Vol. 40 No. 1 Transactions of Information Processing Society of Japan Jan. 1999

Regular Paper

EFR: Efficient Fast Retransmit Scheme for TCP in a Wireless Multiple Access

YOSUKE TAMURA,† YOSHITO TOBE†† and HIDEYUKI TOKUDA†††

In this paper, we propose a new and efficient fast retransmit scheme for wireless LANs. Wireless LANs are becoming popular, and providing an efficient protocol over a wireless link is important. In our experiments with 2.4 GHz WaveLAN, the poor TCP performance observed was attributed to frequent expiration of the retransmission timer. To avoid unnecessary expiration of the retransmission timer, we propose a scheme in which fast retransmission is performed efficiently. The proposed modification only affects a TCP sender, and our version of TCP with the change is interoperable with existing TCP implementations. The change is especially effective in wireless LAN environments, where we have demonstrated dramatic improvements in throughput, 10–15%, via experiments with 2.4 GHz WaveLAN.

1. Introduction

Transmission Control Protocol (TCP)[6] has been widely used in computer networks to achieve reliable communication. Many popular applications such as web browsing, File Transfer Protocol (FTP)[7], and telnet[17] use TCP as the transport protocol. Recent emergence of wireless local area network (LAN) technology enables these applications to be used in wireless environments as well.

Wireless networking, using AT&T's WaveLAN, has recently been introduced on our university campus, with a resulting change in computer usage. When many students access the wireless network simultaneously, interference among mobile hosts sharing a wireless link causes significant performance degradation when using existing TCP implementations. To minimize the performance degradation of TCP over wireless LANs, a more efficient TCP algorithm and its implementation are needed.

Efficient TCP operation over wireless links is also an important consideration for *ad-hoc networks*[9),16]. In ad-hoc networks, multiple mobile hosts share the same wireless link. The sharing leads to a higher data loss rate due to interference, multi-path phasing, and collision. Efficient TCP implementations suited to wireless LANs are also beneficial in ad-hoc networks.

Ever since the release of BSD version 4.3, TCP implementations include sophisticated congestion control schemes such as *fast retransmit* and *fast recovery*[11]. In addition, many schemes have been introduced to improve TCP performance. For example, *selective acknowledgment* (SACK)[12] and an improved version of this mechanism, *forward acknowledgment* (FACK)[14], have been proposed. These performance improvement schemes have been tuned for wired links[3], and although some studies have examined TCP performance over collections of links, including wireless ones[2),4], we are not aware of any studies that address TCP performance improvements for a single wireless link.

Balakrishnan, *et al.*[3] compared various schemes that proposed improvements to TCP performance over wireless links and concluded that the SMART scheme[13], a modification of the basic SACK approach that combines Go-Back-N, outperforms other schemes. A major drawback of the various SACK schemes is the need to modify the TCP software at both the sender and the receiver. Interoperability with existing schemes is thus not possible.

In this paper, we first present results of a study of the dynamics of TCP windowed flow control in a wireless LAN which identifies frequent expiration of the retransmission timer as a major cause of reduced throughput. We then analyze the cause of the expiration. Since retransmission is essential to provide data reliability, we seek a way of reducing the number of timer expirations. In many cases, timer expiration at the sender occurs because the sender waits for a fixed number of duplicate acknowledgments (ACKs) from the receiver before re-

† Graduate School of Media and Governance, Keio University
†† Keio Research Institute at SFC, Keio University
††† Faculty of Environmental Information, Keio University

transmitting data; the problem is that the receiver does not send as many duplicate ACKs as the sender expects. This situation always occurs when the window size of the TCP sender is small, ranging from 1 to 4. We call such windows "tiny windows". Often, retransmission timeouts can be avoided for tiny windows; those timeouts that unnecessarily occur are referred to as "wasted timeouts". Based on this observation, we propose the following *efficient fast retransmit* (EFR) scheme:

- The sender dynamically estimates the maximum number of duplicate ACKs that it expects to receive.
- When the sender receives this many ACKs, it immediately fast retransmits.

The work presented in this paper proposes a modification to TCP called EFR that improves the performance of TCP over a wireless LAN by more efficiently performing fast retransmission. Our scheme interoperates with existing TCP implementations. Evaluation of this scheme was performed using the wireless LAN. Results show that EFR improves TCP throughput by 10–15% in a WaveLAN environment.

The remainder of the paper is organized as follows. Section 2 addresses issues in TCP congestion control in wireless LAN environments. Section 3 describes related work. Observations of actual TCP behavior in our experimental testbed are presented in Section 4, leading to a discussion of our proposed scheme and its implementation in Section 5. Section 6 presents an evaluation of our scheme. Discussions and future work are presented in Section 7, and we conclude in Section 8.

2. Basic Behaviors in TCP Retransmission

In this section, we briefly review TCP-Reno's retransmission algorithms and see basic behaviors of TCP. This forms the basis of discussions in the following sections.

Let us assume that a TCP sender transmits a sequence of segments, segment i ($i = 1, 2, 3, \ldots$), to a TCP receiver. When segment k is lost and the receiver receives segment $(k+1)$, the receiver sends an ACK requesting segment k. When the receiver receives the following segments, segment $(k+2)$, $(k+3)$, ..., it also sends an ACK requesting segment k. These ACKs are called duplicate ACKs. The sender initiates retransmission of segment k when either of the following conditions is satisfied.

- The number of received duplicate ACKs reaches the threshold value for determining loss of a segment.
- The number of received duplicate ACKs does not reach the threshold value and the retransmission timer expires.

