IEICE TRANS. COMMUN., VOL.E90–B, NO.7 JULY 2007

PAPER

Modeling TCP Throughput over Wired/Wireless Heterogeneous Networks for Receiver-Based ACK Splitting Mechanism

Go HASEGAWA[†a)], ***Member***, **Masashi NAKATA**[†], ***Nonmember***, ***and*** **Hirotaka NAKANO**[†], ***Member***

SUMMARY The performance of TCP data transmission deteriorates significantly when a TCP connection traverses a heterogeneous network consisting of wired and wireless links. This is mainly because of packet losses caused by the high bit error rate of wireless links. We proposed receiver-based ACK splitting mechanism in [1]. It is a new mechanism to improve the performance of TCP over wired and wireless heterogeneous networks. Our mechanism employs a receiver-based approach, which does not need modifications to be made to the sender TCP or the base station. It uses the ACK-splitting method for increasing the congestion window size quickly in order to restrain the throughput degradation caused by packet losses due to the high bit error rate of wireless links. In this paper, we develop a mathematical analysis method to derive the throughput of a TCP connection, with/without our mechanism, which traverses wired and wireless heterogeneous networks. By using the analysis results, we evaluate the effectiveness of our mechanism in the network where both of packet losses due to network congestion and those caused by the high bit error rate of wireless links take place. Through An evaluation of the proposed method shows that it can give a good estimation of TCP throughput under the mixture networks of wired/wireless links. We also find that the larger the bandwidth of the wireless link is, the more effective our mechanism becomes, therefore, the mechanism's usability will increase in the future as wireless networks become faster.

key words: *TCP, wired/wireless heterogeneous networks, ACK splitting mechanism, throughput analysis*

1. Introduction

Various wireless network technologies have become popular in recent years and are being used as access networks to the Internet. In particular, heterogeneous networks with wired and wireless links have become common. However, it is a well-known problem that the performance of TCP deteriorates significantly when a TCP connection traverses such networks [2]. This is because TCP can not distinguish the cause of packet loss: whether a *wireless loss* which is caused by the bit error on wireless links or a *congestion loss* which is caused by the network congestion. That is, the TCP sender reduces the congestion window size to half in response to all packet losses, meaning that the transmission rate is slowed down unnecessarily when wireless losses take place.

Many solutions to this problem have been proposed in the past literature [3]–[6]. Some of them [3], [4] modify the functions of the base station. They aim to hide the occurrence of wireless losses from the TCP connection in the wired network. However, such solutions violate TCP's end-to-end principle because they split the TCP connection at the base station. Furthermore, they can not be applied when a lower-layer encryption mechanism such as IPSec is utilized because they need to access TCP header at the base station. Other solutions [5], [6] modify the sender-side TCP algorithm. They try not to slow down TCP's transmission rate when a wireless loss is detected. They preserve the end-to-end principle and can be applied when the traffic is encrypted at a lower-layer. However, they face another difficulty in their deployment path. That is, the TCP sender is generally the server in the wired network, and server administrators do not prefer solutions that introduce additional costs or instability to their systems.

Because of the above drawbacks, most of the existing solutions are not widely deployed. So, in [1], we have proposed a receiver-based ACK splitting mechanism to improve TCP throughput over wired and wireless heterogeneous networks without such drawbacks. Our mechanism only requires one to make a modification to the TCP algorithm at the receiver host that directly connects to the wireless access network. Therefore, it preserves TCP's end-to-end principle, and it can easily be deployed and applied to IP-level encrypted traffic. To restrain throughput degradation only with the modification of the receiver-side TCP algorithm, our mechanism uses ACK-splitting method [7]. In [1], we exhibited the effectiveness of the proposed mechanism by simulation experiments.

In this paper, we develop a mathematical method to derive the throughput of a TCP connection with/without our mechanism which traverses wired and wireless heterogeneous networks. In the analysis, we assume that both of packet losses take place due to network congestion and the high bit error rate of wireless links. Then, through analysis results, we discuss the usability of our mechanism in various (including future) wireless network environments.

The rest of this paper is organized as follows. In Sect. 2, we describe our receiver-based ACK splitting mechanism. In Sect. 3, we explain the details of analysis of our mechanism's throughput. In Sect. 4, we present numerical results of our mechanism and discuss its usability. Section 5 concludes this paper and offers an outline for future work on this topic.

Manuscript received May 19, 2006.
Manuscript revised December 3, 2006.

[†]The authors are with the Graduate School of Infomation Science and Technology, Osaka University, Toyonaka-shi, 560-0043 Japan.
a) E-mail: hasegawa@cmc.osaka-u.ac.jp

DOI: 10.1093/ietcom/e90–b.7.1682

2. Receiver-Based ACK Splitting Mechanism

Our mechanism requires a modification only to a TCP receiver connected to a wireless link in order to be easily deployed and to be applicable to IP level encrypted traffic, and to preserve end-to-end principle. That is, our mechanism doesn't change the congestion control mechanism of the sender to prevent the congestion window size from being reduced in response wireless losses. Instead, it quickly increases the congestion window size by using the ACK-splitting method, which is a method to increase the congestion window size of the sender-side TCP quickly than usual by sending multiple ACKnowledgement (ACK) packets when a data packet arrives at the receiver.

However, since ACK-splitting has some demerits when it is used inappropriately, it is necessary to control its execution. Congestion losses are a sign of network congestion, and halving the congestion window size is the correct way to avoid further network congestion. Consequently, ACK-splitting should not be executed in response to congestion losses because enlarging the congestion window size ends up increasing the network congestion. To avoid such a situation, our mechanism has a function to distinguish wireless losses from congestion losses at a receiver host.

After a packet loss is distinguished as wireless loss, ACK-splitting begins. However, executing ACK-splitting too long after the wireless loss occurs would increase the load of the sender-side TCP or networks by sending too many ACK packets and by making the congestion window size larger than expected. Thus, we need to control the duration of ACK-splitting to avoid such a situation. Because our purpose is to recover from unnecessary reductions in the congestion window size, our mechanism continues ACK-splitting until the congestion window size reaches the value just before the wireless loss occurred.

Furthermore, during ACK-splitting, if the sending rate of split ACK packets is too high, the amount of data transmitted on the return path to the sender increases excessively. Consequently, the uplink of the wireless network becomes congested, because the uplink bandwidth of the wireless networks is usually small. Thus, our mechanism incorporates a function to control dynamically the sending rate of split ACK packets. The sending rate of split ACK packets is tuned by changing their number for one data packet. We determine the number of split ACK packets for one data packet by estimating the congestion level of the uplink by referring to the length of the sending queue for the uplink network interface at the receiver. According to these procedures, the congestion window size changes as depicted in Fig. 1.

