IEICE TRANS. COMMUN., VOL.E90–B, NO.5 MAY 2007

PAPER

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

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

SUMMARY With the rapid development of wireless network technologies, heterogeneous networks with wired and wireless links are becoming common. However, the performance of TCP data transmission deteriorates significantly when a TCP connection traverses such networks, mainly because of packet losses caused by the high bit error rate of wireless links. Many solutions for this problem have been proposed in the past literature. However, most of them have various drawbacks, such as difficulties in their deployment by the wireless access network provider and end users, violation of TCP's end-to-end principle by splitting the TCP connection, or inapplicability to IP-level encrypted traffic because the base station needs to access the TCP header. In this paper, we propose a new mechanism without such drawbacks 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. We evaluate the performance of our mechanism and show that our mechanism can increase throughput by up to 94% in a UMTS network. The simulation results also show that our mechanism does not significantly deteriorate even when the receiver cannot perfectly distinguish whether packet losses are due to network congestion or bit errors on the wireless links.

key words: *TCP, wired/wireless heterogeneous networks, ACK splitting mechanism, congestion control mechanism*

1. Introduction

Various wireless network technologies have become popular in recent years, and as their bandwidths become larger, they are coming to be used as access networks to the Internet. In particular, heterogeneous networks with wired and wireless links have become common. We can now utilize the Internet access environment out of doors in public wireless LAN spots. Furthermore, a lot of people have mobile phones which embody third-generation mobile technologies typified by the Universal Mobile Telecommunications System (UMTS) [1], and mobile phone's wireless network technologies are often used as the last-mile access network to the Internet.

In such heterogeneous networks, except for P2P applications, the sender of the data transmission is a server on the Internet (the wired-side network), and the receiver is a client at the edge of the wireless network. That is, the sender and receiver are connected via wired and wireless networks, and data transmission is done from the wired network hosts to wireless network hosts by using Transmission Control Protocol (TCP) [2], the most popular transport-layer protocol in the current Internet. However, it is a well-known problem that the performance of TCP deteriorates significantly when a TCP connection traverses wired and wireless heterogeneous networks [3], [4]. TCP was originally designed for wired networks, so it regards all packet losses detected at the TCP sender as losses caused by network congestion: the result of buffer overflow of the intermediate routers. In response, the TCP sender slows down its data transmission rate by reducing the congestion window size to half in order to avoid further network congestion. Unfortunately, in heterogeneous networks, since the bit errors occur with a high ratio on wireless links [5], the packets with bit errors are discarded at a lower layer and this causes packet losses in the upper-layer TCP. In this paper, we define such packet losses as *wireless losses*, and packet losses caused by network congestion as *congestion losses*. Wireless losses are not the result of the network congestion, and it is unnecessary to slow down data transmission rate in response to them. However, since TCP lacks a function to know the reason for a packet loss, it decreases the data transmission rate regardless of the cause. This results in a situation in which the higher the bit error rate of the wireless link is, the more often wireless losses occur, and TCP's data transmission performance deteriorates more significantly.

Many solutions to this problem have been proposed in the past literature [6]–[18]. Some of them [6]–[11] modify the functions of the base station, which is the border node of the wired and wireless networks. 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 into a wired part and a wireless part at the base station. Furthermore, they can not be applied when a lower-layer encryption mechanism such as IPSec [19] is utilized because they need to access TCP header at the base station. Other solutions [12]–[18] modify the sender-side TCP algorithm. They try not to slow down TCP's transmission rate when a wireless loss is detected. Since these solutions do not split the TCP connection and do not access the TCP header at nodes other than the end hosts, 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 in-

Manuscript received May 15, 2006.
Manuscript revised December 3, 2006.

[†]The authors are with 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.5.1132

stability to their systems. Because of the above drawbacks, the most of the existing solutions are not widely deployed. To be universally used in the current and the future Internet, what is really needed is a solution without such drawbacks, that is, one which can be deployed by wireless access network providers or end users seeking to improve TCP's performance over heterogeneous networks.

The main contribution of this paper is the proposal of a receiver-based ACK splitting mechanism to improve TCP throughput over wired and wireless heterogeneous networks. 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.

Regarding the modification of the receiver-side TCP algorithm, it is impossible to prevent the sender's congestion window size from being reduced when wireless losses occur. Instead, we restrain the throughput degradation by using the ACK-splitting method [20], which can be accomplished by modifying the receiver's TCP. The ACK-splitting method increases the congestion window size of the sender-side TCP more quickly than usual by sending multiple ACKnowledgement (ACK) packets when a data packet arrives at the receiver. Although ACK-splitting is helpful in itself, an inappropriate execution of it has a bad influence on the network and the sender-side TCP. Consequently, we introduce functions to control ACK-splitting appropriately: packet loss distinction, control of ACK-splitting duration, and control of ACK sending rate.

To evaluate the effectiveness of our mechanism, we compared the throughput of the proposed mechanism with that of the original TCP by simulation. The simulation results show that the throughput is 93% higher than that of the original TCP when the proposed mechanism is incorporated into a receiver TCP on wireless network clients of heterogeneous networks in a UMTS network. The simulations also show that our mechanism is effective even when the receiver can not perfectly distinguish the cause of packet loss.