2.1 Retransmission Timer

The retransmission timeout in non-congestion networks can cause an unnecessary degradation of TCP performance. When the TCP sender transmits a segment, the retransmission timer is set. If the segment is not acknowledged, TCP will retransmit the segment. The value of this timer is calculated dynamically, based on the round-trip time measured by the sender. The retransmission timer is bounded to a value between 1 and 64 seconds. Reno's round-trip time and variance estimates are calculated using a coarse-grained timer (around 500 ms), assuming that the round-trip time (RTT) estimate is not accurate. If the retransmission timer expires, TCP uses the Slow-Start algorithm [12]; the TCP sender starts with a congestion window of one segment and increases the congestion window exponentially*.

2.2 Fast Retransmit and Fast Recovery

Based on duplicate ACKs, the fast retransmit algorithm enables earlier detection of missing segments. Since the sender does not know whether a duplicate ACK is caused by a lost segment or merely a reordering of segments, it waits for three consecutive duplicate ACKs to be received [18]. When the sender receives three consecutive duplicate ACKs, it initiates retransmission. The fast recovery is a mechanism which enables the sender to go into congestion avoidance mode instead of slow start mode after the fast retransmit.

These schemes have improved TCP performance since the sender can retransmit the lost segments without waiting for a retransmission timeout. The waiting time for a retransmission timeout is very long for TCP in the order of seconds.

3. Related Work

This section describes the previous work on TCP performance improvements over wireless

* It is not exactly exponential because the receiver may delay its ACKs, typically sending one ACK for every two segments that it receives.

links. We also state the difference between our goals and these works.

As described in the previous section, reducing the occurrence of the retransmission timer's timeouts improves TCP performance. The NewReno[10] TCP is an approach that changes the fast retransmit algorithm that eliminates Reno's waiting for the retransmission timer's timeout when multiple segments are lost within a single window. As described in Ref. 10), loss of multiple segments prohibits invoking fast retransmit of the second lost segment. The NewReno TCP solves this problem by modifying the fast retransmit algorithm.

TCP Vegas[1),5)] also seeks to eliminate unnecessary expiration of the retransmission timer. In TCP Vegas, a fine-grained timer is introduced to calculate RTT more precisely. When a single duplicate ACK is returned to the sender, the sender retransmits a segment without wait for three duplicate ACKs if calculated RTT is greater than the timeout value.

The TCP receiver with SACK options[12)] can inform the sender which segments have been correctly received, when it holds noncontiguous segments. A simulation-based study[8)] investigated the robustness of SACK under several types of segment loss. An advanced version of SACKs, FACK[14)], uses the additional information provided by the SACK option to keep an explicit measurement of the amount of segment outstanding in the network and uses this information to improve congestion control.

Indirect TCP (I-TCP)[2)] and Snoop[3)] Protocols use split-connections. The key idea behind these protocols is to split one TCP connection into multiple pieces in wired and wireless links. Congestion control at the wireless link is done differently from the control at the wired link. Thus, these protocols aim at better performance of the overall TCP connection.

Our goals of designing a scheme are the interoperability between the existing TCP implementation and improving TCP over a single wireless LAN. In this respect, the objectives of I-TCP and Snoop Protocol are different from ours. Although SACKs and related algorithms improve TCP performance, they need cooperation between the sender and the receiver.

The objective of NewReno and Vegas are similar to ours. Although NewReno focuses on loss of multiple segments, we seek a scheme which solves wasted timeouts also for loss of a single segment. Vegas may improve TCP performance over a wireless LAN, but we seek a scheme which does not introduce an extra timer. We compare our scheme with Vegas in Section 7. Like NewReno, we seek problems in behaviors of TCP with an actual wireless LAN and create an algorithm to solve the problems.

4. TCP Behaviors in Wireless Networks

In this section, we examine behavior of the TCP sequence number at a sender in the experiments with 2.4 GHz AT&T WaveLAN. And investigate the cause of retransmission.

4.1 Experimental Methodology

Figure 1 shows the experimental environment which is used throughout the experiments in this paper. In Fig. 1, Host CH, connected to a wired 10BASE-T Ethernet, is an IBM PC-AT compatible computer with 200-MHz Pentium Pro and 32-MByte RAMs. Hosts MH1–MH4 are Toshiba Dynabook SSR-590 with 90-MHz Pentium and 16-MByte RAMs equipped with a PC-Card of 2 Mbps WaveLAN. Hosts MH1–MH4 communicate with Host CH via WavePOINT. The round-trip delay between Hosts MH1–MH4 and Host CH measured with ping program is about 6 ms. In our lab, the segment loss rate in transferring data between a single MH and CH with no other network traffic is maintained at approximately 3.0×10^{-4}. We maintained the loss rate to measure performance under a typical environment without any electromagnetical interference.

FreeBSD 2.2.1-Release which includes TCP-Reno, the current standard TCP implementation, is installed in all hosts. All the experiments noted in this paper use 1440-byte segment size, and the maximum window size for a connection is set to 35 segments.

We measure the throughput at the sender in

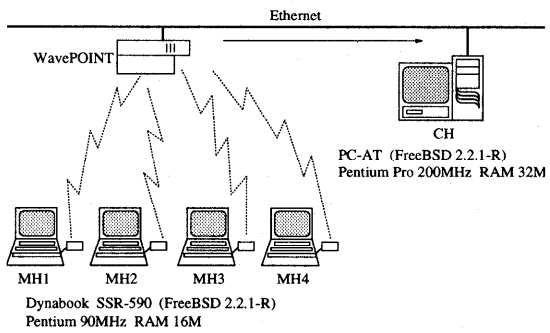


Fig. 1 Our evaluation environment.

the unit of 10^3 bit per second (kbps). To compare our scheme with TCP-Reno, we use the same setup in all hosts. In addition, all information regarding the segment transmissions on the Ethernet and WaveLAN are recorded for analysis using both tcpdump [15] and a Toyo Technica's LAN protocol analyzer "Network-General Sniffer". We have modified the TCP source code to obtain information of TCP Control Block (TCB).