The proposed mechanism may be regarded as vulnerabe attacks by TCP sender. As we know, the proposed mechanism is useful for TCP sender of FreeBSD and Solaris, but Linux 2.2 or later versions has already equipped with the byte-counting TCP mechanism against the vulnerabe attacks [7]. However, since we believe the advantage of the proposed mechanism as shown in [1] and the present paper, we would propose that TCP sender should react the splitted ACK packets, with the appropriate mechanisms for distinguishing ACK-splitting mechanisms and vulnerabe attacks, which is one of important research topics as our future work. The investigation of the effectiveness of the proposed mechanism for TCP sender of other OSes, especially variants of Microsoft Windows, would be also one of our future works.

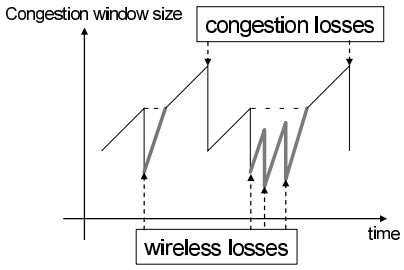

Fig. 1 Expected changes in congestion window size with our mechanism.

3. Analysis

This section presents a mathematical analysis of an average throughput of a TCP connection with/without the proposed mechanism, under wired/wireless heterogeneous network. The analysis is then used to discuss the usability of the mechanism in various wireless network environments.

The analysis is based on the mathematical method reported in [8]. In [8], the throughput of a long-lived TCP connection over a wired network is calculated by modeling TCP's congestion avoidance behavior on the assumption that packet losses occur with a constant ratio[†]. It models the number of packets transmitted during the period between packet loss indications (Fig. 2). The period is called the Triple Duplicate Period (*TDP*). In Fig. 2, a *round* indicates how many RTTs have elapsed from the beginning of the TDP. The average number of packets transmitted in one TDP and the average time length of one TDP are denoted as Y [pkts] and A [sec], respectively. The average throughput of a TCP connection, B [pkts/sec], is calculated as follows:

$$B = \frac{Y}{A} \tag{1}$$

In this paper, we expand the analysis in [8] to obtain the average throughput of a long-lived TCP connection using our mechanism in a heterogeneous network. The following assumptions from [8] will be used: an ACK packet for first data packet in RTT returns to the sender after all data packets in the RTT have been sent out from the sender, and the congestion window size is not limited by the receiver's advertised flow control window. We assume that our mechanism can distinguish the cause of packet loss (congestion loss or wireless loss) perfectly, meaning that we use inter-layer approach for packet loss type distinction (which corresponds

[†]The authors assumed that all of the packet losses are caused by network congestion.

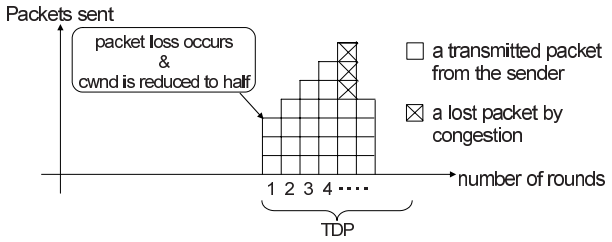

Fig. 2 Model of the number of packets transmitted in a TDP in [8].

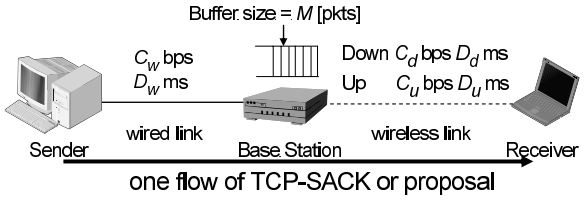

Fig. 3 Network model for the throughput analysis.

to the MAC-method described in [1]).

The network model is depicted in Fig. 3. We define C_w [bps] to be the bandwidth and D_w [s] to be the propagation delay of the wired link. C_d [bps] and D_d [s] are the bandwidth and the propagation delay of the downlink of the wireless network, and C_u [bps] and D_u [s] are those of the uplink of the wireless network. The buffer size of the base station is denoted as M [pkts]. We assume that there is only one persistent TCP connection between the sender and the receiver.

We consider two types of packet loss: wireless loss and congestion loss. We assume that wireless losses take place in the wireless network with a constant ratio p. p is calculated as

$$p = \left\{1 - (1 - e)^S\right\}^4 \tag{2}$$

where e is the bit error rate of the wireless link and S [bits] is packet size. This is based on the assumption that wireless loss occurs after the first transmission and three successive retransmissions of the packet by ARQ fail. Here, we do not assume any Forward Error Correction (FEC). However, it is easy to consider the effect of FEC in the analysis, because all we need to do is change Eq. (2) to decrease p, and the following analysis remains unchanged.

Congestion loss is assumed to occur when the congestion window size reaches the value of W_M [pkts]. W_M is calculated as

$$W_M = \frac{C_d}{S} \times RTT_{min} + M \tag{3}$$

where RTT_{min} [s] is the minimum RTT of the transmission path, which is calculated as follows:

$$RTT_{min} = 2D_w + D_d + D_u + \frac{S + 320}{C_w} + \frac{S}{C_d} + \frac{320}{C_u} \tag{4}$$

where 320 is the size of an ACK packet in bits and we omit the processing delay at receiver host and the base station.

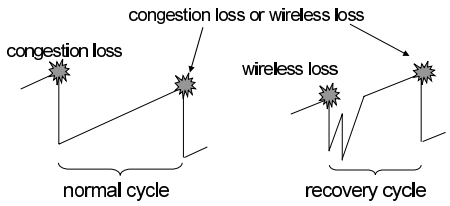

Fig. 4 Changes in congestion window size in the normal and recovery cycles.

Equation (3) is based on the assumption that buffer overflow takes place when the congestion window size becomes larger than the sum of the bandwidth delay product of the network and the buffer size at the base station. Equation (4) is derived from the sum of propagation delays and transmission delays on the round-trip transmission path.

In this paper, we define the period between two packet loss indications as a *cycle* (corresponding to a TDP in [8]). We categorize a cycle into two types, since the congestion window size behaves differently according to whether the detected packet loss is a wireless loss or a congestion loss. We denote the cycle which starts just after a congestion loss as the *normal cycle*, and the cycle which starts just after a wireless loss as the *recovery cycle*. The normal cycle continues until the next packet loss occurs. On the other hand, the recovery cycle continues until the congestion window regains its initial size before the wireless loss by ACK-splitting and packet loss occurs after that. Figure 4 depicts typical changes in the congestion window size in normal and recovery cycles. We assume that a TCP connection uses TCP-SACK [9] and it is always in the congestion avoidance phase in both cycles. Therefore, in the normal cycle, because ACK-splitting is not executed, the speed of the congestion window increase is identical to that of the original TCP-Reno: 1 segment every RTT. On the other hand, in the recovery cycle, the speed of the congestion window size increase is larger than 1 segment per RTT (by using ACK-splitting), and it becomes 1 segment per RTT after the congestion window size reaches the value it had just before the wireless loss occurred.