The rest of this paper is organized as follows. In Sect. 2, we explain the details of performance degradation problem of TCP over wired and wireless heterogeneous networks, and describe the existing solutions and the ACK-splitting method. In Sect. 3, we describe our receiver-based ACK splitting mechanism in detail. In Sect. 4, we present the results of a simulation to evaluate the mechanism. Section 5 concludes this paper and offers an outline for future work on this topic.

2. Research Background

In this section, we introduce the problem of TCP over wired and wireless heterogeneous networks that is addressed in this paper, describe some of the existing solutions for this problem, and point out their drawbacks. We also explain the ACK-splitting method that is the basis of our mechanism.

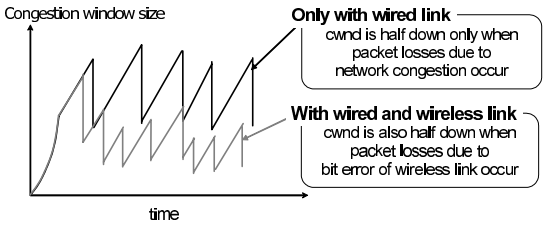


Fig. 1 Unnecessary degradation of the congestion window.

2.1 Problems on TCP over Wired/Wireless Heterogeneous Networks

The congestion control algorithm of TCP-Reno increases the congestion window size by its inverse when the sender receives an ACK packet, as long as no packet loss is detected. That is, the congestion window size increases additively by one segment every RTT. When TCP-Reno detects packet losses, it reduces the congestion window size by half to slow down the data transmission rate. These behaviors are based on an assumption that all packet losses are caused by network congestion when buffer overflow occurs at intermediate routers. This assumption works well if the TCP connection traverses only wired networks, since the bit error rate of wired links is now negligible [21], [22]. However, when wireless links are in the transmission path, this assumption becomes inappropriate because wireless network links have a high bit error rate, and this causes packet losses that are not the result of the network congestion. In this paper, we define such losses as *wireless losses* and losses caused by network congestion as *congestion losses*.

Wireless loss is not a sign of network congestion, so it is unnecessary to slow down the data transmission rate and TCP should not decrease its congestion window size in response to it. However, TCP can not distinguish wireless losses from congestion losses, and thus, it reduces the congestion window size by half in response to either type of packet loss. Figure 1 depicts unnecessary changes in the congestion window size against wireless losses. This figure indicates that the performance of TCP's data transmission decreases, even when there is unutilized bandwidth that is available for the TCP connection.

2.2 Existing Solutions

The past literature proposes many solutions to solve performance degradation problem of TCP over networks including wireless links [6]–[18]. Here, we focus on the solutions with modification of the sender-side TCP, since the proposed mechanism is this paper is also an end-to-end approach. We suppose that the TCP sender is in the wired network and the TCP receiver is connected to it through a wireless access network. This supposition is valid because the situation is commonplace on wired and wireless heterogeneous networks.

In [12]–[18], the sender-side TCP algorithm is modi-

fied so that it does not unnecessarily reduce the congestion window size. These approaches modify the congestion control mechanism of the sender TCP so that it will not shrink the congestion window in response to wireless losses.

In TCP-Westwood [12], the sender dynamically estimates the available network bandwidth by measuring the arrival rate of ACK packets. When packet losses are detected, the sender sets a slow start threshold ($ssth$) to a value calculated from the measured available bandwidth, regardless of whether the losses are wireless or congestion losses. The effectiveness of a rate-based solution such as TCP-Westwood depends on the accuracy of the available bandwidth estimation. The other literature describes many algorithms for estimating available bandwidth, e.g., TCP-Jersey [15] and these are used to solve the performance degradation problem of TCP.

Jitter-based TCP (JTCP) [16] is different from the rate-based solutions. It makes the sender-side TCP able to distinguish the causes of packet loss. Only when congestion losses are identified does the sender reduce its congestion window by half. The cause of packet loss is distinguished calculating Jr, i.e.,

$$Jr = \frac{(R_{n-1} - R_{n-w}) - (S_{n-1} - S_{n-w})}{R_{n-1} - R_{n-w}} \quad (1)$$

where R_i is the receiving time for the i th data packet, S_i is the sending time for the i th data packet, and w is the current congestion window size in packets. Jr is an estimate of the network congestion level from the jitter of the one-way transmission delay of data packets. Using this value, JTCP distinguishes the cause of packet loss as follows:

$$\begin{cases} Jr > k/w & \rightarrow \text{ congestion losses} \\ Jr \leq k/w & \rightarrow \text{ wireless losses} \end{cases}$$

where k is a control parameter. If k is greater than 1, the sender may misjudge congestion losses as wireless losses. If k is less than 1, the sender will become more sensible, but some wireless losses might be misjudged as congestion losses. In [16], the authors' experiments indicate that JTCP has the best performance with $k = 1$.

Although these solutions can avoid unnecessary reductions in congestion window size in response to wireless losses, their deployment scenarios face problems. In many cases, the TCP sender is a server host on the wired network and the TCP receiver is a wireless client machine. As such, we would need to deploy the above solutions on the network's servers. The problem is that server administrators might see this as an undesirable additional cost, since only the wireless network users would receive the benefit of the modification. Furthermore, the server administrators would likely prefer not to change the system kernel of their servers because doing so may cause instabilities in their system.

