 ms	Proxy	–	65535	default
	Client #1	66608	–	default
	Client #2	63712	–	default
	Client #3	17520	–	default
500 ms (WSO)	Proxy	–	700000	
	Client #1	524140	–	
	Client #2	520144	–	
	Client #3	524140	–	
500 ms (TGW)	Proxy	–	700000	
	Client #1	66608	–	default
	Client #2	63712	–	default
	Client #3	17520	–	default

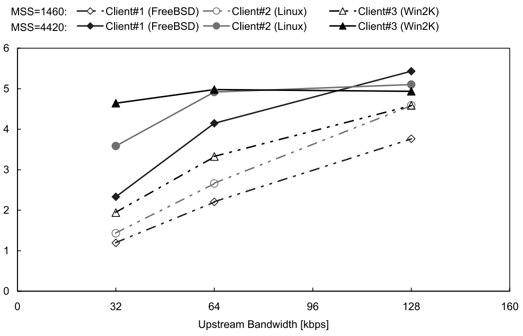

Fig. 9 TCP throughput (Rtd = 500 ms, WSO).

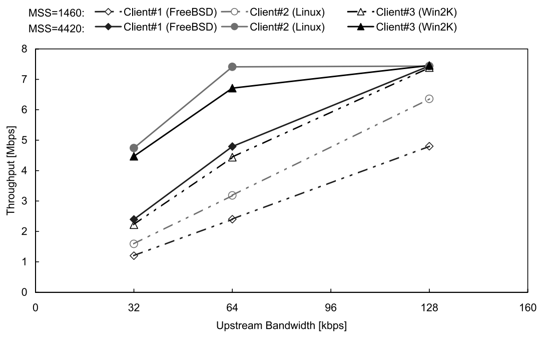

Fig. 8 TCP throughput (no delay).

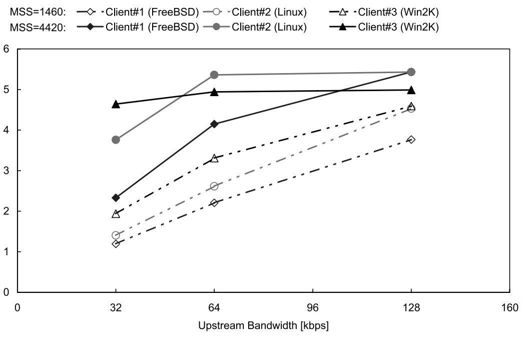

Fig. 10 TCP throughput (Rtd = 500 ms, TGW).

tried HTTP-based web-page access and binary file downloading using *Internet Explorer* (*IE*). As a result, all the clients were able to successfully download a file. These results prove that our mechanism can work with today's major OSes.

5.3 Performance Test

For the performance test, three clients (from #1 to #3) individually access the server via the proxy and perform HTTP-based 10 MB binary file download using the *wget* application. The TCP parameters in the proxy and/or clients are changed from the default values in 500 ms Rtd cases according to the approach adopted, i.e., the window scale option (WSO) or the TCP gateway (TGW). On the other hand, no TCP parameter modification is applied in 0 ms Rtd cases, where no WSO or TGW approach is used. **Table 2** shows the details of the parameters, where *Rxwin* is the maximum window size advertised by the clients and *Txbuf* is the sender socket buffer size configured in the proxy. Note in the TGW case that the proxy works as if it received a 524,280-byte window advertisement (see Section 4.2 item (4)), and note in the WSO case that *Rxwin* becomes nearly 524,280 bytes (shift.cnt = 3), considering that Bandwidth-Delay Product (BDP) is 500 KB (1 MB/sec × 0.5 sec) in 500 ms Rtd cases. The other TCP parameters are unchanged from the default settings in both clients and proxy. As a result, the TCP timestamp option, which requires an 12 additional bytes per ACK, is enabled in the cases of clients #1 and #2.

Figures 8, **9**, **10** depict the TCP throughput calculated from the captured packet information. These results show that our experimental proxy implementation is successfully accelerating TCP throughput in all cases, including the 500 ms Rtd cases. In particular for a 32 kbps upstream bandwidth, the throughput for client #2 is approximately 3 times as fast.