4.2 Overhead of TCP Retransmission

In Fig. 1, Hosts MH1-MH3 continue to transmit data to Host CH through TCP sockets, to provide a sharing link environment common in LANs. Throughout the experiment, MH1-MH3 are the senders, and CH is the receiver. We investigated the behavior of the TCP connection between Host MH1 and CH by conducting ftp for a 1 Mbyte data under this environment.

In Figs. 2–5, we show the dynamics of only one host, MH1, since all hosts have the same state. The measured transition of sequence numbers is shown in **Fig. 2**. In this figure one can see a frequent gap of time where the growth of the sequence number ceases. The gaps are caused by the expiration of either the retransmission timer, or the fast retransmit.

Figure 3 shows the magnified portion of Fig. 2 where retransmission occurs due to timer expiration. We can observe that the sender receives only one ACK corresponding to a lost segment, which results in the timer expiration after approximately 2 sec. Each timer expiration ceases transmission for approximately 2 sec. and all ceased periods noticeable in Fig. 2 are due to the timer expiration.

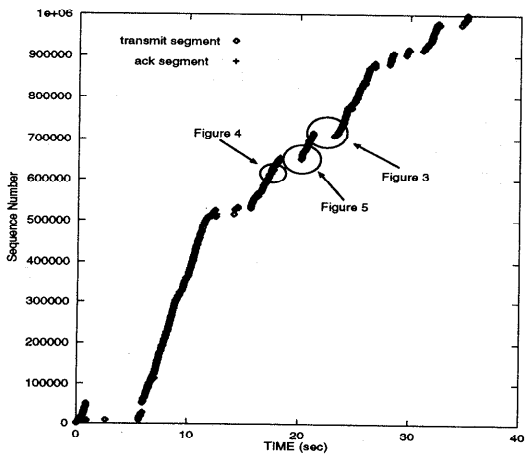


Fig. 2 A 1-Mbyte transfer in a wireless environment with 3 hosts (sequence number vs. time).

Figure 4 shows the magnified portion where retransmission occurs due to the fast retransmit. The fast retransmit invokes the *tcp_output()* function immediately after the sender receives the third duplicate ACK. Transmission of other segments is prioritized over the fast retransmit, but the overhead associated with this fast retransmit is only 0.2 sec. which proves to be substantially smaller than that with the timer expiration.

The difference in overhead between the two is closely related to the retransmit timeout value (RTO) and the number of the timer expiration. The RTO is set to the average RTT plus 4 times its variance. Since the statistics of RTT are measured in 500 ms clock ticks, the typical RTO is set to 2 sec.

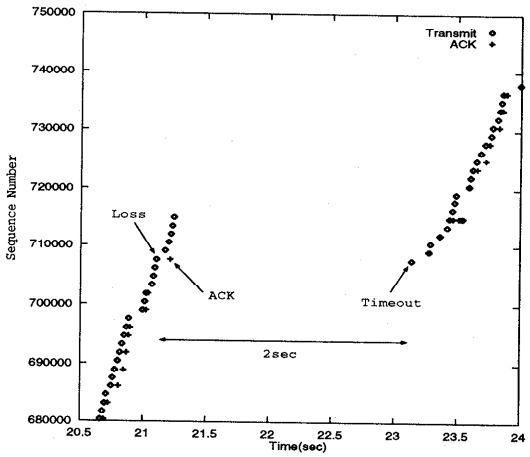


Fig. 3 Retransmission timeout occurs because the sender does not receive any duplicate ACKs.

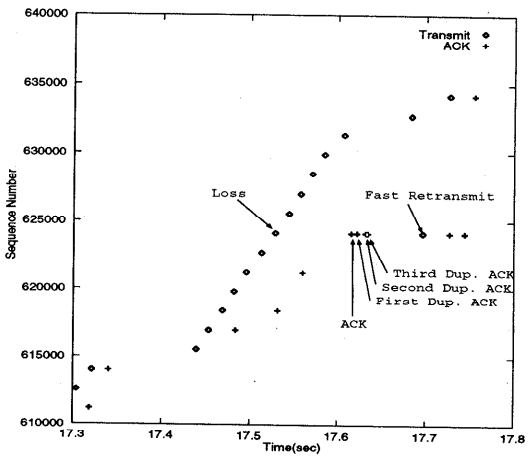


Fig. 4 Fast retransmit is performed after the sender receives the third duplicate ACK.

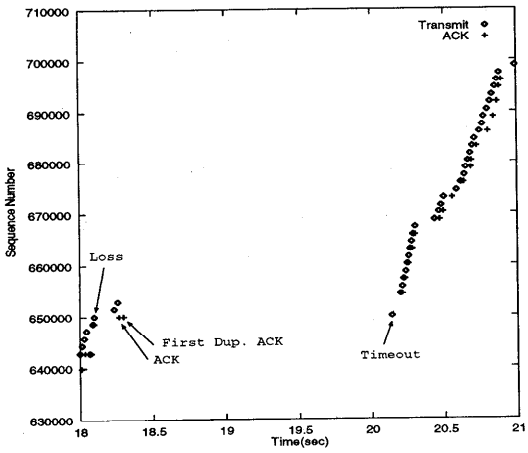


Fig. 5 Retransmission timeout occurs because the sender receives only one duplicate ACK.

4.3 Causes of Retransmission

Let us further investigate what occurs when the retransmission timer expires. **Figure 5** shows an enlarged portion where the retransmission timer expires due to TCP's tiny window. In Fig. 5, a segment corresponding to sequence number 650,000 is lost. We call this segment k. The segments corresponding to sequence number 651,440, 652,800, and 654,320 will be expressed as segments $(k+1)$, $(k+2)$, and $(k+3)$, respectively. In the figure, ACKs of segments $(k+1)$ and $(k+2)$ are returned to the sender, while ACK of segment k is not returned. Hence, segment k is lost. The ACK of segment $(k+2)$ becomes the first duplicate ACK. Since segment $(k+3)$ is not transmitted from the sender because the window size is presumed to be 3, the sender cannot receive the ACK of segment $(k+3)$. Therefore both the sender and the receiver cease transmission that causes the expiration of the retransmission timer. Whenever the window size ranges from 1 to 4, the fast retransmit is never invoked and the expiration of the retransmission timer always happens. If the fast retransmit is performed whenever the sender receives a duplicate ACK of a lost segment, several benefits to the TCP connection will occur; (1) Retransmission of the lost segment with the fast retransmit starts 1.5 seconds earlier than that without the fast retransmit. (2) Unnecessary congestion control is avoided. (3) The throughput is improved by the fast recovery.