2.3 ACK-Splitting Method

As described in Sect. 2.2, the existing solutions which modify the base station or the sender-side TCP have various drawbacks. In this paper, we propose a mechanism that modifies only the receiver-side TCP and does not have the drawbacks mentioned above. Here, we describe the ACK-splitting method [20] which is the basis of our mechanism.

ACK-splitting increases the speed at which TCP's congestion window size changes by sending back multiple ACK packets for one data packet received by a TCP receiver. This method is derived from the incongruence of the error control and congestion control of TCP. TCP is a basically byte-stream protocol. Therefore, each data packet has a sequence number field that refers to byte offsets within a TCP data stream. For example, let's say a packet has a sequence number 4001 and the previously transmitted packet has a sequence number 3001. Thus, the former packet includes data represented by byte offsets from 3001 to 4000. An ACK packet also has a sequence number field that indicates the last byte offsets received consecutively. When the TCP sender receives an ACK packet whose sequence number is 4000, it means the data has fully arrived at the receiver until 4000 bytes from the first byte. On the other hand, TCP's congestion control mechanism is maintained in terms of packets rather than bytes. For example, at the TCP sender, the congestion window size is updated on receiving each ACK packet which acknowledges new data, regardless of the acknowledged data size. During the slow start phase, the TCP sender increases its congestion window size by 1 segment size for each ACK packet, and during a congestion avoidance phase, the congestion window size is increased by its inverse value for each ACK packet.

The mismatch between the byte granularity of the error control and the packet granularity of the congestion control enables an ACK-splitting method as follows. Upon receiving a data packet containing N bytes, the receiver generates M separate ACK packets ($M < N$) rather than 1 ACK packet as the normal TCP does. Each of M separate ACK packets covers one of M distinct pieces of the received data packet. That is, each ACK packet has a different acknowledgement sequence number to acknowledge new data. Without ACK-splitting, the sender updates its congestion window size only once upon receiving the original ACK packet. With ACK-splitting, the sender receives M ACK packets and updates the congestion window size M times. Because the congestion window size increase for 1 ACK packet is independent of the acknowledged data size, the congestion window size increases more rapidly.

The behavior of ACK-splitting is depicted with the time chart in Fig. 2. In this figure, each data and ACK packet exchanged between the sender and the receiver is indicated with an arrow, and time increases toward the bottom of the chart. The numbers on the sender side indicate the sequence number of sent data packets, and those on the receiver side indicate the sequence number of ACK packets. In this example, the sender-side TCP is in a slow start phase. Initially, the congestion window size is 1 segment, and a packet with sequence number 1 and whose size is 1000 bytes is sent. On receiving the data packet, the receiver normally sends back an ACK packet with sequence number 1000. Instead, in this

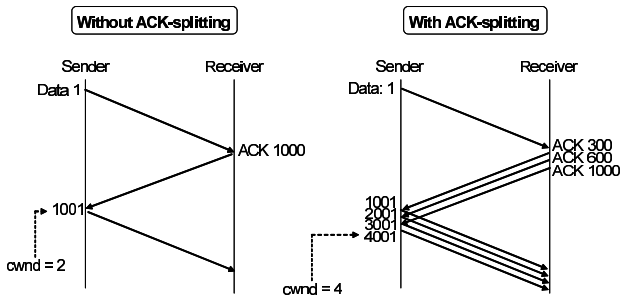


Fig. 2 Time chart of ACK-splitting during slow start phase.

example, the receiver splits it into three ACK packets with different sequence numbers: 300, 600, and 1000 (other sets of acknowledgement numbers are acceptable). On receiving the ACK packet with sequence number 300, the sender updates the congestion window size since the ACK packet acknowledges new data. On each reception of the remaining two ACK packets with sequence numbers 600 and 1000, the sender updates the congestion window size, since those ACK packets also acknowledge new data. As a result, the congestion window size increases by 3 segments in 1 RTT, whereas it increases by only one segment per RTT in the slow start phase of the normal TCP.

SPACK [11] is a solution that uses ACK-splitting at the base station. It detects wireless loss at the base station by comparing the sequence numbers and the acknowledgement numbers of packets passing through the base station, and splits the first ACK packet which acknowledges new data of N bytes after the wireless loss into N separate ACK packets. This means that each separate ACK packet acknowledges only 1 byte. SPACK can not be applied if the IP level encryption mechanism is utilized, since it reads the TCP header at the base station, and it does not consider the trade-off between effectiveness and the bad influence of ACK-splitting.

Although ACK-splitting is helpful in avoiding unnecessary reductions in congestion window size, it may increase the load of the sender-side TCP and may increase network congestion on the path from the sender or to the sender. From another viewpoint, ACK-splitting can be made into an attack on network servers [20]. In [20], TCP-Daytona is mentioned as a variant that does not increase the congestion window size for split ACK packets. If such a TCP variant is introduced in the server, we can not use an ACK-splitting method to recover the congestion window size. In fact, operating system of Linux 2.2 and latter versions adopt such an anti-ACK-splitting mechanism [20]. Therefore, our mechanism is invalid for servers with Linux2.2 or latter versions. On the other hand, FreeBSD6.0 and Solaris10 do not deploy such a mechanism, so our mechanims is valid for them. However, since we believe the advantage of the proposed mechanism as presented in this 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, that 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.