In the analysis, we first derive the average number of transmitted packets and the average durations of the normal cycle and the recovery cycle. We also calculate the probability for which a TCP connection with our mechanism is in each cycle. We then get the average throughput by averaging these values. Sects. 3.1 and 3.2 explain the analyses of the cycles. In Sect. 3.3, we derive the average RTT of a TCP connection, which is needed to derive the duration of cycles, and in Sect. 3.4, we show how to get the average throughput from them.

3.1 Analysis of Normal Cycle

The analysis for the normal cycle is almost the same as that of [8]. Figure 5 depicts typical changes in the number of transmitted packets in each round of the normal cycle. Since the normal cycle starts just after the congestion loss occurs,

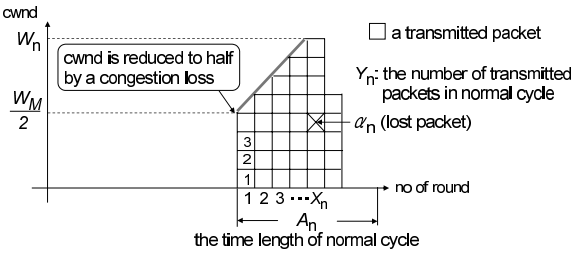

Fig. 5 Change in the number of packets transmitted in the normal cycle.

the congestion window size at the beginning of the normal cycle is $\frac{W_M}{2}$, and $\frac{W_M}{2}$ packets are transmitted from the sender in the first RTT. The number of transmitted packets in round increases by 1 segment every RTT. The normal cycle continues until wireless or congestion loss occurs. We assume that congestion loss takes place when the window size reaches W_M, which is calculated with Eq. (3). That is, the normal cycle is terminated when wireless loss occurs before the congestion window size reaches W_M or when the congestion window size reaches W_M and congestion loss occurs. Here, we denote the number of transmitted packets from the beginning of the normal cycle to when the congestion window size reaches W_M as s_n [pkts]. s_n includes the lost packet, and it is calculated as the sum of an arithmetic progression as follows:

$$s_n = \frac{3\,W_M(W_M+2)}{8} \tag{5}$$

If wireless loss occurs before s_n packets are transmitted from the beginning of the normal cycle, the normal cycle is terminated. On the other hand, if no wireless loss occurs while s_n packets are being transmitted, the loss must be congestion loss.

In Fig. 5, we assume that the α_n-th transmitted packet from the beginning of the cycle is lost, and define W_n [pkts] and X_n as the congestion window size and the round number when the packet loss occurred, respectively. Then, using the wireless loss probability (Eq. (2)), α_n can be calculated as follows:

$$\begin{aligned}\alpha_n &= \sum_{k=1}^{s_n-1}(1-p)^{k-1}pk + \sum_{k=s_n}^{\infty}(1-p)^{k-1}ps_n \\ &= \frac{1-(1-p)^{s_n}}{p}\end{aligned} \tag{6}$$

From Fig. 5, we can derive Y_n [pkts], which is the number of packets transmitted in the normal cycle, in two different ways. The sender detects a packet loss when three duplicate ACK packets arrive, so the packet loss is detected one RTT after the sender sent the packet that became lost. During the RTT, the sender sends additional packets corresponding to the ACK packets that were received before receiving three duplicate ACK packets; therefore, Y_n is calculated as

$$Y_n = \alpha_n + W_n + 1 \tag{7}$$

On the other hand, the congestion window size is incremented by 1 segment per round, so $\frac{W_M}{2} + k - 1$ packets are transmitted in the k-th round. By approximating that $\frac{W_n}{2}$ packets are transmitted in the $(X_n + 1)$-th round on an average [8], Y_n can also be obtained as

$$Y_n = \sum_{k=1}^{X_n}\left(\frac{W_M}{2}+k-1\right) + \frac{W_n}{2} \tag{8}$$

Here, there is a clear relation between W_n and X_n:

$$W_n = \frac{W_M}{2} + X_n - 1 \tag{9}$$

From Eqs. (7), (8), and (9), we can obtain W_n and X_n as follows:

$$W_n = \frac{\sqrt{W_M^2 - 2\,W_M + 8\alpha_n + 8}}{2} \tag{10}$$

$$X_n = \frac{2 - W_M}{2} + W_n \tag{11}$$

Y_n can be obtained by substituting Eqs. (6) and (10) to Eq. (7) as follows:

$$Y_n = \frac{1-(1-p)^{s_n}}{p} + \frac{\sqrt{W_M^2 - 2\,W_M + 8\alpha_n + 8}}{2} + 1 \tag{12}$$

Meanwhile, A_n [s], which is the average duration of the normal cycle, is obtained by taking the product of average RTT and the number of rounds in the normal cycle, considering TCP's timeout after a packet loss occurs:

$$A_n = (X_n + 1)RTT_n + A_{to} \tag{13}$$

where RTT_n [s] is average RTT in the normal cycle, and A_{to} is average duration added when timeout occurs. We assume that timeout takes place only when a retransmitted packet is lost since we use TCP-SACK as the basis of our mechanism. Thus A_{to} is calculated as follows:

$$A_{to} = \frac{pRTO}{1-p}(1+p+2p^2+4p^3+8p^4+16p^5+32p^6) \tag{14}$$

where RTO [s] is TCP's initial retransmission timeout. The derivation of RTT_n is shown in Sect. 3.3.

3.2 Analysis of Recovery Cycle

In the recovery cycle, if wireless loss occurs during ACK-splitting, the congestion window size is reduced. After the reduction, ACK-splitting starts again, and it continues until the congestion window size reaches the value it had just before the beginning of the recovery cycle. Therefore, the length of the recovery cycle varies according to how many wireless losses occur during ACK-splitting, as depicted in Fig. 6. We define the *recovery step* as the period which starts from the beginning of ACK-splitting to when the packet loss takes place (Fig. 6). The number of recovery steps indicates how many times ACK-splitting starts in the recovery cycle. Figure 6 depicts the changes in the congestion window size in recovery cycles with 1, 2, and 3 recovery steps. Apparently from this figure, in the recovery cycle with r recovery

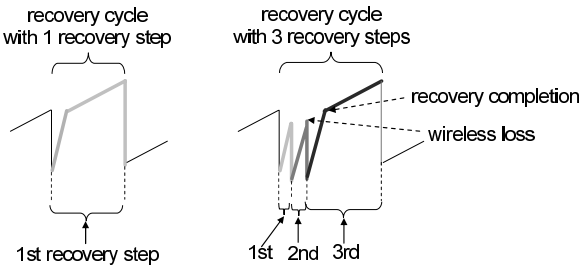

Fig. 6 Behavior of ACK-splitting in the recovery cycle.

steps, first $(r-1)$ recovery steps are terminated by wireless losses during ACK-splitting, and the final r-th recovery step is terminated by a packet loss after the completion of ACK-splitting. Therefore, the form of the last recovery step of the recovery cycle is different from that of other recovery steps in Fig. 6.