6. Discussion

6.1 TCP Throughput Analysis

According to the discussion in Section 2.1, upstream bandwidth is expected to be extremely limited in the 32 kbps and 64 kbps cases. As shown in Figs. 8 through 10, the results without our mechanism (MSS: 1,460 bytes) confirm this expectation, i.e., TCP throughput is degraded in proportion to the upstream bandwidth. On the other hand, when our mechanism is used (MSS: 4,420 bytes), all the clients can obtain higher throughput in any

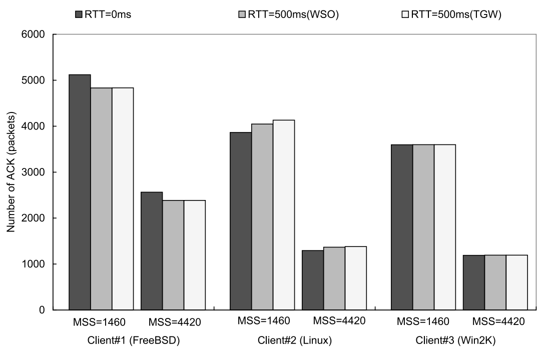

Fig. 11 Number of ACKs (32 kbps upstream).

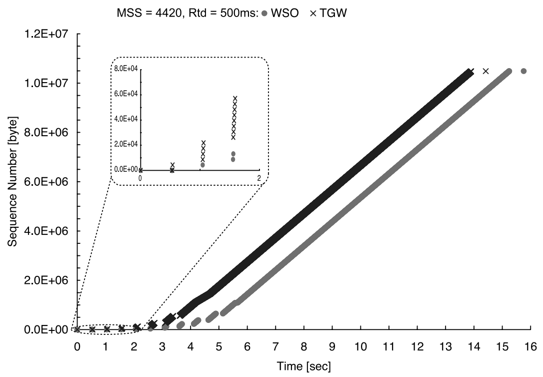

Fig. 12 Time sequence graph (client #2, 64 kbps upstream).

upstream bandwidth case. For client #2 in the 32 kbps case, the throughput is about 3 times as fast.

Figure 11 shows the number of ACKs transmitted on the upstream link when its bandwidth is set to 32 kbps. Almost the same results are observed in other bandwidth cases. These results confirm that our proposed mechanism can suppress the number of ACKs. The suppression ratio is about 1/2 in client #1 (FreeBSD), and about 1/3 in clients #2 (Linux) and #3 (Windows 2000). The ratios observed in clients #2 and #3 are quite reasonable, because our mechanism uses 4,420 bytes MSS in the evaluation, where one TCP DT will be fragmented into three IP fragments. On the other hand, some analysis is required for the result in client #1. By tracing the correspondence between DTs and ACKs using captured packet data, we found that client #1 returns an ACK on receipt of every DT. This means that the delayed ACK mechanism does not work in FreeBSD when the received DT size is larger than the MSS value advertised by itself on establishment of a connection. Since client #1 generates more ACKs, the throughput improvement is relatively small compared with that for other clients, as shown in Figs. 8 through 10. On the other hand, we also confirmed the situation for clients #2 and #3, where an ACK is generated after receipt of two DTs, i.e., the relation between DTs and ACKs is maintained. Note that client #2 generates slightly more ACKs than #3 because the Linux TCP implementation yields one ACK per DT at the beginning of data transfer.

A throughput difference is also observed between clients #2 and #3 for A 1,460-byte MSS in Fig. 8 in the 32 kbps and 64 kbps upstream cases. This is mainly due to the segment size of ACKs in addition to the number of ACKs. As described in Section 5.3, the segment size of ACKs is 52 bytes (#2) or 40 bytes (#3), depending on the presence of the TCP Timestamp option. For example, in the 32 kbps upstream case, the byte amounts of ACKs are 200,928 bytes (#2) and 143,800 bytes (#3) which are inversely proportion at to the throughput results, i.e., 1.6 Mbps (#2) and 2.2 Mbps (#3). Since client #1 generates more ACKs with a 52-byte length, the byte amount of ACKs is 266,188 bytes and the throughput is limited to 1.2 Mbps.

On the other hand, comparing WSO cases with TGW cases (see Figs. 9 and 10), there are some throughput differences in client #2 (Linux) with a 4420-byte MSS, where TGW cases are 5–9% faster than WSO cases. Figure 12 illustrates the relation between communication time and TCP sequence number in client #2 at 64 kbps upstream. It is easy to see that the sequence number increase in the WSO case is slower than in the TGW case. By tracing the window field of ACKs, we found that the receive window advertised in client #2 is insufficient, starting from 5,840 bytes and increasing gradually (5,840, 5,840, 8,816, 13,224, ...) at the beginning of data transfer in the WSO case. As a result, the proxy can send only one DT per ACK until the third ACK is received, so that the speed of congestion window increase in the proxy's TCP is suppressed (see Time $\leq$ 1.1 sec in **Fig. 12**). As the response of the fourth ACK advertising a 13,224-byte receive window, two DTs are consecutively transmitted at Time = 1.6 sec.