4.4 Trace of Windows

Next, we traced the behavior of the sender window. In Fig. 1, Hosts MH1–MH3 continue to transmit data to Host CH through TCP sockets. **Figures 6**, **8**, and **10** show the acquired behavior of sequence number of MH1–MH3. Although only behavior of sequence number at one host is presented in Fig. 2, we can see the gaps are similarly seen in all hosts in these figures.

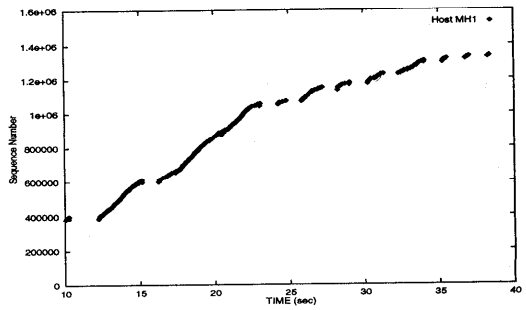


Fig. 6 Behavior of sequence number at MH1.

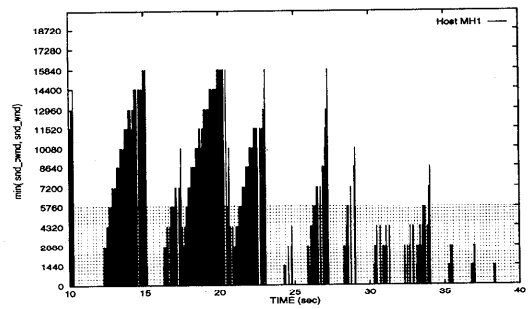


Fig. 7 snd_cur at MH1 over a wireless LAN.

Here, we let *snd_cur* denote the window size. The value of *snd_cur* corresponds to the minimum of *cwnd*, a congestion window, and *snd_wnd*, the receiver's advertised window. **Figures 7**, **9**, and **11** illustrate transition of *snd_cur* for three Hosts, MH1–MH3. The vertical line is expressed in units of 1440 bytes, one segment. Bars are drawn when transmission is done and the value of *snd_cur* is zero when the transmission is not performed. The areas where the value of *snd_cur* is equal to or less than 4 segments are shaded in these figures. The area represents where TCP has a tiny window, and a possible wasted timeout.

As seen in Figs. 7, 9, and 11, *snd_cur* increases as ACKs are successfully returned to Hosts MH1–MH3, but once they fail in receiving ACKs, it is decreased to a small number within the shaded area, The value of *snd_cur* remains less than four segments. To make a comparison with a wired LAN, Hosts MH1–MH3 transmit data to Host CH continuously through TCP sockets with 10BASE-T. **Fig-**

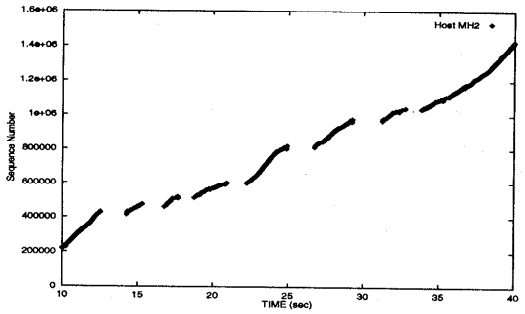


Fig. 8 Behavior of sequence number at MH2.

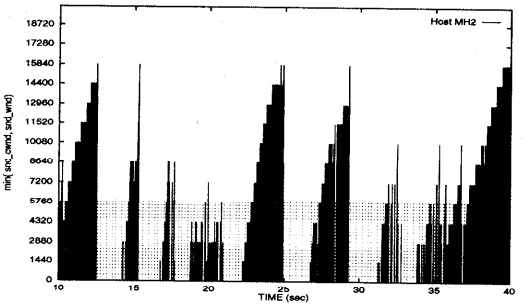


Fig. 9 snd_cur at MH2 over a wireless LAN.

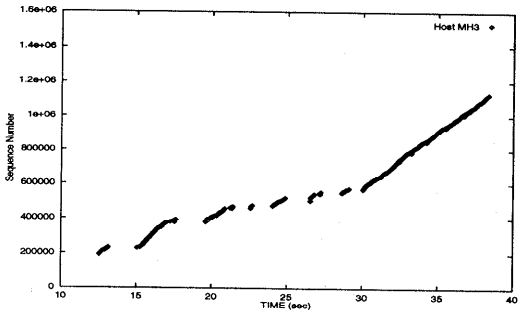


Fig. 10 Behavior of sequence number at MH3.

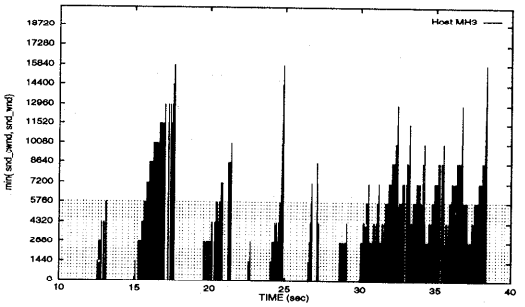


Fig. 11 snd_cur at MH3 over a wireless LAN.