Here, we assume that the congestion window size increases by δ segments per 1 round during ACK-splitting, where δ means the average number of ACK packets for one data packet during ACK-splitting. Under the assumption that data packets arrive at the receiver with a constant interval according to the bandwidth of the downlink of the wireless network, δ can be calculated from the packet size and the bandwidth of the downlink and the uplink of the wireless network as follows:

$$\frac{C_u}{320} \div \frac{C_d}{S} = \frac{SC_u}{320C_d} \tag{15}$$

where $\frac{C_u}{320}$ is the number of ACK packets which can be sent per second, and $\frac{C_d}{S}$ is the number of data packets which arrive per second. We assume that the arrival interval of data packets is $\frac{S}{C_d}$. However, it becomes small when a data packet with bit errors is retransmitted by ARQ, because such a data packet and the ones following it are passed together to TCP from the MAC layer after it has been successfully retransmitted. Then, our mechanism sends only one ACK packet for these data packets. That is, the number of ACK packets for these data packets decreases by $(\frac{SC_u}{320C_d} - 1)$. Here, we denote the probability that a bit error occurs in a data packet and that the $(i-1)$ retransmission of the packet fails and the i-th retransmission succeeds as e_i $(1 \le i \le 3)$, and the probability that the bit error occurs and retransmission succeeds as E. They are calculated as follows:

$$e_1 = \{1-(1-e)^S\}(1-e)^S \tag{16}$$

$$e_2 = \{1-(1-e)^S\}^2(1-e)^S \tag{17}$$

$$e_3 = \{1-(1-e)^S\}^3(1-e)^S \tag{18}$$

$$E = e_1 + e_2 + e_3 \tag{19}$$

We calculate the average number of data packets that arrive at the receiver from when the bit error in a packet is detected on its first transmission to when that packet is successfully retransmitted, L [pkts], as follows:

$$L = \frac{e_1 + 2e_2 + 3e_3}{E}\left(D_u + D_d + \frac{S}{C_d}\right) \div \frac{S}{C_d} \tag{20}$$

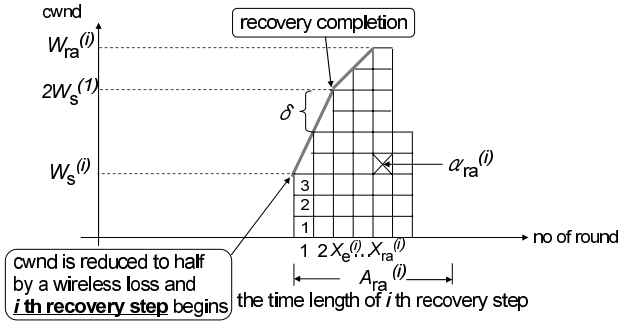

Fig. 7 Model of the number of packets transmitted in i-th recovery step where $i = r$.

If the bit error does not occur in L packets, L packets are passed together with the first erroneous data packet to the TCP layer from the MAC layer; therefore, the arrival interval of L packets is nearly 0. However, if the bit error occurs in the k-th packet among L packets, the packets following the k-th one wait until the k-th one is successfully retransmitted, so the number of data packets which are passed together with the first erroneous data packet from the MAC layer to the TCP layer is $k-1$. The probability that a bit error occurs in the k-th packet after $k-1$ packets have been successfully transmitted is $(1-E)^{k-1}E$. Thus, the average number of data packets that are passed together from the MAC layer to the TCP layer when the bit error occurs is

$$(1-E)E + 2(1-E)^2E + \cdots + (L-1)(1-E)^{L-1}E + L(1-E)^L = \frac{1-E-(1-E)^{L+1}}{E} \tag{21}$$

As a result, δ is

$$\delta = \frac{SC_u}{320C_d} - E\left(\frac{SC_u}{320C_d} - 1\right)\frac{1-E-(1-E)^{L+1}}{E} \tag{22}$$

(1) $i = r$ **Case**

We first analyze the i-th recovery step where $i = r$, i.e., the recovery step that is terminated by a packet loss after ACK-splitting finishes. Figure 7 depicts the model of the number of transmitted packets in the i-th recovery step, where $i = r$. During ACK-splitting, all packets are successfully transmitted and the congestion window size increases by δ segments per round. From Fig. 7, we can derive $Y_{ra}^{(i)}$ [pkts] and $A_{ra}^{(i)}$ [s], which are the average number of transmitted packets and the average duration of the i-th recovery step, where $i = r$, in a similar way to the normal cycle's derivation. First, we derive the number of transmitted packets from the beginning of the i-th recovery step until the recovery of the congestion window size finishes. We denote it as $Y_e^{(i)}$ [pkts] and calculate it as follows:

$$Y_e^{(i)} = \frac{X_e^{(i)}(2W_s^{(i)} + \delta X_e^{(i)} - \delta)}{2} \tag{23}$$

where $X_e^{(i)}$ is the round number when the congestion window has recovered to the value just before the recovery cycle begins, and $W_s^{(i)}$ [pkts] is the congestion window size at the

beginning of the i-th recovery step. $X_{\rm e}^{(i)}$ is counted from the beginning of i-th recovery step, and it is obtained by

$$X_{\rm e}^{(i)} = \frac{2\,W_{\rm s}^{(1)} - W_{\rm s}^{(i)}}{\delta} + 1 \tag{24}$$

Since $W_{\rm s}^{(1)}$ is the congestion window size when the recovery cycle begins, $2\,W_{\rm s}^{(1)}$ is the congestion window size when ACK-splitting stops. $W_{\rm s}^{(i)}$ is obtained from the analysis of the $(i-1)$-th recovery step, which is explained later.