3. Proposed Mechanism

3.1 Overview of Proposed 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 back to its original value just before wireless losses occur, by using the ACK-splitting method explained in the previous section. However, since ACK-splitting has some demerits when it is used inappropriately, it is necessary to control its execution. In the following subsections, we outline three problems stemming from inappropriate execution of ACK-splitting and functions to alleviate them.

Figure 3 is an overview of how the mechanism works. When packet loss occurs, the mechanism distinguishes the cause of the loss by using the functions described in Sect. 3.2. If the loss is wireless loss, it executes ACK-splitting to recover the congestion window size. Here, the function described in Sect. 3.3 determines the duration of ACK-splitting, and the function described in Sect. 3.4 controls the rate of the congestion window's increase so as not to congest the uplink of the wireless network.

3.2 The Cause of Packet Loss Distinction

The purpose of our mechanism is to increase the congestion window size more quickly than usual after wireless losses occur and the congestion window is halved. This is desirable because wireless losses do not indicate network congestion. On the other hand, 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, we introduce a function to distinguish wireless losses from congestion losses at a receiver host. There are three distinct methods (Sects. 3.2.1–3.2.3) to distinguish the cause of packet loss.

3.2.1 MAC-Method

It is possible to distinguish the cause of packet loss perfectly at a receiver host connected to a wireless link by using information from the MAC layer. We call this method the MAC-method. In most wireless networks, a packet is divided into

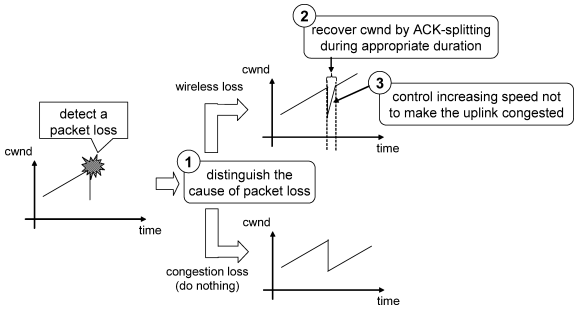


Fig. 3 Overview of the proposed mechanism.

multiple frames before it is transmitted. The base station retransmits transmitted frame in which unrecoverable bit errors occur by using Automatic Repeat Request (ARQ) [23]. The frame is discarded if the retransmission fails a specific number of times (usually three times). The packet including a discarded frame(s) is then discarded in the MAC layer, and this results in a wireless loss. Because the receiver checks the bit error in the MAC layer, we can use MAC layer information to identify which packets have bit errors. However, since such a method requires cooperation between the transport layer and MAC layer, it is undesirable from the viewpoint of the hierarchical structure of network protocols. In addition, its implementation is difficult because the behavior of the MAC layer would have to be modified. (For example, we would need to put additional codes in the device driver of the wireless NIC at the receiver host.) The MAC-method does have the advantage of providing complete information on the cause of packet loss, whereas the following non inter-layer approaches can not distinguish the cause of packet loss perfectly.

3.2.2 JTCP-Method

This non inter-layer distinction method leverages the Jitter-based TCP [16] described in Sect. 2.2.2. In JTCP, the network congestion level Jr is estimated from the jitter of the one-way delay of data packets (Eq. (1)). The required information to calculate Jr is the receiving time for the i th data packet, R_i, the sending time for the i th data packet, S_i, and the current congestion window size of the sender, w. These values are obtained or estimated at the receiver.

R_i can be directly obtained when the i th data packet arrives at the receiver, and S_i can be obtained from the TCP header of the i th data packet by using TCP's timestamp option [24]. The sender's time and the receiver's time are usually not perfectly synchronized, but this does not affect Jr's derivation, since the difference in the times at each host are used in Eq. (1): $S_{n-1} - S_{n-w}$ and $R_{n-1} - R_{n-w}$. Furthermore, the clock skew time is not the same at the sender and the receiver. However, this difference is negligible when one considers that the clock skew time is tens of pico-seconds [25], whereas the one-way transmission delay is usually larger than milli-second order.

w is not directly obtained at the receiver; Instead, a simple estimation method is used as follows. The congestion window size of a TCP connection can be approximated by the product of RTT and average throughput of the TCP connection. At the receiver host, RTT can be obtained with the timestamp option, and the average throughput can be gotten by counting the total bytes of arrival data packets in each RTT, under the assumption that the receive socket buffer size is sufficiently large. We calculate *cwnd_r*, the estimated size of the congestion window, as

$$cwnd\_r = s\_rtt \times \overline{\rho} \tag{2}$$

where *s_rtt* is smoothed RTT, and $\overline{\rho}$ is average throughput during the last RTT. This estimation is done every RTT. Since the actual congestion window of the sender TCP increases in size during the period between estimations, there would be an error between *cwnd_r* and the window's actual size. To reduce the estimation error, we update *cwnd_r* every time an ACK packet is sent to acknowledge new data. That is, we increase *cwnd_r* by its inverse for every ACK packet.

We then calculate Jr and when a packet loss occurs, we judge it as congestion loss if $Jr > \frac{1}{w}$, and wireless loss otherwise (we chose 1 as the value of the control parameter k, which the author of [16] says is the best value)

3.2.3 RTT-Method

The other non inter-layer distinction method is the RTT-method, which distinguishes the cause of packet loss at random by using only the RTT value. Reference [26] proposes a flexible mechanism for controlling the congestion window size of a TCP connection. The flexible mechanism dynamically changes the congestion window size according to the network congestion level estimated from RTT observations. Our RTT-method uses this mechanism to distinguish the cause of packet loss.