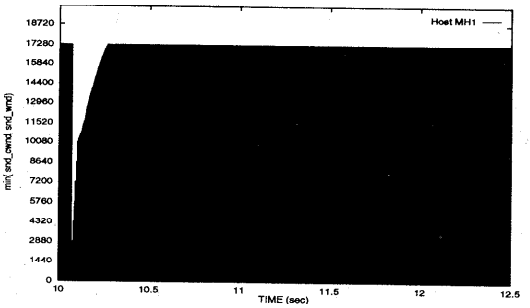


Fig. 12 snd_cur at MH1 over a wired LAN.

ure 12 shows the behavior of *snd_cur* in the wired LAN. Since the values are drawn with bars, most of the *snd_cur* is 17280 bytes. Unlike wireless LAN, the value of *snd_cur* is mostly outside the area. Remaining in the area results in unavailability of receiving three consecutive duplicate ACKs and trigger of wasted timeout. Wasted timeout is likely to occur more in wireless LAN.

5. Efficient Fast Retransmit Scheme

In this section, we propose a modification to the fast retransmit algorithm that comes bundled with current TCP implementations. The fast retransmit algorithm retransmits a segment after a fixed number of duplicate ACKs (three) for the same segment are received. In contrast, our modified algorithm, called EFR (Efficient Fast Retransmit), dynamically calculates *tcprexmtthresh*, the number of consecutive duplicate ACKs that trigger fast retransmit.

We also consider the timing of acknowledgments and the issue of false fast retransmit.

5.1 EFR Overview

The goals of EFR are to be interoperable with existing TCP implementations and to prevent wasted timeouts from occurring. To meet these goals, EFR requires changing only a TCP sender. The sender avoids waiting for duplicate ACKs that the receiver will never transmit and initiates retransmission before the retransmission timer expires.

In EFR, when the sender receives the first duplicate ACK, it calculates *tcprexmtthresh*, the maximum number of duplicate ACKs it can receive before retransmitting the lost segment corresponding to the duplicate ACKs. When the number of duplicate ACKs received at the sender reaches this threshold, the sender retransmits the lost segment. The value of *tcprexmtthresh* is determined based on the window size.

Figure 13 compares period of time before a retransmission occurs between TCP-Reno and EFR when the value of *snd_cur* is four; TCP has a tiny window. In the figure, *t_dupacks* is the number of received duplicate ACKs. In EFR, the sender performs the fast retransmit when

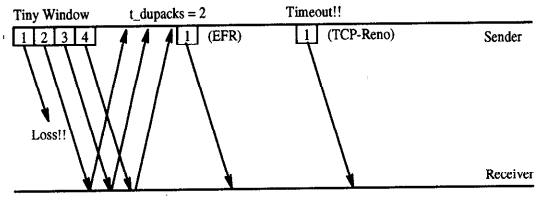


Fig. 13 Period of time before a retransmission occurs when TCP has a tiny window.

it receives the second duplicate ACK, while in TCP-Reno the sender waits until the retransmission timer expires (on the order of seconds) before retransmitting the lost segments. Note that EFR retransmits the lost segment without waiting for the retransmission timer to expire.

5.2 Issue in False Fast Retransmit

TCP operates on top of the Internet Protocol (IP), which does not guarantee reliable packet delivery. In some cases, when segments arrive out of sequence at a receiver, a false fast retransmit may occur. Despite no actual congestion, the fast retransmit algorithm may mistakenly force the sender into congestion avoidance mode. When the TCP sender has a large current window, these false fast retransmissions significantly degrade TCP performance. As *tcprexmtthresh* becomes smaller, the probability of falsely invoking fast retransmit increases. We therefore dictate the policy that *tcprexmtthresh* should not be fixed at one or two.

In our scheme, however, EFR is triggered only when TCP has a tiny current window. In this case, even if false fast retransmit occurs and TCP enters congestion avoidance mode, TCP performance is not substantially affected.

5.3 Consideration of Acknowledgments

TCP does not always send an ACK immediately on receipt of a segment. Instead, normal acknowledgements occur when either a 200 ms ACK timer expires or the window at the receiver has increased by two segments. On the other hand, a receiver sends a duplicate ACK immediately upon receipt of out-of-order segments. We should note here that an ACK of segment $(k-1)$ may not be transmitted to the sender when segment k is lost and the ACK of segment $(k+1)$ is transmitted to the sender.

EFR counts duplicate ACKs to trigger retransmissions. However, it is not always possible to distinguish between the first ACK and subsequent duplicate ACKs. Consider the following example. As shown in Fig. 13, assume that segment 0 is transmitted before segment 1 and suppose that segment 1 is lost. If the ACK of segment 0 is not received by the sender, then the ACK of segment 2 corresponds to the first ACK received by the sender. Then, the ACK of segment 3 becomes the first duplicate ACK. On the other hand, if the ACK of segment 0 is received by the sender, the ACK of segment 2 becomes the first duplicate ACK because the ACK of segment 0 has already requested segment 1. In either case, when the send window is four segments, the sender can send up to segment 4. Therefore the number of duplicate ACKs is 2 and 3, respectively. If we set the threshold value of duplicate ACKs to 2, a false fast retransmit can occur. But if we set the threshold value of duplicate ACKs to 3, wasted timeouts can occur. In designing EFR, we assume that the ACK of segment 0 is not received by the sender because the cost of a false fast retransmit is less than that of wasted timeouts when TCP has a tiny window.

5.4 Implementation of EFR

In this section, we describe our implementation of EFR in detail, using the FreeBSD TCP-Reno source code as our initial starting point. The EFR algorithm is only added to the fast retransmit part of the TCP source code. We add two variables to TCB: *snd_cur* and *tcprexmtthresh*. The variable *tcprexmtthresh* represents the threshold value used to trigger fast retransmit; it is implemented as a global variable. A default value of *tcprexmtthresh* is fixed at three in the FreeBSD TCP-Reno implementation. In EFR however, a separate *tcprexmtthresh* is maintained per connection and dynamically changed, based on the value of *snd_cur*.