In the case of TGW, on the other hand there is no restriction on congestion window increase, because a 524,280-byte receive window is always given to the proxy's TCP. Therefore 2, 4, and 8 DTs are transmitted at Time = 0.5, 1.1, and 1.6 sec, respectively.

6.2 Comparison with RFC3449 Large MSS Approach

As described in Section 2.2.3, RFC3449 [2)] mentions the possibility of large MSS. However, MSS is practically limited to 1,460 bytes (or smaller) because the majority of current Internet hosts and routers use a 1,500-byte MTU size, which is directly reflected in end-to-end MSS selection and path MTU discovery. Although RFC3449 refers to the possibility of IP fragmentation by routers in the case of a smaller path MTU than that of the end hosts, it does not mention the usage of oversized MSS irrespective of the end hosts' MTU size. In other words, servers (or the proxy) cannot use larger MSS than 1,460 bytes as long as the client hosts in the customer premises use a 1,500-byte MTU size.

Therefore, from the viewpoint of providing a carrier-side-only solution for TCP throughput improvement, we proposed a mechanism which dares to use oversized MSS (i.e., ignoring the client host's MSS and the proxy's IP MTU size) with intentional IP fragmentation inside a proxy itself. Note that further fragmentation in routers can be avoided by the use of a path MTU discovery mechanism [19)]. Although applying oversized MSS violates RFC1122 [13)] for a TCP sender, we believe our approach is promising on the basis of the following conviction:

(1) MSS information mainly affects TCP sender side behavior, helping to avoid unwanted IP fragmentation in routers and defragmentation in a receiver host. On the other hand, there is no critical reason to prohibit accepting oversized DTs from the viewpoint of TCP receiver side behavior, which is byte-stream-oriented.

(2) TCP implementations in today's major OSes are robust enough to accept oversized DTs and manage to process them as long as they are within the receive window.

The results of our conformance verification prove that our approach can work with today's major OSes.

However, it is also very important to guarantee interoperability with hosts that cannot accept such oversized DTs. We think the following workaround can be adapted:

(1) At the beginning of DT transmission, the proxy's TCP duplicately and consecutively transmits the following two types of DTs:
   - DTs applying oversized MSS (e.g., 4,420 bytes) with fragmentation, and
   - DTs truncated from the above DTs applying normal MSS (e.g., 1,460 bytes) without fragmentation.

(2) The proxy's TCP selects one of the above MSS values based on the acknowledgment number in the corresponding ACK.

Note that throughput fairness may be degraded between an oversized MSS session and a normal one, because the congestion window grows rapidly in proportion to the MSS. Furthermore, in oversized MSS cases, correspondence between DTs and ACKs is not maintaind in FreeBSD, as decribed in Section 6.1. In order to preserve throughput fairness, some additional mechanism needs to be designed for the proxy, based on Increasing TCP's Initial Window [25)] and Appropriate Byte Counting [26)] techniques.

6.3 Consideration of IP Fragmentation Processing Overhead

The proxy uses the IP fragmentation technique, whose drawbacks in processing overhead were pointed out by Kent and Mogul[27)]. However, with regard to the CPU load and memory usage in both proxy and the clients, we cannot distinguish the difference between the cases with and without IP fragmentation and defragmentation in our experimental environment. Since the additional overhead in the proxy is merely fragmenting TCP DTs one by one incoming from the TCP layer, which does not require any additional state information and/or buffer memory, we consider that the proxy overhead is not particularly harmful. Furthermore, by the use of the path MTU discovery mechanism for adjusting the MSS in the proxy, an additional IP fragmentation effort at intermediate routers, which may cause serious performance degradation, can be avoided. Meanwhile, as for the client, at least this fact proves that IP defragmentation is not a very heavy task for today's PCs if the following conditions are satisfied, considering the discussion in Ref. 27) and our evaluation environment:

(1) There is no fragment loss between the proxy and the client.

(2) There is no fragment re-ordering between the proxy and the client.