We assume that the receiver-side TCP uses the timestamp option in backward way. From the description of RFC [24], the behavior of TCP with timestamp option in backward way is that the receiver TCP adds a timestamp to the option field of each ACK packet to be sent, and the sender TCP copies the timestamp to the data packets triggered by receiving the ACK packet.

We assume that the more congested the network is, the larger RTT will be, because the queuing delay at congested routers will increase. Thus, when packet loss is detected, we determine it to be congestion loss with the following probability, P_c:

$$P_c = \frac{RTT_{loss} - RTT_{min}}{RTT_{max} - RTT_{min}} \tag{3}$$

where RTT_{loss} is the RTT when packet loss is detected, RTT_{min} and RTT_{max} are the minimum and maximum RTTs of the TCP connection. Since RTT_{loss} is between RTT_{min} and RTT_{max}, P_c would be between 0 and 1. P_c is 0 when RTT_{loss} is identical to RTT_{min}, it increases additively as RTT_{loss} increases. It reaches 1 when RTT_{loss} is the same

as RTT_{max}. This behavior is clearly based on the assumption that increasing the network congestion level increases RTT. Note that the receiver TCP determines packet loss type in on stochastic behavior: it determines the loss is wireless loss with the probability of P_c, and congestion loss with the probability of $1 - P_c$.

3.3 Control of ACK-Splitting Duration

ACK-splitting begins after a packet loss is distinguished as wireless loss by using distinction method we explained in the previous subsection. 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. Note that our purpose is to recover from unnecessary reductions in the congestion window size. Therefore, ACK-splitting should be executed until the congestion window size reaches the value just before the wireless loss occurred. By maintaining an estimation of the congestion window size as described in Sect. 3.2.2, we can realize such a behavior to control the duration of ACK-splitting:

1. When a wireless loss occurs, we record the congestion window size as *target_cwnd*. We do not override *target_cwnd* when the wireless loss occurs during ACK-splitting since it has been already set at a precedent wireless loss.
2. During ACK-splitting, the current estimation of the congestion window size (*cwnd_r*) is compared with *target_cwnd*. Then, while *cwnd_r* < *target_cwnd*, ACK-splitting continues, and when *cwnd_r* ≥ *target_cwnd*, ACK-splitting stops.
3. If congestion loss occurs during ACK-splitting, the ACK-splitting stops, to avoid network congestion.

According to the above procedure, the congestion window size changes as depicted in Fig. 4.

The fairness between TCP with the proposed methods and the traditional TCP Reno is important when we propose enhancement of TCP congestion control mechanism. We have injected our fundamental idea for fairness into the proposed mechanism, that is, we stop splitting ACK packets when the window size is enough recovered, as explained in Sect. 3.3. That is, the proposed mechanism mimics the behavior of the original TCP Reno without wireless packet losses. Therefore, if the network congestion occurs in wired networks, the proposed mechanism share the bottleneck bandwidth equally with competing TCP Reno connections.

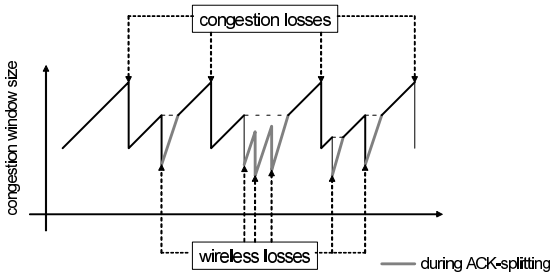


Fig. 4 Expected changes in congestion window size with our mechanism.

Note that the proposed algorithm for recovering the congestion window size is used both in the slow start phase and the congestion avoidance phase. Since we stop the recover of the congestion window size when it reaches the size when the wireless loss occurred, the rapid increase of the window size in slow start phase would last only for 1 RTT.

3.4 Control of ACK Sending Rate

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. That is, the ACK sending rate is low if a few ACK packets are sent back for one data packet, and the ACK sending rate is high if many ACK packets are sent back for one data packet. To control the number of split ACK packets, we estimate the congestion level of the uplink by referring to the length of the sending queue for the uplink network interface at the receiver.

We denote the sending queue length of an uplink network interface just before sending back ACK packets for the i th data packet as r_queue_i, and the length just after sending back ACK packets for the i th data packet as s_queue_i. Then, ΔQ_i, the number of ACK packets which are sent out on the uplink from the sending queue between the arrivals of the $(i-1)$ th data packet and the i th data packet, can be described as follows:

$$\Delta Q_i = s\_queue_{i-1} - r\_queue_i \tag{4}$$

Assuming that the arrival interval of data packets does not change rapidly, at least ΔQ_i ACK packets can be sent out between arrivals of the i th data packet and the $(i+1)$ th data packet without making the uplink congested. Thus, after ACK packets for the i th data packet are put on the sending queue, the number of ACK packets in the sending queue should become ΔQ_i. Furthermore, if the sending queue is empty when the ACK packets are sent, we can guess that the uplink can transmit more ACK packets. Given these considerations, ACK_i, the number of ACK packets for i th data packet, is calculated as

$$ACK_i = \begin{cases} \Delta Q_i - r\_queue_i & (\text{if} \quad r\_queue_i > 0) \\ \Delta Q_i + 1 & (\text{if} \quad r\_queue_i = 0) \end{cases} \tag{5}$$

Figure 5 depicts typical changes in the sending queue length

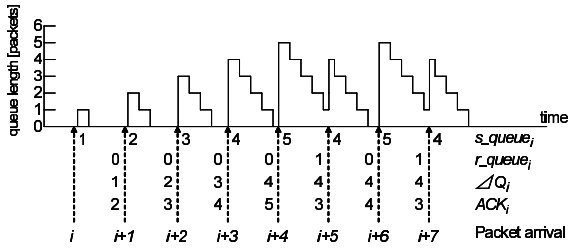


Fig. 5 Typical changes in the number of ACK packets for each data packet.