Because the congestion window will have recovered when $Y_{\rm e}^{(i)}$ packets are successfully transmitted, $a_{\rm r}^{(i)}$, which is the probability at which a packet loss occurs after recovery completion, is expressed as

$$a_{\rm r}^{(i)} = (1-p)^{Y_{\rm e}^{(i)}} \tag{25}$$

$s_{\rm r}^{(i)}$ [pkts] is the number of transmitted packets from the beginning of i-th recovery step to when the congestion window size reaches W_M, and it is calculated as

$$s_{\rm r}^{(i)} = Y_{\rm e}^{(i)} + \frac{(W_M - 2\,W_{\rm s}^{(1)})(W_M + 2W_{\rm s}^{(1)} + 1)}{2} \tag{26}$$

The above equation can be obtained easily by watching Fig. 7 with replacing $W_{\rm ra}^{i}$ by W_M. We assume that the $\alpha_{\rm ra}^{(i)}$-th packet from the beginning of the i-th recovery step is lost under the condition that wireless loss does not take place during ACK-splitting. In a similar way as we derived $\alpha_{\rm n}$ in Eq. (6), we can derive $\alpha_{\rm ra}^{(i)}$ as follows:

$$\begin{aligned}\alpha_{\rm ra}^{(i)} &= \sum_{k=Y_{\rm e}^{(i)}+1}^{s_{\rm r}^{(i)}-1} \frac{(1-p)^{k-1}p}{a_{\rm r}^{(i)}}k + \sum_{k=s_{\rm r}^{(i)}}^{\infty} \frac{(1-p)^{k-1}p}{a_{\rm r}^{(i)}}s_{\rm r}^{(i)} \\ &= \frac{1-(1-p)^{s_{\rm r}^{(i)}-Y_{\rm e}^{(i)}} + pY_{\rm e}^{(i)}}{p}\end{aligned} \tag{27}$$

We can derive $Y_{\rm ra}^{(i)}$ in two different ways:

$$Y_{\rm ra}^{(i)} = \alpha_{\rm ra}^{(i)} + W_{\rm ra}^{(i)} + 1 \tag{28}$$

$$Y_{\rm ra}^{(i)} = Y_{\rm e}^{(i)} + \sum_{k=1}^{X_{\rm ra}^{(i)}-X_{\rm e}^{(i)}} (2\,W_{\rm s}^{(1)} + k) + \frac{W_{\rm ra}^{(i)}}{2} \tag{29}$$

where $W_{\rm ra}^{(i)}$ and $X_{\rm ra}^{(i)}$ [pkts] are the congestion window size and the round number when packet loss occurs after recovery completion. Here, there is a clear relation between $X_{\rm ra}^{(i)}$ and $W_{\rm ra}^{(i)}$:

$$W_{\rm ra}^{(i)} = 2\,W_{\rm s}^{(1)} + X_{\rm ra}^{(i)} - X_{\rm e}^{(i)} \tag{30}$$

From Eqs. (28), (29), and (30), we obtain $W_{\rm ra}^{(i)}$ and $X_{\rm ra}^{(i)}$ as follows:

$$W_{\rm ra}^{(i)} = \sqrt{4{W_s^{(1)}}^2 + 2\,W_s^{(1)} - 2Y_{\rm e}^{(i)} + 2\alpha_{\rm ra}^{(i)} + 2} \tag{31}$$

$$X_{\rm ra}^{(i)} = X_{\rm e}^{(i)} - 2\,W_s^{(1)} + W_{\rm ra}^{(i)} \tag{32}$$

Note that Eq. (31) can be obtained by substituting Eqs. (28) and (30) to (29) and solving the equation for $W_{\rm ra}^{(i)}$. Then, $Y_{\rm ra}^{(i)}$ is calculated from Eqs. (27), (28) and (31), and $A_{\rm ra}^{(i)}$ is calculated as follows:

$$A_{\rm ra}^{(i)} = (X_{\rm ra}^{(i)} + 1)RTT_{\rm ra}^{(i)} + A_{to} \tag{33}$$

where $RTT_{\rm ra}^{(i)}$ [s] is average RTT of the i-th recovery step, where $i = r$. We show how to calculate $RTT_{\rm ra}^{(i)}$ in Sect. 5.1.3.

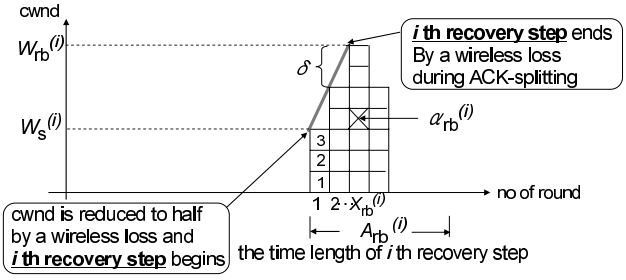

Fig. 8 Model of the number of transmitted packets in i-th recovery step where $1 \le i < r$.

(2) $1 \le i < r$ **Case**

Next, we analyze the i-th recovery step where $1 \le i < r$. Figure 8 depicts the model of the number of transmitted packets in the i-th recovery step where $1 \le i < r$. In Fig. 8, the congestion window size increases by δ segments per round. From this figure, we derive $Y_{\rm rb}^{(i)}$ [pkts] and $A_{\rm rb}^{(i)}$ [s], which are the average number of transmitted packets and the average duration of the i-th recovery step where $1 \le i < r$, in a similar way to the normal cycle's derivation. We assume that the $\alpha_{\rm rb}^{(i)}$-th packet from the beginning of the i-th recovery step is lost if a wireless loss occurs during ACK-splitting. $\alpha_{\rm rb}^{(i)}$ is derived as:

$$\alpha_{\rm rb}^{(i)} = \sum_{k=1}^{Y_{\rm e}^{(i)}} \frac{(1-p)^{k-1}p}{1-a_{\rm r}^{(i)}}k = \frac{1}{p} - \frac{Y_{\rm e}^{(i)}a_{\rm r}^{(i)}}{1-a_{\rm r}^{(i)}} \tag{34}$$

We derive $Y_{\rm rb}^{(i)}$ in two different ways, as we did with $Y_{\rm n}$ in the normal cycle:

$$Y_{\rm rb}^{(i)} = \alpha_{\rm rb}^{(i)} + W_{\rm rb}^{(i)} + 1 \tag{35}$$

$$Y_{\rm rb}^{(i)} = \sum_{k=1}^{X_{\rm rb}^{(i)}} (W_{\rm s}^{(i)} + \delta(k-1)) + \frac{W_{\rm rb}^{(i)}}{2} \tag{36}$$

where $W_{\rm rb}^{(i)}$ and $X_{\rm rb}^{(i)}$ [pkts] are the congestion window size and the round number when a wireless loss takes place during ACK-splitting. Here, there is a clear relation between $X_{\rm rb}^{(i)}$ and $W_{\rm rb}^{(i)}$:

$$W_{\rm rb}^{(i)} = W_{\rm s}^{(i)} + \delta(X_{\rm rb}^{(i)} - 1) \tag{37}$$

From Eqs. (35), (36), and (37), we obtain $W_{\rm rb}^{(i)}$ and $X_{\rm rb}^{(i)}$ as follows:

$$W_{\rm rb}^{(i)} = \sqrt{{W_{\rm s}^{(i)}}^2 - \delta W_{\rm s}^{(i)} + 2\delta + 2\delta\alpha_{\rm rb}^{(i)}} \tag{38}$$

$$X_{\rm rb}^{(i)} = \frac{\delta - W_{\rm s}^{(i)}}{\delta} + \frac{W_{\rm rb}^{(i)}}{\delta} \tag{39}$$