(3) There is no harmful defragmentation conflict, i.e., the client has only one DT

(= one IP IDentification) in the defragmentation process.

With regard to items (1) and (3), we suppose that the fragment loss ratio is very small (see Section 6.4). In such an environment, the number of DTs in an incomplete defragmentation situation is just one in most cases (i.e., no packet loss). Even if a packet loss occurs, at most two incomplete DTs need to be processed if the same IP ID is used on both the original and retransmitted DTs[27]. In addition, as is shown in Fig. 3, there are few multiplexing and demultiplexing points between the proxy and the client, i.e., it is expected that the client can receive IP fragments without reordering and conflict situations. Therefore items (2) and (3) can be preserved. As a result, we consider that the processing overhead of IP fragmentation and defragmentation is negligible.

6.4 Applicability of Proposed Mechanism

In order for the proposed mechanism to work effectively, it is important to discuss the influence of packet loss and MSS values.

The degradation of TCP-level DT segment loss probability is a serious problem in IP fragmentation. Since our experiments were conducted without any fragment loss, such a problem was concealed. However, if networks include links with some transmission error or nodes with congestion, DT losses can easily occur corresponding to MSS, because even a single IP fragment loss in a DT causes the loss of the whole DT. We therefore briefly studied how the increase in DT losses affects The TCP throughput and retransmission ratio for the original TCP user data amount in WSO cases.

First we will clarify our preconditions clear. Our approach is PEP[8] -based, where IP fragments caused by oversized MSS appear only in the asymmetrical access link and customer-side LANs behind it (if any), as shown in Fig. 3. We suppose that the proxy accommodates all the traffic outgoing to and incoming from customer premises, and does not overrun the access link capacity. This we consider only the transmission error probability on the access link. In addition, we also assume that every DT loss is detected on the basis of triple duplicate ACKs and recovered by single DT retransmission.

We estimate the theoretical throughput B_b on a bit-rate basis. According to Ref. 28), the theoretical packet rate B_p and the expected packet number of congestion windows $cwnd_p$ at loss detection satisfy

$$B_p \approx \frac{1}{Rtd}\sqrt{\frac{3}{2Np}} \tag{2}$$

and

$$cwnd_p \approx \sqrt{\frac{8}{3Np}}, \tag{3}$$

where p ($\ll 1$), N, and Rtd indicate the probability of DT segment loss, the number of DTs acknowledged by an ACK (normally $N = 2$), and the round-trip delay, respectively. Suppose that with a random bit error whose ratio is e ($\ll 1$) on the access link, the probability of DT segment loss p is given by:

$$p = 1 - (1 - e)^L \approx eL, \tag{4}$$

where L is the total bit amount in the physical layer for one DT transmission. Hereafter we neglect the TCP/IP header and datalink layer overhead for simplicity; i.e., L is approximated by the MSS value on a bit basis. B_p and $cwnd_p$ can be replaced by bit- based valuables B_b and $cwnd_b$ as follows:

$$B_b = B_p \times L \approx \frac{1}{Rtd}\sqrt{\frac{3L}{2Ne}} \tag{5}$$

and

$$cwnd_b = cnwd_p \times L \approx \sqrt{\frac{8L}{3Ne}}. \tag{6}$$

The estimation of the retransmission ratio is described below. Let $U = U_0$ be the total bit amount of user data, which yields $U_1 = pU_0$ bits retransmission. Since (re) transmission of U_n yields further retransmission $U_{n+1} = pU_n$, the total bit amount of retransmission $R = R_w$ and retransmission ratio $r = r_w = R/U$ are expressed by

$$\begin{aligned} r_w &= \frac{1}{U}\sum_{i=1}^{\infty} p^i U = \frac{p}{1-p} \\ &= \frac{1-(1-e)^L}{(1-e)^L} \approx eL, \end{aligned} \tag{7}$$

using Eq. (4).

According to Eqs. (4), (5) and (7), we can summarize the DT loss probability and its effects for small p ($\ll 1$) as follows.

(1) The DT loss probability p becomes worse in proportion to L (i.e., MSS).

(2) Meanwhile, the bit-rate-based throughput B_b is increased in proportion to $\sqrt{L}$, because a larger congestion window is expected.

(3) The retransmission ratio r tendency is the same as in item (1).

