Mitigations in TCP Performance for Jitter and Loss in W-CDMA Networks

Hiroshi Inamura,[†‡] Motoharu Miyake[†] and Kenichi Okada[‡]

Summary
We are facing two major cause of performance degradation in TCP traffic over wireless network such as W-CDMA that utilizing link ARQ mechanisms; jitter and loss in streams of segments. For mitigating the degradations, we explored and achieved as follows; 1) Leveraging the link layer optimizations techniques and established standardization, we evaluated the contribution of Eifel for WCDMA network against jitter. As a result, we can recommend the use of STO avoidance mechanism, such as Eifel. Eifel improved the throughput by around 25 kbps at BLER 10% sample, even with unoptimized link parameters. This result will have substantially impact for network operation because if we could assume the presence of STO avoidance mechanism, the network can be optimized more toward link efficiency on the trade-off curve. 2) We proposed an improvement for the Fast Timeout algorithm with loss retransmission mechanism for strengthen in loss recovery. Simulation results yield the following; a sender using the proposed algorithms holds one half of the previous cwnd value and so avoids unnecessary throughput degradation even if duplicate ACKs arrive following a timeout and the retransmitted segment is lost. The proposed algorithm overcomes Fast timeout at most around 33 kbps of improvement in throughput for single set of recovery in our simulation settings.

Key words:
TCP, Wireless, 3G, W-CDMA

1. Introduction

TCP is the glue for the Internet. Tremendous numbers of applications are built upon it. The deployment of 2.5G/3G cellular technologies brought the popularity of TCP based wireless services8). The quality of transmission in the wireless networks that incur bit error can be improved with link layer retransmission. Using TCP on top of the link-layer error recovery can increase end-to-end performance. Therefore link-layer and end-to-end recovery can coexist; this can lead to inefficient interaction between the two layers of ARQ (Automatic Repeat reQuest) protocols[4),13),14)].

The ARQ mechanism can provide the packet service that offers a negligibly small probability of undetected error due to the use of RLC frame retransmission.[1)] However, the diverse interval between arrivals of packets caused by error recovery can lead to an unexpected increase in round trip time (RTT). In this event, the TCP sender experiences a retransmission timeout (RTO) because it has no information about the wireless conditions. We call the diverse interval as jitter.

Several algorithms have been proposed to avoid costly retransmission time-outs. The Eifel algorithm[21)] provide spurious timeout (STO) detection[17)]. STO occurs due to the retransmission ambiguity problem; the sender experiences unnecessary go-back-N retransmission and throughput degradation caused by false congestion control[22)].

In a case of failure of link layer transmission, ARQ gives up the retransmission attempt which leads to the loss of a TCP segment[1)]. Thus, the first unacknowledged segment is discarded, and the receiver acknowledges a series of the next arrived segments as duplicate ACKs. Moreover, handovers in mobile communications can cause the arrival of duplicate ACKs after a timeout[3)]. As a result, these STO detection algorithms, such as the Eifel, only work as conventional TCP[7)].

We are facing two major cause of performance degradation in TCP traffic; jitter and loss in streams of segments. For mitigating the degradations caused by jitter and loss, we define the problem space to explore in this paper as follows; 1) Utilizing our experience for link layer optimizations and established standardization, we will evaluate the contribution of Eifel for W-CDMA network against jitter. The result will have substantial impact for network operation. 2) We evaluate and propose an

[†] DoCoMo Communication Laboratories, USA Inc.
[‡] Department of Information and Computer, Keio University

Manuscript received January 5, 2009
Manuscript revised January 20, 2009

improvement for the Fast Timeout algorithm with loss retransmission mechanism for strengthen in loss recovery.

The rest of this paper is organized as follows: Section 2 overviews W-CDMA link ARQ mechanisms. Section 3 surveys related topics and explains important prior arts including Eifel, Fast Timeout and Loss Retransmission. Section 4 defines our approach. Section 5, we evaluates Eifel on W-CDMA network emulator, Section 6 discusses our proposed improvements for Fast Timeout algorithms and its evaluation. Finally, Section 8 provides a summary.

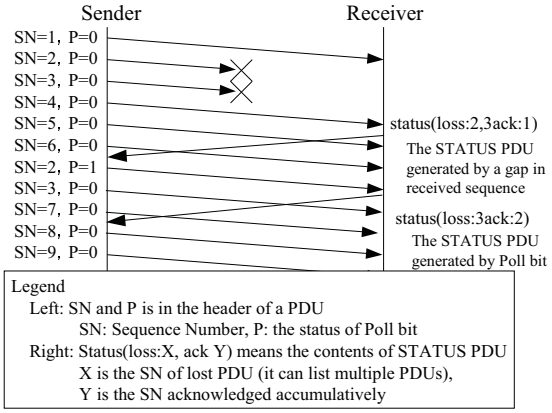


Fig. 1 Retransmission in RLC

2. The Case for Jitter and Loss

We briefly describe how the ARQ causes jitter and loss in the stream of packet9) as a background. First we introduce the L2 ARQ in W-CDMA network, and then discuss the cause of jitter in terms of link utilization and loss of packet in link persistence. Also we will introduce a couple of parameter in link ARQ, namely Timer Status Prohibit and MAX DAT.