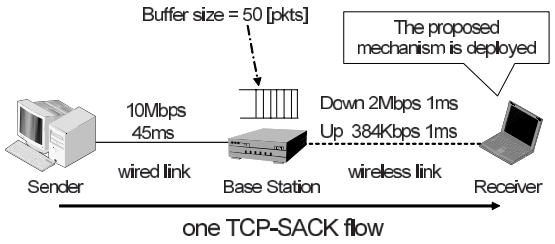


Fig. 6 Network model of the simulation experiments.

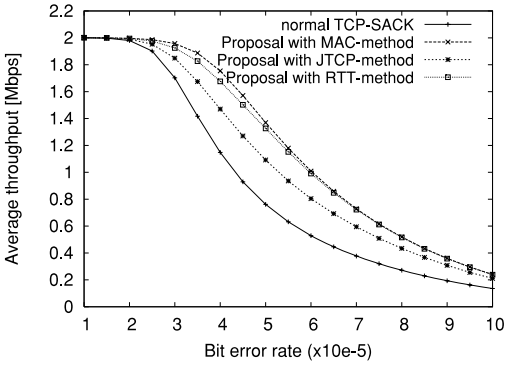


Fig. 7 Average throughputs in the simulation.

during ACK-splitting with this control function. The arrival timing of data packets and values of s_queue_i, r_queue_i, ΔQ_i, and ACK_i are shown. Note that we set the number of ACK packets for the first data packet to 1, because there is no information to determine the congestion level of the uplink.

4. Performance Evaluation with Simulation Experiments

This section presents the results of a simulation that evaluated the performance of our mechanism.

4.1 Simulation Settings

We used ns-2 [27] for the simulation experiments. To evaluate our mechanism over simulated wired and wireless heterogeneous networks, we compared the throughput when our mechanism was installed at the receiver with throughput of the original version of TCP-Reno. The network model is depicted in Fig. 6. It consisted of a sender host, a receiver host, a base station, a wired link connecting the sender and the base station, and a wireless link between the base station and receiver. The bandwidth and propagation delay of the wired link were 10 Mbps and 45 ms. We determined the settings of the wireless link from the UMTS specifications [1], [5], [28], [29] and the model of wireless links reported in [30]. The wireless channels were used independently as download and upload links, and the downlink bandwidth and propagation delay were 2 Mbps and 1 ms. The uplink had a bandwidth of 384 kbps and a propagation delay of 1 ms. We assumed that bit errors occurred with a constant ratio when packets were transmitted on the wireless link. The bit error rate was set between 10^{-5} and 10^{-4}. We modeled the transmission delay on the wireless network as follows: we calculated the packet error rate for the bit error rate and packet size in bits. When at least one bit in the packet had an error, we added 6.16 ms, which is the sum of the round-trip propagation delay of the wireless link and the time needed to send a packet of 1000 bytes on the wireless link, to the transmission delay of the packet. The packet was retransmitted up to three times, according to the default setting of ARQ, and it was discarded if these retransmissions failed, meaning that wireless loss occurred. The base station had a packet buffer of 50 data packets, and congestion loss occurred when the buffer overflowed.

One TCP connection was set between the sender and the receiver, and the sender sent balk data to the receiver for 1000 seconds. We used the SACK option of TCP, since the most operating systems implement it.

4.2 Simulation Results

Figure 7 shows the throughput results. The x-axis indicates the bit error rate of the wireless link, and y-axis is the average throughput of the TCP connection. Results are plotted the four kinds of TCP. One is the normal TCP-SACK, which does not deploy our mechanism at the receiver. The other three are our mechanism using three different methods to distinguish the cause of packet loss (MAC-, JTCP-, or RTT-method as described in Sect. 3.2). The results indicate that the throughputs of all four methods deteriorate with increasing bit error rate of the wireless link. However, our mechanisms perform better than the normal TCP-SACK, and the mechanism with the MAC-method performs the best. The MAC-method increases throughput by up to 94% in comparison with that of TCP-SACK when the bit error rate is around 7×10^{-5}, and the throughput is up to 640 kbps higher than that of normal TCP-SACK when the bit error rate is around 4.5×10^{-5}. The JTCP-method and RTT-method yield a maximum throughput improvement of 61% and 90%, or up to 340 kbps and 530 kbps higher throughput than that of normal TCP-SACK, respectively. We note that the effectiveness of the proposed mechanism depends on the method of distinguishing the cause of packet loss, because it strongly affects the effectiveness of ACK-splitting.