In order to use our mechanism efficiently, it is necessary to keep p small. For example, in the case of p = 1E-3, 0.1% retransmission bytes will occur between the proxy and the client. Since L is approximately 3.5E+4 bits in 4,420-byte oversized MSS, e = 2.8E-8 is required as the transmission quality of the access link in order to realize p = 1E-3. We are convinced that such quality is probable, e.g., IESS-308[29] G.826 quality requires e = 1E-9 with more than 99.36% reliability as a typical performance in degraded conditions.

Appropriate MSS value selection is another important issue. From the viewpoint of the ACK suppression ratio and expected throughput (see Eq. (5)), a larger MSS is preferred. However, considering the TCP fast retransmit mechanism[30], a sufficient number of DTs (i.e., a smaller MSS) is required either per receive windows or per congestion window (the smaller one) in order to ensure the generation of triple duplicate ACKs. In addition, a markedly larger MSS may cause an aggressive initial window increase, as described in Section 6.2. Practically, therefore, we think that the MSS should not be larger than 4 times the size of a normal MSS. In view of the 17520 receive window size in Windows 2000 and assuming a very small packet loss rate, we decided on a 4,420-byte MSS where 6 DTs exist in a receive window. Note that Eq. (6) will be used to decide the MSS size instead of the receive window size if the packet loss rate is relatively high.

7. Conclusions

In this paper we have described a novel mechanism for obtaining sufficient TCP bulk-data transfer performance over a link with bandwidth asymmetry. The proposed mechanism does not require any additional functions to be used in customer premises, but can suppress the number of ACKs by using our oversized MSS with compulsory IP fragmentation technique to avoid upstream link congestion. Moreover, it is possible to realize a completely configuration-free environment on the customer side by harmonizing with our proprietary TCP gateway approach even in the case of an access environment with a large propagation delay. We also implemented the mechanism as a proxy system and evaluated its effectiveness using a real instance of TCP/IP communication, downloading 10 MB file via HTTP in an error- free environment. The proxy can work together with today's major OSes without any problem. The results of our performance evaluation show that our experimental proxy implementation is capable of increasing the speed of TCP throughput in 10 MB bulk-data transfer by about 3 times in an extremely asymmetrical environment (32 kbps upstream and 8 Mbps downstream) in comparison with the case without the proxy. We also confirm that the processing overhead of this mechanism is negligible in both the clients and the proxy.

Acknowledgments The authors wish to thank Dr. T. Asami, President & CEO of KDDI R&D Laboratories Inc., for the continuous encouragement of this study.

References

1) Postel, J.: Transmission Control Protocol, RFC793 (Sep. 1981).
2) Balakrishnan, H., Padmanabhan, V.N., Fairhurst, G. and Sooriyabandara, M.: TCP Performance Implications of Network Path Asymmetry, RFC3449 (Dec. 2002).
3) Jacobson, V.: Compressing TCP/IP Headers for Low-Speed Serial Links, RFC1144 (Feb. 1990).
4) Degermark, M., Nordgren, B. and Pink, S.: IP Header Compression, RFC2507 (Feb. 1999).
5) Balakrishnan, H., Padmanabhan, V.N. and Katz, R.H.: The Effects of Asymmetry on TCP Performance, *Proc. ACM MOBICOM '97* (Sep. 1997).
6) Balakrishnan, H., Padmanabhan, V.N. and Katz, R.H.: The Effects of Asymmetry on TCP Performance, *ACM Mobile Networks and Applications* (*MONET*), Vol.4, No.3, pp.219–241 (1999).
7) Allman, M., Paxson, V. and Stevens, W.: TCP Congestion Control, RFC2581 (Apr. 1999).
8) Border, J., Kojo, M., Griner, J., Montenegro, G. and Shelby, Z.: Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations, RFC3135 (June 2001).
9) Allman, M., Glover, D. and Sanchez, L.: Enhancing TCP over Satellite Channels Using Standard Mechanisms, RFC2488 (Jan. 1999).
10) Miyake, Y., Hasegawa, T., Hasegawa, T. and Kato, T.: Proposal of TCP gateway for Satellite-Based Internet Access, *IEICE Trans. Commun.* (Japanese Edition), Vol.J84-B, No.12, pp.2330–2341 (Dec. 2001).
11) Hasegawa, T., Miyake, Y. and Hasegawa, T.: TCP Gateway for Satellite-Based Internet Ser-