Note that Eq. (38) can be obtained by substituting Eqs. (35)

and (37) to (36) and solving the equation for $W_{\mathrm{rb}}^{(i)}$. Here, $W_{\mathrm{s}}^{(i+1)} = \frac{W_{\mathrm{rb}}^{(i)}}{2}$, because the $(i+1)$-th recovery step starts if a wireless loss occurs during ACK-splitting in the i-th recovery step. Then, $Y_{\mathrm{rb}}^{(i)}$ is calculated from Eqs. (34), (35) and (38), and $A_{\mathrm{rb}}^{(i)}$ is calculated as follows:

$$A_{\mathrm{rb}}^{(i)} = (X_{\mathrm{rb}}^{(i)} + 1)RTT_{\mathrm{rb}}^{(i)} + A_{to} \tag{40}$$

where $RTT_{\mathrm{rb}}^{(i)}$ [s] is the average RTT of the i-th recovery step where $1 \leq i < r$. We show how to derive $RTT_{\mathrm{rb}}^{(i)}$ in Sect. 3.3.

The analysis results of the i-th recovery step where $i = r$ and where $1 \leq i < r$ can now be used to derive $Y_{\mathrm{r}}^{(i)}$ [pkts] and $A_{\mathrm{r}}^{(i)}$ [s], which are the average number of transmitted packets and the average duration of the recovery cycle with i recovery steps as follows:

$$Y_{\mathrm{r}}^{(r)} = \sum_{i=1}^{r-1} Y_{\mathrm{rb}}^{(i)} + Y_{\mathrm{ra}}^{(r)} \tag{41}$$

$$A_{\mathrm{r}}^{(r)} = \sum_{i=1}^{r-1} A_{\mathrm{rb}}^{(i)} + A_{\mathrm{ra}}^{(r)} \tag{42}$$

3.3 Derivation of Average RTT

In this subsection, we derive the average RTT of a TCP connection, which is needed for the analyses in Sects. 3.1 and 3.2. We define the average congestion window size in a cycle as $\bar{W}$, and calculate the average RTT from $\bar{W}$. In [8], the RTT is assumed to be a constant of the congestion window size. However, we derive RTT from the congestion window size to derive the average throughput of our mechanism precisely, since the RTT significantly depends on the congestion window size. Furthermore, the derivation considers the influence of ARQ since the time for retransmission is added to the RTT when a bit error occurs in a packet.

First, we calculate the duration from when a data packet is sent out from the base station to when a data packet corresponding to an ACK packet for that data packet arrives at the base station as follows:

$$2D_w + 2D_d + 2D_d + \frac{320}{C_u} + \frac{S + 320}{C_w} \tag{43}$$

When the congestion window size is W, $W-1$ data packets are sent out from the base station after first data packet is sent. Therefore, the number of packets which remain in the buffer of the base station when a data packet corresponding to an ACK packet for first data packet arrives at the base station, N [pkts], is calculated as

$$\begin{aligned} N &= (W-1) - \left(2D_w + 2D_d + 2D_d + \frac{320}{C_u} + \frac{S + 320}{C_w}\right) \div \frac{S}{C_d} \\ &= W - RTT_{min} \div \frac{S}{C_d} \end{aligned} \tag{44}$$

The queuing delay of packets arriving at the base station is expressed as $\frac{SN}{C_d}$. The RTT of the transmission path is the sum of the queuing delay at the base station and RTT_{min}; therefore, it follows that

$$\frac{SN}{C_d} + RTT_{min} = \frac{SW}{C_d} \tag{45}$$

However, this is the RTT without considering the additional ARQ delay. We should consider the influence of ARQ. When a packet with the bit error is successfully retransmitted in the i-th retransmission, the additional delay for that packet is as follows:

$$i\left(D_u + D_d + \frac{S}{C_d}\right) \tag{46}$$

Furthermore, the RTT of a packet increases even if the bit error occurs in a preceding packet, since the packet is passed to TCP after the erroneous packet was successfully retransmitted. We denote the additional delay for a data packet when a bit error occurs in the k-th preceding packet and is transmitted successfully by the i-th retransmission as $T_{i,k}$ [s]. It follows that

$$T_{i,k} = \max\left(i\left(D_u + D_d + \frac{S}{C_d}\right) - k\frac{S}{C_d}, 0\right) \tag{47}$$

We assume that retransmission is repeated up to three times; therefore, the average RTT in consideration of ARQ becomes

$$\frac{S\bar{W}}{C_d} + \sum_{k=0}^{\infty}(e_1 T_{1,k} + e_2 T_{2,k} + e_3 T_{3,k}) \tag{48}$$

Here, the average congestion window size of the normal cycle is obtained by dividing the number of transmitted packets in the normal cycle by the number of rounds in the normal cycle. Therefore, RTT_{n}, which is the average RTT of the normal cycle, is obtained from Eq. (48) as

$$RTT_{\mathrm{n}} = \frac{S}{C_d} \cdot \frac{Y_{\mathrm{n}}}{X_{\mathrm{n}} + 1} + T \tag{49}$$

where $T = \sum_{k=0}^{\infty}(e_1 T_{1,k} + e_2 T_{2,k} + e_3 T_{3,k})$. Similarly, $RTT_{\mathrm{ra}}^{(i)}$ and $RTT_{\mathrm{rb}}^{(i)}$, the average RTT of the i-th recovery step which is terminated after recovery completion or during ACK-splitting, are calculated as follows:

$$RTT_{\mathrm{ra}}^{(i)} = \frac{S}{C_d} \cdot \frac{Y_{\mathrm{ra}}^{(i)}}{X_{\mathrm{ra}}^{(i)} + 1} + T \tag{50}$$

$$RTT_{\mathrm{rb}}^{(i)} = \frac{S}{C_d} \cdot \frac{Y_{\mathrm{rb}}^{(i)}}{X_{\mathrm{rb}}^{(i)} + 1} + T \tag{51}$$

3.4 Throughput Derivation

In the previous subsections, we showed how to derive the average number of transmitted packets and the average durations of the normal and recovery cycles. In this subsection, we derive the probability with which a TCP connection is in each cycle, and the average throughput of the mechanism by taking their average.