Figure 8 depicts the ACK sending rate, which is de-

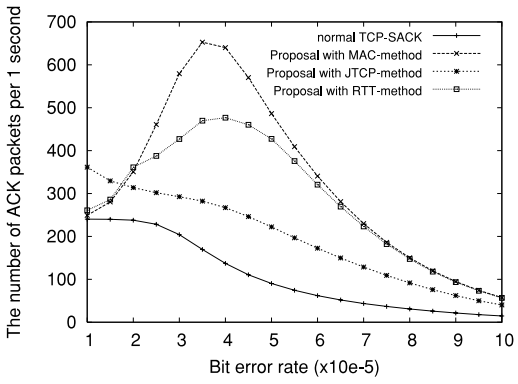


Fig. 8 Comparison of the sending rate of ACK packets.

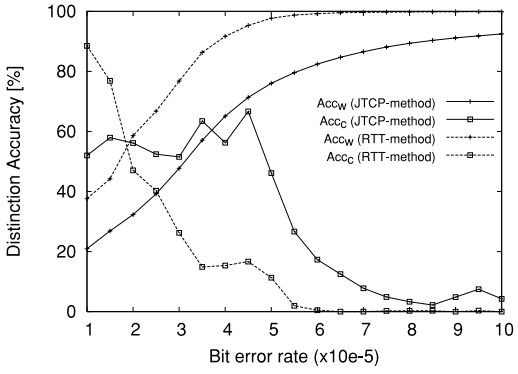


Fig. 9 Distinction accuracies of JTCP-method and RTT-method.

fined as the average number of ACK packets sent from the receiver per second. Because additional ACK packets are generated by using ACK-splitting, our mechanism increases the number of ACK packets. Basically, throughput increases with the number of ACK packets, except when the bit error rate is low. When the bit error rate is low, fewer ACK packets are necessary since normal TCP-SACK does not degrade the throughput. However, the mechanism with the JTCP-method significantly increases the number of ACK packets.

Next we discuss how the method of packet-loss distinction affects the effectiveness and number of ACK packets. Figure 9 depicts Acc_w, the accuracy of detecting wireless losses, and Acc_c, the accuracy of detecting congestion losses. These values are calculated as follows:

$$Acc\_w = \frac{N_{wc}}{N_{wa}} \tag{6}$$

$$Acc\_c = \frac{N_{cc}}{N_{ca}} \tag{7}$$

where N_{wc} is the number of correctly distinguished wireless losses, N_{wa} is the total number of wireless losses, N_{cc} is the number of correctly distinguished congestion losses, and N_{ca} is the total number of congestion losses. Note that there is no data on the accuracy for detecting congestion losses when the bit error rate is higher than 6.5×10^{-5}, since no congestion losses occur.

For wireless losses, the accuracy of the RTT-method of is higher than that of the JTCP-method. Since ACK-splitting is not executed when a wireless loss is not detected correctly, our mechanism becomes less effective if the accuracy of detecting wireless losses becomes small. Consequently, in Fig. 7, the mechanism with the JTCP-method has a lower throughput than those with the MAC-method or RTT-method. On the other hand, because the accuracy of detecting wireless losses with the RTT-method is high enough in the high bit error rate region, ACK-splitting is correctly executed for wireless losses, when necessary. Therefore, the mechanism with the RTT-method is similar in effect to the mechanism with the MAC-method.

Neither the JTCP-method nor the RTT-method accurately detects congestion losses. If congestion loss is mistaken for wireless loss, ACK-splitting is unnecessarily executed, meaning that the number of ACK packets increases unnecessarily. The effect of such a situation can be clearly seen in the low bit error rate region in Fig. 8. In that region, the JTCP- and RTT-methods send out more ACK packets than the MAC-method does, and the performance of MAC-method is the highest among the three. In fact, the degree of throughput improvement is not heavily affected by the accuracy of detecting congestion losses, because the throughput of normal TCP-SACK does not deteriorate much in the low bit error rate region where the throughput is not dependent on whether ACK-splitting is executed or not. Meanwhile, congestion losses hardly occur when the bit error rate is high since wireless losses occur before the buffer overflows. Therefore, the accuracy of detecting congestion losses has little impact on the performance of our mechanism. However, since unnecessary ACK-splitting may make the network more congested, it is desirable for the congestion losses to be detected with high accuracy.

The RTT-method has a very simple algorithm, so its accuracy may be improved by using another algorithm. For example, in [31], the change in packet transmission delay is divided into two types, one reflecting a short-term trend and the other a long-term trend, and packet loss (congestion loss) is predicted with a weighting function dependent on the congestion level. By applying such an algorithm, the RTT-method could distinguish the cause of packet loss more accurately. In this paper, we do not investigate such a modification, so it remains as a future work.

5. Conclusion

In this paper, we proposed a new mechanism to improve the performance of TCP over wired and wireless heterogeneous networks. The mechanism restrains the throughput degradation by increasing the congestion window size faster than the original TCP-Reno by using an ACK-splitting method when wireless losses occur. It needs a modification only to the receiver TCP algorithm. Consequently, it doesn't have the drawbacks of the existing solutions: it can be deployed easily and applied to IP level encrypted traffic, and it preserves TCP's end-to-end principle. Through simulation experiments, we confirmed that the mechanism can improve

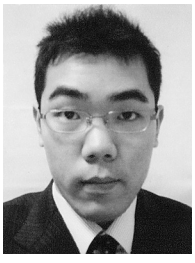
TCP-SACK throughput by up to 94% on wired and wireless heterogeneous networks (assuming UMTS for the wireless network). The simulations also showed that the mechanism is effective even when the receiver can not perfectly distinguish the cause of packet loss.