2.1 L2 ARQ in W-CDMA network

W-CDMA[10], an IMT2000 standard, was developed by 3GPP (3rd Generation Partnership Project).

W-CDMA system controls transmission error by link layer ARQ. RLC is used as the ARQ protocol in W-CDMA. The RLC protocol works between handset terminal and RNC (Radio Network Controller), an intermediate level node in core network in 3GPP architecture.

When RLC receives an IP packet from the upper layer, the packet is segmented into PDUs (Protocol Data Units), each of which has a 2 octet header and 40 octet payload. The PDU is the unit of acknowledgment and retransmission in RLC. RLC is categorized as ARQ with Selective Repeat. Its retransmission mechanism uses Polling control and Status Report PDU. The protocol interaction of RLC sender/receiver is depicted in Figure 1. The sender can force status reporting (i.e. the return of status PDU from receiver) by setting P(oll) bit in each PDU header. The receiver sends status PDU in the following two conditions:

(1) The receiver finds a gap in the sequence number of received PDUs which means the detection of PDU loss. If PDU(s) are lost, the receiver will see a gap in the numbers (say 1 2 3 5, if 4 is lost).

(2) The received PDU holds the P bit.

Sender controls retransmissions according to the received STATUS PDUs.

The benefits of RLC retransmission comparing only relying to the end-to-end reliability offered by TCP, is two fold. 1) The small PDU size (42 octets) used in RLC makes retransmission more efficient. 2) The response time against feedback from the receiver is smaller than is possible with TCP's end-to-end feedback, since only the wireless subnetwork is involved.

2.2 The trade-off between jitter suppression and link utilization

The delay in packet arrival depends on the number of PDU retransmissions. If the jitter increases rapidly, TCP interprets it as packet loss and triggers time out spuriously. To avoid this retransmission, we have to suppress the jitter.

For suppressing the delay-jitter in the link layer of W-CDMA, we have to set the RLC parameters appropriately, including Timer Status Prohibit (TSP). TSP defines the delay imposed on the receiver before it can issue a STATUS PDU. For example, if this value is set to 0, a STATUS PDU is immediately generated and issued if PDU loss is detected. Setting TSP to larger than 0 makes it possible for a single STATUS PDU to cover multiple losses. If TSP is too large, PDU flow stops due to a lack of acknowledgments and jitter increases.

The trade-off between the suppression of jitter and link layer utilization impacts the optimal TCP throughput. When TSP is small, many STATUS PDUs are generated and loss notification is quick, while the STATUS PDUs themselves and multiple retransmissions of the same PDU decrease the effective link bandwidth, vice versa. A large TSP causes deferred notification of loss which increases the retransmission interval and results in large jitter, which degrades RTO and TCP throughput.

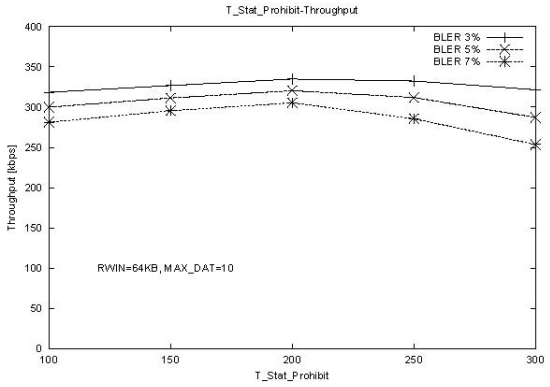


Figure. 2 Throughput versus Timer Status Prohibit (using Opnet simulation)

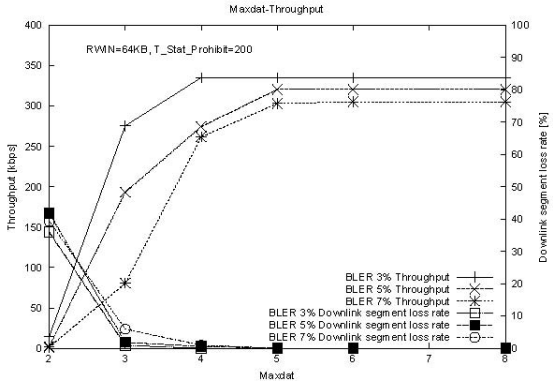


Figure. 3 Throughput and Down link segment loss rate vs MAX DAT (using Opnet simulation)

Figure 2 shows TCP throughput versus TSP [1]. As discussed above, small TSP causes low link utilization which degrades throughput falls (see the left side of Figure 2). The throughput degradation seen on the right side of Figure 2 is caused by the spurious timeouts triggered by the delay-jitter. So if all of TCP implementations have a way of STO avoidance mechanism, the optimum point will be shifted toward the right side that provides larger link efficiency.