vice Considering Accommodation of Multiple Customers, *J. IPS Japan*, Vol.43, No.12, pp. 3869–3877 (Dec. 2002).
12) Jacobson, V., Braden, R., and Borman, D.: TCP Extensions for High Performance, RFC1323 (May 1992).
13) Braden, R. (ed.): Requirements for Internet Hosts: Communication Layer, RFC1122 (Oct. 1989).
14) Hasegawa, T., Lagreze, M. and Hasegawa, T.: A Study of TCP Throughput Improvement over Asymmetrical Environment, *Proc. IPSJ DICOMO 2002 Symposum* (July 2002)(in Japanese).
15) Hasegawa, T., Hasegawa, T. and Lagreze, M.: A Mechanism for TCP Performance Enhancement over Asymmetrical Environment, *Proc. IEEE ISCC 2003* (June 2003).
16) Bakre, A. and Badrinath, B.R.: I-TCP: Indirect TCP for Mobile Hosts, *Proc. IEEE ICDCS'95* (May 1995).
17) Hasegawa, T., Hasegawa, T., Kato, T. and Suzuki, K.: Implementation and Performance Evaluation of TCP Gateway for LAN Interconnection through Wide Area ATM Network, *IEICE Trans. Commun.* (Japanese Edition), Vol.J79-B-I, No.5, pp.262–270 (May 1996).
18) Nagle, J.: Congestion Control in IP/TCP Internetworks, RFC896 (Jan. 1984).
19) Mogul, J. and Deering, S.: Path MTU Discovery, RFC1191 (Nov. 1990).
20) Kiracofe, D.: Transparent Proxy with Linux and Squid Mini-HOWTO (Jan. 2002).
21) http://www.apache.org/
22) Braden, R.: T/TCP — TCP Extensions for Transactions Functional Specification, RFC1644 (July 1994).
23) Route(8), Linux Programmer's Manual (Aug. 1997).
24) Rizzo, L., Dummynet: A simple approach to the evaluation of network Protocols, *ACM Computing Communication Review* (Jan.1997).
25) Allman, M., Floyd, S. and Partridge, C.: Increasing TCP's Initial Window, RFC2414 (Sep. 1998).
26) Allman, M.: TCP Congestion Control with Appropriate Byte Counting, RFC3465 (Feb. 2003).
27) Kent, C. and Mogul, J.: Fragmentation Considered Harmful, *Proc. ACM SIGCOMM '87* (Aug. 1987).
28) Padhye, J., Firoiu, V., Towsley, D. and Kurose, J.: Modeling TCP Reno performance: A simple model and its empirical validation, IEEE/ACM Trans. on Networking, Vol.8, No.2, pp.133–145 (Apr. 2000).
29) INTELSAT: Performance Characteristics for Intermediate Data Rate Digital Carriers Using Convolutional Encoding/Viterbi Encoding and QPSK Modulation, Intelsat Earth Station Standard (IESS)-308 (Rev. 11) (Jan. 2003).
30) Stevens, W.: TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms, RFC2001 (Jan. 1997).

(Received May 19, 2005)
(Accepted November 1, 2005)
(Online version of this article can be found in the IPSJ Digital Courier, Vol.2, pp.108–119.)

Teruyuki Hasegawa received the B.E. and M.E. degrees of electrical engineering from Kyoto University, Japan, in 1991 and 1993 respectively. Since joining KDD (now KDDI) in 1993, he has been working in the field of high speed communication protocol and multicast system. He is currently a senior research engineer of IP Communication Quality Lab. in KDDI R&D Laboratories Inc. He received Best Paper Award for Young Researchers of the National Convention of IPSJ in 1996, Best Paper Award of the National Convention of IPSJ in 1999 and 2002, Young Engineer Award of IEICE in 2000, and The Meritorious Award on Radio of ARIB in 2003. He is a member of IEICE.

Toru Hasegawa received the B.E., the M.E. and Dr. Informatics degrees in information engineering from Kyoto University, Japan, in 1982, 1984 and 2000, respectively. Since joining KDD (now KDDI) in 1984, he has been working in the field of formal description technique (FDT) of communication protocols. From 1990 to 1991, he was a visiting researcher at Columbia University. His current interests are Internet measurement and routing protocols. He is currently the executive director of IP Network Division in KDDI R&D Laboratories Inc. He is also a guest professor at National Institute of Informatics. He received IPSJ Convention Award in 1987 and The Meritorious Award on Radio of ARIB in 2003. He is a member of IEICE.