In the future, we will consider a more accurate method to distinguish the cause of packet loss and confirm its effectiveness in simulation experiments. We also plan to implement the mechanism in the actual wireless network environments and confirm its effectiveness in actual operation.

References

[1] 3GPP, "TS 21.103: Technical specification group services and system aspects; 3rd generation mobile system release 5 specifications," June 2003.
[2] W.R. Stevens, TCP/IP Illustrated, Volume1: The Protocols, Addison-Wesley, 1994.
[3] F. Lefevre and G. Vivier, "Understanding TCP's behavior over wireless links," Proc. Communications and Vehicular Technology, pp.123–130, Oct. 2000.
[4] V. Tsaoussidis and I. Matta, "Open issues on TCP for mobile computing," Wireless Communications and Mobile Computing, vol.2, no.1, pp.3–20, Feb. 2002.
[5] 3GPP, "TS 22.105: Services and service capabilities v6.2.0," June 2003.
[6] 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.
[7] 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.
[8] K. Wang and S.K. Tripathi, "Mobile-end transport protocol: An alternative to TCP/IP over wireless links," Proc. IEEE INFOCOM'98, pp.1046–1053, March 1998.
[9] K. Ratnam and I. Matta, "WTCP: An efficient mechanism for improving TCP performance over wireless links," Proc. Third IEEE Symposium on Computers Communications, pp.74–78, June 1998.
[10] H. Balakrishnan and R.H. Katz, "Explicit loss notification and wireless web performance," Proc. IEEE GLOBECOM Internet Mini-Conferenve, Nov. 1998.
[11] K. Jin, K. Kim, and J. Lee, "SPACK: Rapid recovery of the TCP performance using SPlit-ACK in mobile communication environments," Proc. IEEE Region 10 Conference, pp.761–764, Sept. 1999.
[12] 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.
[13] C. Liu and R. Jain, "Approaches of wireless TCP enhancement and a new proposal based on congestion coherence," Proc. 36th Hawaii International Conference on System Sciences, p.307a, Jan. 2003.
[14] C.P. Fu and S.C. Liew, "TCP Veno: TCP enhancement for transmission over wireless access networks," IEEE J. Sel. Areas Commun., vol.21, no.2, pp.216–228, Feb. 2003.
[15] K. Xu, Y. Tian, and N. Ansari, "TCP-Jersey for wireless IP communications," IEEE J. Sel. Areas Commun., vol.22, no.4, pp.747–756, May 2004.
[16] 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.
[17] A. Capone, L. Fratta, and F. Martignon, "Bandwidth estimation schemes for TCP over wireless networks," IEEE Trans. Mobile Comput., vol.3, no.2, pp.129–143, June 2004.
[18] N. Parvez and E. Hossain, "Improving TCP performance in wired-wireless networks by using a novel adaptive bandwidth estimation mechanism," Global Telecommunications Conference 2004, GLOBECOM'04 IEEE, pp.2760–2764, Dec. 2004.
[19] A.S. Tanenbaum, Computer Networks, fourth ed., ch. 8.6.1, pp.772–776, Prentice Hall, 2002.
[20] 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.
[21] E.S. Chang and R. Taborek, "Recommendation of 10e-13 bit error rate for 10 gigabit ethernet," IEEE802.3 High Speed Study Group July 1999 Plenary Week Meeting, July 1999.
[22] IEEE802 LAN/MAN Standards Committee, "IEEE standard for local and metropolitan area networks: Overview and architecture," Dec. 2001.
[23] H. Balakrishnan, V.N. Padmanabhan, S. Seshan, and R. Katz, "A comparison of mechanisms for improving TCP performance over wireless links," IEEE/ACM Trans. Netw., vol.6, no.5, pp.756–769, Dec. 1997.
[24] V. Jacobson, R. Braden, and D. Borman, "TCP extensions for high performance," Request for Comments (RFC) 1323, May 1992.
[25] D.W. Bailey and B.J. Benschneider, "Clocking design and analysis for a 600 MHz alpha microprocessor," IEEE J. Solid-State Circuits, vol.33, no.11, pp.1627–1633, Nov. 1998.
[26] H. Shimonishi, M.Y. Sanadidi, and M. Gerla, "Improving efficiency-friendliness tradeoffs of TCP in wired-wireless combined networks," Proc. IEEE International Conference on Communications 2005, May 2005.
[27] The VINT Project, "UCB/LBNL/VINT network simulator -ns (version 2)," available from http://www.isi.edu/nsnam/ns
[28] 3GPP, "TS 25.322: Radio link control (RLC) protocol specification v6.2.0," Dec. 2004.
[29] 3GPP, "TR 25.853: Delay budget within the access stratum v4.0.0," March 2001.
[30] A. Gurtov and S. Floyd, "Modeling wireless links for transport protocols," ACM SIGCOMM Computer Communication Review, vol.34, no.2, pp.85–96, April 2004.
[31] L. Roychoudhuri and E. Al-Shaer, "Real-time packet loss prediction based on end-to-end delay variation," IEEE Trans. Network and Service Management (TNSM), vol.2, no.1, Nov. 2005.

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.