Figure 14 presents the EFR algorithm. Each time a TCP sender receives a duplicate ACK, it executes a portion of the EFR algorithm. If the received duplicate ACK is the first for a segment, the minimum of *snd_wnd* and *snd_cwnd* is saved into *snd_cur*. To estimate the number of segments, *snd_cur* is divided by *t_maxseg*, the maximum segment size. We set *tcprexmtthresh* to (*snd_cur*/*t_maxseg*)-2. Recall that our proposed modification only invokes the fast retransmit to recover lost segments when the sender has a tiny window. If *tcprexmtthresh* is more than three, EFR sets this variable to three. Note that we do not claim that the smaller value of *tcprexmtthresh* is desirable, but we need to adjust *tcprexmtthresh* to a more reasonable value when the send window is tiny.

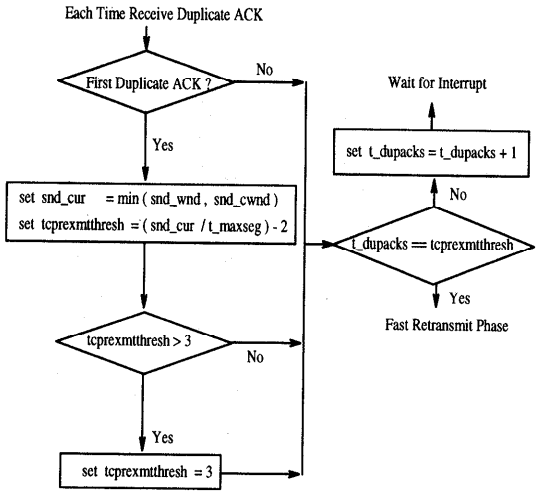


Fig. 14 EFR algorithm.

6. Performance Results

In this section, we present the performance of EFR. The experimental environment, which is shown in Fig. 1, is described in Section 4 in detail. We have compared EFR with TCP-Reno under multiple access environments.

6.1 Retransmission Characteristics

To examine how the number of timeouts is reduced by EFR, we transmitted 10-Mbyte using a UNIX socket-type client-server program. The number of hosts was varied from 2 to 4 by averaging the results of 50 trials in consideration of statistical fluctuations.

Figures 15 and **16** show the number of retransmitted segments and the ratio between the Fast Retransmit and Timeout, respectively. Regardless of the number of hosts, the number of retransmitted segments is reduced approximately by 10% with EFR.

To investigate timeouts further, let $N_{dup}(i)$ represent the number of each occurrence that the number of duplicate ACKs is i. When i is greater than 3, the occurrence is counted in $N_{dup}(3)$. The value of $N_{dup}(3)$ can be positive because the fast retransmitted segment and the following segments may be lost. The values of $N_{dup}(i)$ $(i = 0, 1, 2, 3)$ are shown in **Fig. 17**. When the number of hosts is 2, $N_{dup}(1)$ and $N_{dup}(2)$ are both reduced with EFR from 12 to 8, and from 30 to 7, respectively. It is presumed that this is because wasted timeouts are removed with EFR. The reduction of $N_{dup}(1)$ and $N_{dup}(2)$ results in the reduction of the number of timeouts. The values

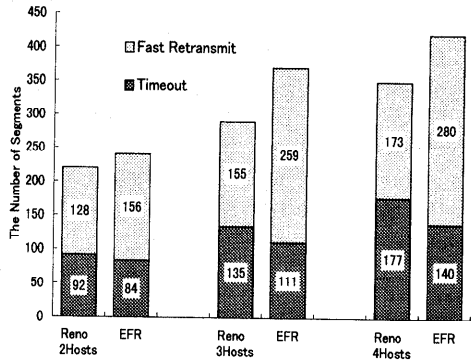


Fig. 15 Number of retransmitted segments (TCP-Reno vs. EFR).

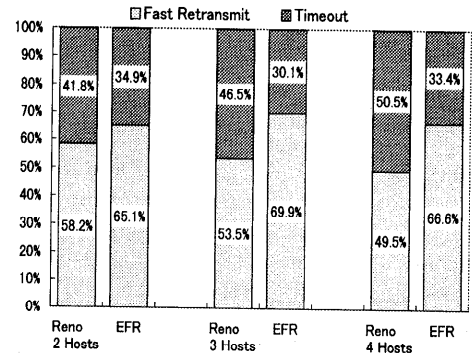


Fig. 16 Ratio between the fast retransmit and timeout (TCP-Reno vs. EFR).

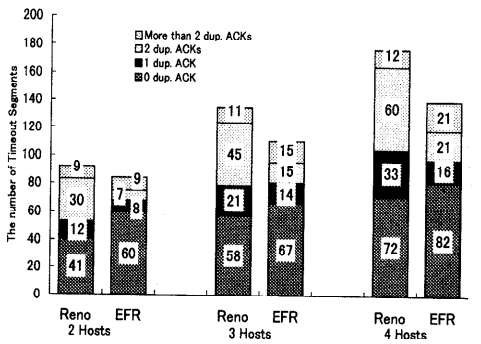


Fig. 17 Number of duplicate ACKs when the retransmission timer expires (TCP-Reno vs. EFR).

of $N_{dup}(1)$ and $N_{dup}(2)$ are not zero because there are cases when *tcprexmtthresh* is 2 and the sender receives only one duplicate ACK, and when *tcprexmtthresh* is 3 and the sender receives either one or two duplicate ACKs. These cases occur when segments following a lost seg-

ment or a duplicate ACK are lost. The value of $N_{dup}(0)$ cannot be reduced because the number of duplicate ACKs is zero when a burst of segments is lost. The value of $N_{dup}(3)$ is counted, for instance, when the segment of the fast retransmit is lost. The timeouts at which the number of duplicate ACKs is more than or equal to three are not wasted timeouts. EFR does not remove such timeouts.