The probability that the number of recovery steps is r in a recovery cycle is $\prod_{i=1}^{r-1}(1-a_{\rm r}^{(i)})$, because the $(r-1)$ recovery step is terminated during ACK-splitting. We define $P_{\rm n}$ as the probability with which a TCP connection is in the normal cycle. Since the normal cycle starts when congestion loss occurs, the following relation is true:

$$P_{\rm n} = P_{\rm n}c_{\rm n} + \sum_{r=1}^{\infty}\left\{(1-P_{\rm n})\prod_{i=1}^{r-1}(1-a_{\rm r}^{(i)})c_{\rm r}^{(r)}\right\} \tag{52}$$

where $c_{\rm n} = (1-p)^{s_{\rm n}-1}$, which indicates the probability that a congestion loss occurs in the normal cycle, and $c_{\rm r}^{(r)} = (1-p)^{s_{\rm r}^{(r)}-1}$, which is the probability that a congestion loss occurs in a recovery cycle with r recovery steps. From this equation, it follows that

$$P_{\rm n} = \frac{\sum_{r=1}^{\infty}\prod_{i=1}^{r-1}(1-a_{\rm r}^{(i)})c_{\rm r}^{(r)}}{1-c_{\rm n}+\sum_{r=1}^{\infty}\prod_{i=1}^{r-1}(1-a_{\rm r}^{(i)})c_{\rm r}^{(r)}} \tag{53}$$

Finally, the average throughput of a TCP connection with our mechanism, B [pkts/sec], is obtained by using $P_{\rm n}$ as follows:

$$B = \frac{P_{\rm n}Y_{\rm n}+\sum_{r=1}^{\infty}\left\{(1-P_{\rm n})\prod_{i=1}^{r-1}(1-a_{\rm r}^{(i)})(1-p)^{s_{\rm r}^{(r)}}Y_{\rm r}^{(r)}\right\}}{P_{\rm n}A_{\rm n}+\sum_{r=1}^{\infty}\left\{(1-P_{\rm n})\prod_{i=1}^{r-1}(1-a_{\rm r}^{(i)})(1-p)^{s_{\rm r}^{(r)}}A_{\rm r}^{(r)}\right\}} \tag{54}$$

Note that we can also derive the average throughput of a normal TCP-SACK connection by substituting $\delta = 1$ instead of using Eq. (22).

4. Numerical Results

In this subsection, we show the numerical results of the throughput analysis. We first set parameters (C_w, D_w, C_d, D_d, C_u, D_u, M) in Fig. 3 to (10 Mbps, 45 ms, 2 Mbps, 1 ms, 384 kbps, 1 ms, 50 pkts), and we choose 1 second for RTO. Figure 9 compares the throughputs of the analysis and the simulation. The figure shows that the analysis gives good estimations of the throughput of TCP-SACK and or mechanism. The throughput of the analysis is a little larger than that of the simulation especially when the bit error rate is high. This is because we did not consider the occurrence of timeout enough in the analysis: timeout caused by a packet loss which occurs when the congestion window size is not over 3 segments, or timeout when multiple packet losses occur in one RTT and the third duplicate ACK packet does not return. Timeout occurs in the simulation more often than we expected in the analysis, so the simulated throughput significantly deteriorates in comparison with the analysis's prediction.

Next, we show the analysis results for various wireless network environments, with different bandwidths for the downlink and uplink of the wireless network. That is, we change C_d and C_u in Fig. 3 and keep the other parameters the same as in Fig. 3. Figure 10 shows the analysis's results

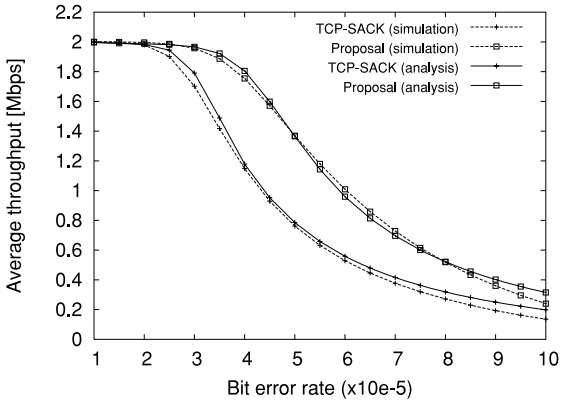

Fig. 9 Comparison of analysis and simulation results.

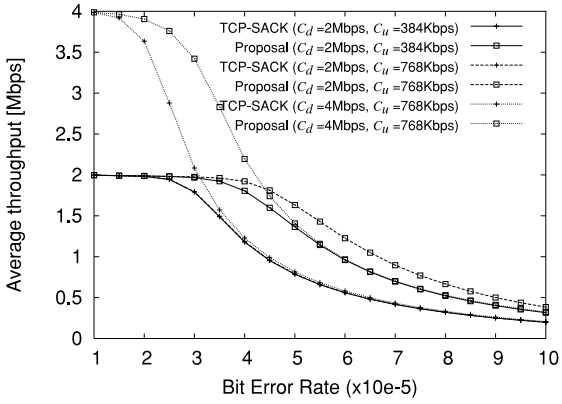

Fig. 10 Analysis results for different environments.

for (C_d, C_u) of (2 Mbps, 384 kbps), (2 Mbps, 768 kbps), and (4 Mbps, 768 kbps). For (C_d, C_u) of (2 Mbps, 384 kbps), our mechanism increases throughput by up to 74% when the bit error rate is around 5×10^{-5}. This means our mechanism is up to 650 kbps faster than normal TCP-SACK when the bit error rate is around 4.5×10^{-5}. For (C_d, C_u) of (2 Mbps, 768 kbps), that is, when the uplink bandwidth becomes larger, our mechanism becomes more effective. The throughput becomes 118%; i.e., our mechanism is 850 kbps faster than normal TCP-SACK. This is because δ, the number of ACK packets which can be sent for one data packet, increases when the uplink bandwidth becomes larger, as indicated in Eq. (22). This means that the speed of the congestion window increase during ACK-splitting becomes higher and the throughput degradation is restrained more promptly.

For (C_d, C_u) of (4 Mbps, 768 kbps), that is, when downlink and uplink bandwidths both become larger, the throughput improvement becomes up to 80%, and it does not change so largely compared with the (2 Mbps, 384 kbps) case. When the downlink bandwidth becomes larger, the arrival interval of data packets becomes short, so δ does not increase, even if the uplink bandwidth becomes larger. Therefore, the throughput increase doesn't change so much. However, in terms of the amount of throughput, our mechanism becomes more effective. In this environment, the throughput

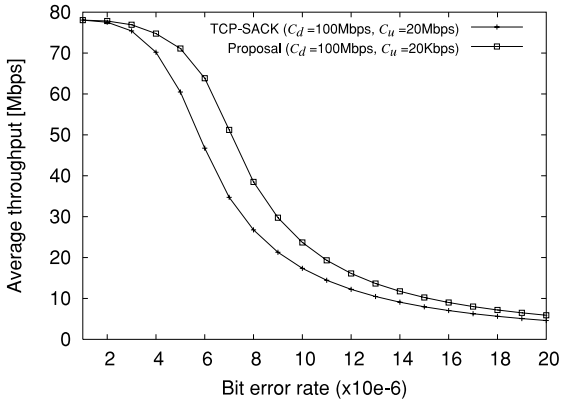

Fig. 11 Analysis results for a large bandwidth network such as a fourth-generation mobile network.

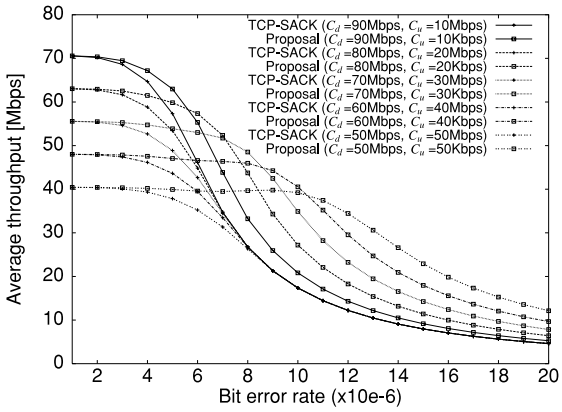

Fig. 12 Analysis results for a large bandwidth network such as IEEE802.11n.

is up to 1.34 Mbps higher than that of normal TCP-SACK.

The above results indicate that our mechanism works more effectively in an environment where the bandwidth of the wireless network is large. Since the bandwidth of wireless networks will increase significantly in the coming years, we can conclude that the usability of our mechanism will increase in the future. Finally, we show some examples of the analysis results for two kinds of future wireless network. One network is based on fourth generation mobile technology, for which we assume that its downlink bandwidth is 100 Mbps and its uplink bandwidth is 20 Mbps [10]. We set C_w to 1 Gbps and kept other parameters the same. The result is shown in Fig. 11. We see that the maximum throughput improvement is 47%, and it is small compared with the case when the bandwidth is small. This is because δ is reduced when the bandwidth of the wireless downlink is large, according to the second term of Eq. (22). However, the throughput of our mechanism is up to 17 Mbps higher than that of normal TCP-SACK.

The other sort of future network is based on the wireless LAN technology of IEEE802.11n [11]. We assume that the bandwidth of the wireless link is 100 Mbps. Since the wireless channels are shared by the downlink and uplink in wireless LAN, we calculate the throughput in different allocations of bandwidth for the downlink and the uplink. Figure 12 depicts the analysis results when we assume that the bandwidth allocations for the downlink and the uplink are (90 Mbps, 10 Mbps), (80 Mbps, 20 Mbps), (70 Mbps, 30 Mbps), (60 Mbps, 40 Mbps), and (50 Mbps, 50 Mbps). We see that when the downlink bandwidth is large, the throughput of our mechanism is large for a low bit error rate. On the other hand, when the bit error rate is high, our mechanism is more effective with a large uplink bandwidth. Here, an interesting characteristic of our mechanism is that the best allocation of downlink and uplink bandwidths depends on the bit error rate. Moreover, our mechanism increases throughput by up to several tens of Mbps, and its effectiveness is very large regardless of the bandwidth allocation.

5. Conclusion

In this paper, we mathematically analyzed the receiver-based ACK splitting mechanism which improves the TCP throughput over wired and wirelesss heterogeneous networks. The analysis indicates that the larger the bandwidth of the wireless link is, the more effective our mechanism becomes. As a result, we concluded that the mechanism's usability will increase in the future as wireless networks become faster.

In the future, we plan to implement the mechanism in the actual wireless network environments and confirm its effectiveness in actual operation.

References

[1] M. Nakata, G. Hasegawa, and H. Nakano, "Receiver-based ACK splitting mechanism for TCP over wired/wireless integrated networks," Proc. Center of Excellence Wireless and Information Technology 2005, p.22, Dec. 2005.

[2] F. Lefevre and G. Vivier, "Understanding TCP's behavior over wireless links," Proc. Communications and Vehicular Technology, pp.123–130, Oct. 2000.

[3] A. Bakre and B.R. Badrinath, "I-TCP: Indirect TCP for mobile hosts," Proc. 15th International Conference on Distributed Computing Systems, pp.136–143, May 1995.

[4] H. Balakrishnan, S. Seshan, and R.H. Katz, "Improving reliable transport and handoff performance in cellular wireless networks," ACM/Baltzer Wireless Networks, vol.1, no.4, pp.469–481, Dec. 1995.

[5] C. Casetti, M. Gerla, S. Mascolo, M.Y. Sanadidi, and R. Wang, "TCP Westwood: Bandwidth estimation for enhanced transport over wireless links," Proc. ACM MOBICOM, pp.287–297, July 2001.

[6] E.H.K. Wu and M.Z. Chen, "JTCP: Jitter-based TCP for heterogeneous wireless networks," IEEE J. Sel. Areas Commun., vol.22, no.4, pp.757–766, May 2004.

[7] S. Savage, N. Cardwell, D. Wetherall, and T. Anderson, "TCP congestion control with a misbehaving receiver," ACM SIGCOMM Computer Communication Review, vol.29, no.5, pp.71–78, Oct. 1999.

[8] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP throughput: A simple model and its empirical validation," Proc. ACM SIGCOMM'98, pp.303–314, Aug. 1998.

[9] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP selective acknowledgment options," Request for Comments (RFC) 2018, Oct. 1996.
[10] NTT Docomo, "Press release article: NTT docomo achieves 1 Gbps packet transmission in 4G field experiments," available from http://www.nttdocomo.com/presscenter/pressreleases/, June 2005.
[11] IEEE Computer Society/Local and Metropolitan Area Networks, "802.11n: Amendment to standard: wireless LAN medium access control (MAC) and physical layer (PHY) specifications: enhancements for higher throughput," Sept. 2003.

Go Hasegawa received the M.E. and D.E. degrees in Information and Computer Sciences from Osaka University, Osaka, Japan, in 1997 and 2000, respectively. From July 1997 to June 2000, he was a Research Assistant of Graduate School of Economics, Osaka University. He is now an Associate Professor of Cybermedia Center, Osaka University. His research work is in the area of transport architecture for future high-speed networks. He is a member of the IEEE.

Masashi Nakata is now a master-course student at Graduate School of Information Science and Technology, Osaka University. His research work is in the area of transport architecture for wired/wireless integrated networks.

Hirotaka Nakano received the M.E. and D.E. degrees in Electrical Engineering from Tokyo University, Tokyo Japan, in 1974 and 1977, respectively. He joined NTT Laboratories in 1977 and has been engaged in research and development of picture production systems, videotex systems, and multimedia-on-demand systems. He had been an executive manager of the Multimedia Systems Laboratory of the NTT Human Interface Laboratories from 1995 to 1999. Afterwards, he served as the head in the Multimedia Laboratory of the NTT Docomo until 2004, and now he is an Professor of Cybermedia Center, Osaka University. His research work is in the area of ubiquitous networks. He is a member of the IEEE and the institute of Image Information and Television Engineers of Japan.