The following point should also be noted. The addition of the number of timeouts and fast retransmit is with EFR. If EFR were to simply change some timeouts to the fast retransmit, the added value would remain the same. A possible explanation for this is that the possibility of segment loss with collision increases with transmission. If the overhead of the timeout is much larger than that of the fast retransmit, reduction of the total loss time is expected. The total loss time of a TCP connection is calculated as follows,

$$T_{loss} = N_{out} \times T_{out} + N_{fast} \times T_{fast},$$

where T_{loss}, T_{out}, T_{fast}, N_{out}, and N_{fast} are the total loss time, the consumed time for timeout, the consumed time for the fast retransmit, the number of timeout, and the number of the fast retransmit, respectively. An evaluation of effectiveness needs to be carried out with the throughput of TCP connections described in the following section. Although we only present a case using two hosts, the same arguments are valid for other cases as well.

6.2 Throughput Measurements

Figure 18 shows the total aggregate throughput through each pair of TCP connections using either EFR and TCP-Reno. The values shown in the figure are obtained by averaging results fifty times. Since a segment loss causes transmission to be interrupted and causes additional messages to be sent, the 2 Mbps bandwidth of WaveLAN is not fully utilized. Yet, EFR contributes to enhancing the effective usage of bandwidth.

Figure 19 shows throughput with 1 Mbytes, 2 Mbytes, 4 Mbytes, and 8 Mbytes of data. It was obtained by averaging results fifty times. The bars indicate the averaged values and the straight lines associated with the bars show the range of measured values. As the data size decreases, the variance of the throughput spreads over wider range. These figures indicate that EFR has higher performance than TCP-Reno under a multiple hosts environment. In the case of 8 Mbyte data transfer with three and four

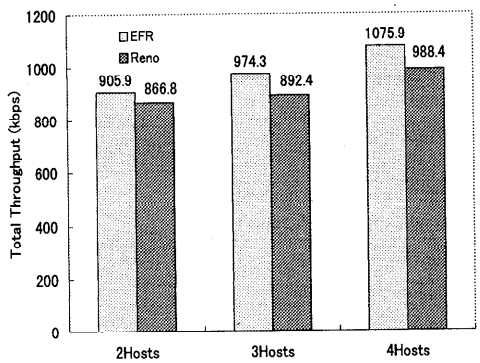


Fig. 18 Total throughput of all hosts.

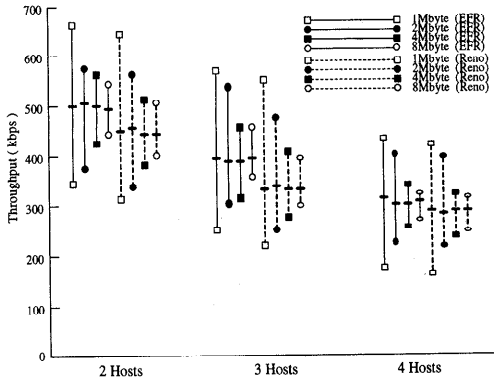


Fig. 19 Comparison of throughput.

hosts, the throughputs with EFR are 14.4% and 12.2% higher than that with TCP-Reno, respectively. These results demonstrate that EFR attains better performance than TCP-Reno in a shared wireless link with multiple hosts.

7. Discussions and Future Works

In this section, we compare EFR with TCP Vegas and discuss applicability of EFR to various types of networks. At the end of this section, we address our future works.

7.1 EFR vs. TCP Vegas

EFR and TCP Vegas's retransmission approach are similar in that both attempt to reduce wasted timeouts. They differ in their mechanisms. EFR is based on the send window, *snd_cur*, while TCP Vegas uses RTT estimation with its fine-grained timer. Unlike RTT, the send window does not depend on the amount of traffic. Therefore we can eliminate wasted timeouts easily with EFR.

7.2 EFR Performance

We have only investigated EFR over wireless links because our motivation was in improve-

ments in a wireless LAN in this paper. EFR is designed, however, to enhance TCP throughput when the current window size is small and the TCP sender cannot push enough segments to receive three duplicate ACKs. In wired networks, the same situation may happen, for instance, at the beginning of slow-start mode after the expiration of the retransmission timer or the establishment of a TCP connection. Therefore EFR can be also used to improve such situation in wired networks.

7.3 Wired LAN Environments

A wired LAN has a lower error rate and a smaller round-trip delay compared with a wireless LAN. Unlike Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) used in a wireless LAN, Carrier Sense Multiple Access with Collision Detection (CSMA/CD) deployed in a wired Ethernet supports detection of collision and retransmission of collided frames at the link layer with limitation in the number of retries. Therefore the multiple access over wired Ethernet does not induce TCP segment loss as seriously as that over a wireless link except a case where retransmission at the link layer fails frequently in overloaded traffic. Thus, the size of the congestion window increases smoothly and the window size is so large that EFR algorithm is not operated in the wired LAN. As a result, EFR offers almost the same performance as a TCP-Reno's fast retransmit with some exceptions with segment retransmission at the beginning of slow-start mode after the expiration of the retransmission timer or the establishment of a TCP connection.

We emphasize that EFR offers performance not worse than TCP-Reno even in wired LANs because an algorithm cannot be accepted when the performance is improved in a domain and deteriorates in another with it.

7.4 WAN Environments

TCP connections in WAN environments, such as these for access to World Wide Web servers, may not be able to transfer data smoothly due to packet losses over the connections with a long delay. In this case, for instance, Web server's TCP window is kept small due to congestion control. In addition, RTO becomes long. Therefore, EFR can be performed in WAN Environments when TCP waits for non-existing duplicate ACKs.

Congestion in routers leads to bursty segment losses. Since EFR does not work in the multiple losses within single window, EFR is not an obstacle to congestion control which will bring about a network collapse.

7.5 Future Works

Although we have evaluated our algorithm in an experimental system, evaluation in a larger real wireless environment remains for our future work.

Evaluation of extensive feasibility of our algorithm to long delay networks which include end-to-end wired link is also our future work.

We are planning to further evaluate EFR scheme combined with NewReno to investigate improvements in the occurrence of multiple segments loss. Evaluation of such a combined scheme in our operating wireless network should be done based on various mobile applications.

8. Summary

We proposed a new EFR scheme for improving TCP performance over wireless LANs. Since TCP is becoming popular over wireless as well as wired LANs, TCP should run efficiently also on lossy wireless multiple access networks.

EFR is based on the clever determination of the timing of the fast retransmit. It allows a TCP sender to adjust the timing of retransmission dynamically depending on the current window size; the TCP sender is able to avoid a duplicate ACK which the receiver never transmits and initiate retransmission before the retransmission timer expires. EFR is compatible with existing TCP implementations.

We implemented our proposed algorithm on FreeBSD-2.2.1R. Evaluation with multiple hosts under 2.4 GHz WaveLAN as is common in actual environments demonstrated significant improvements in throughput, 10–15%, compared with TCP-Reno.

Acknowledgments The authors are grateful to members of the MKng/R3 Project for valuable comments and criticisms, especially Hiroshi Inamura supplied ideas for extending this research.

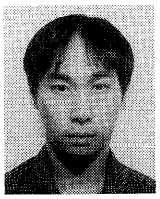
We thank Kevin Fall, Ion Stoica, Hari Balakrishnan and the anonymous reviewers for several comments and suggestions that helped improve the quality of this paper. Our special thanks to Yasunori Matsui, Richard Buskens, and Stephen Chou, who gave us valuable comments and suggestions on a draft of this paper.

References

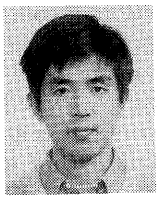
1) Ahn, J.S., Danzig, P.B., Liu, Z. and Yan, L.:

Evaluation of TCP Vegas: Emulation and Experiment, *Proc. ACM SIGCOMM '95* (1995).
2) Bakre, A. and Badrinath, B.R.: I-TCP: Indirect TCP for Mobile Hosts, *Proc. ICDCS '95* (1995).
3) Balakrishnan, H., Padmanabhan, V.N., Seshan, S. and Katz, R.H.: A Comparison of Mechanisms for Improving TCP Performance over Wireless Links, *Proc. ACM SIGCOMM '96* (1996).
4) Balakrishnan, H., Seshan, S. and Katz, R.H.: A Improving TCP/IP Performance over Wireless Networks, *Proc. ACM MOBICOM '95* (1995).
5) Brakmo, L.S., O'Malley, S.W. and Peterson, L.L.: TCP Vegas: New Techniques for Congestion Detection and Avoidance, *Proc. ACM SIGCOMM '94* (1994).
6) Postel, J. (Ed.): Transmission Control Protocol – DARPA Internet Program Protocol Specification, RFC 793 (1981).
7) Postel, J. (Ed.): FILE TRANSFER PROTOCOL, RFC 765 (1980).
8) Fall, K. and Floyd, S.: Simulation-based Comparisons of Taho, Reno, and SACK TCP, *ACM SIGCOMM*, 26(3) (1996).
9) Hodes, T.D., Katz, R.H., Schreiber, E.S. and Rowe, L.: Composable Ad-hoc Mobile Services for Universal Interaction, *Proc. ACM MOBICOM '97* (1997).
10) Hoe, J.C.: Improving the Start-up Behavior of a Congestion Control Scheme for TCP, *Proc. ACM SIGCOMM '96* (1996).
11) Jacobson, V.: Congestion Avoidance and Control, *Proc. ACM SIGCOMM '88* (1988).
12) Jacobson, V. and Branden, R.: TCP Extensions for Long-Delay Paths, RFC 1072 (1988).
13) Keshav, S. and Morgan, S.P.: SMART Retransmission: Performance with Random Losses and Overload, *Proc. IEEE INFOCOM '97* (1997).
14) Mathis, M. and Mahdavi, J.: Forward Acknowledgment: Refinding TCP Congestion Control, *Proc. ACM SIGCOMM '96* (1996).
15) McCanne, S. and Jacobson, V.: The BSD Packet Filter: A New Architecture for User-Level Packet Capture, *Proc. Winter '93 USENIX Conference* (1993).
16) Park, V.D. and Corson, M.S.: A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks, *Proc. IEEE INFOCOM '97* (1997).
17) Postel, J. and Reynolds, J.: TELNET PROTOCOL SPECIFICATION, RFC 854 (1983).
18) Stevens, W.R.: *TCP/IP Illustrated, Volume 1: The Protocols*, Addison-Wesley (1994).

(Received May 6, 1998)
(Accepted September 7, 1998)



Yosuke Tamura received his B.S. degree in environmental information from Keio University in 1997. He is a graduate school of Media Governance, Keio University. He is currently studying wireless TCP/IP networks. He is a member of the Information Processing Society of Japan.



Yoshito Tobe received his B.E. and M.E. degrees in electrical engineering from the University of Tokyo in 1984 and 1986, respectively. He received M.S. degree in Electrical and Computer Engineering from Carnegie Mellon University in 1992. He is a research staff at Keio Research Institute at SFC, where he is currently studying QoS-aware protocols. He is a member of IEEE Communications Society, ACM, and the Information Processing Society of Japan.

Hideyuki Tokuda received his B.S. and M.S. degrees in electrical engineering from Keio University in 1975 and 1977, respectively; the Ph.D. degree in computer science from the University of Waterloo in 1983. He joined the School of Computer Science at Carnegie Mellon University in 1983, and is an Adjunct Associate Professor from 1994. He joined the Faculty of Environmental Information at Keio University in 1990, and is a Professor since 1996. His current interests include distributed operating systems and computer networks.