2.3 Link Persistence and Segment Loss

Persistence in link layer retransmission involves a trade-off between IP packet loss and the efficiency of link utilization. We define persistence as the maximum number of PDU retransmission attempts allowed by the RLC operation; the MAX DAT parameter defines the maximum number of retransmission attempts for a single PDU.

[1] Using original simulator implemented RLC on OPNET with BSD RTO emulation

To show the relation between persistence and TCP throughput, Figure 3 plots TCP throughput versus MAX DAT, the parameter that defines the maximum number of retransmission attempts for each PDU. From the result shown in Figure 3, we found that MAX DAT should be at least 5 to achieve BLER (BLock Error Rate) 0 = 10% in the simulated network.

3. Related Works

3.1 Suppressing Spurious Timeout

Several algorithms have been proposed to avoid the effects of costly retransmission timeout [21)19)]. The Eifel algorithm [21)] with the TCP time stamp option can identify if the acknowledgment is in response to the original segment or the retransmitted segment. The time stamp option is standardized as RFC1323 and is implemented in most operating systems. Only the sender need implement the Eifel algorithm. If a sender running the Eifel algorithm detects Spurious Time-out (STO), it reverts to the cwnd and the ssthresh to avoid unnecessary retransmission and throughput degradation. Moreover, it can adjust parameters for setting the RTO, to prevent more RTO events. Eifel is already standardized but needs real world evaluation.

Gurtov [2)] discusses link layer characteristics of GSM and GPRS including long sudden delays. It focuses on link level retransmission and resource management. It also suggests applicability of Eifel; however it lacks key insights such as trade-off between link utilization and transport layer performance.

M-TCP [5)] discusses and achieves performance improvements for problem such as high bit error rate and long disconnection cased by wireless link characteristics. Still, comparing server side approach such as Eifel, its split-connection approach requires extra complexity in either mobile host or base station when to apply to cellular system that is limiting factor for its wider acceptance.

3.2 Coping with segment loss and RTO caused by ARQ

All of the above STO detection algorithms provide only conventional TCP responses (i.e. keep the slow start phase and transmit nothing until an acceptable ACK arrives) [7)11)]. To extend the conventional TCP function so that it can handle timeout, the fast timeout algorithm [20)] uses the duplicate ACKs raised after a timeout as implicit segment loss information. It allows the sender to directly

switch from the slow start algorithm to the fast recovery algorithm. It then transmits new segments allowed by the value of new congestion window upon the arrival of the duplicate ACKs, even if the sender times out. However, it has no function to detect an oversight of a loss of first unacknowledged segment which was retransmitted by a timeout.

In order to strengthen TCP against segment loss, Lin and Kung originally proposed a loss retransmission algorithm using TCP's ACK-clock[6]. It retransmits the first unacknowledged segment if the number of duplicate ACKs under the fast retransmit/recovery phase reaches the number of outstanding segments plus DupThresh. Originally it is invented for enhancement only for Reno and lacks insights for applying other TCP enhancement mechanisms.

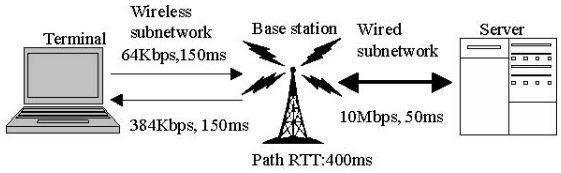


Figure. 4 Simulated Network

BLER	1%	2%	5%	10%
FreeBSD2.2.8	0	0	0	1
FreeBSD4.5	0	0	1	1
Linux2.4+Eifel	0	1	0	3
Linux2.4-Eifel	0	0	1	3
Solaris8	5	2	6	8
Linux2.2	7	6	6	6
Windows XP(SP2)	0	0	0	0

Figure. 5 The number of incidents of RTO vs BLER (using W-CDMA emulator)

4. Approach: Mitigations in TCP Performance for Jitter and Loss

As we introduced the W-CDMA link mechanisms and its characteristics, we are facing two major cause of performance degradation in TCP traffic; jitter and loss. For mitigating the degradations caused by jitter and loss, we define the problem space to explore in this paper as follows.

Utilizing the link layer optimization technique, established standardization, we will evaluate the contribution of Eifel for W-CDMA network against jitter. The result will have substantial impact for network operation because as discussed in 2.2, if all of TCP implementation could have a way of STO avoidance mechanism, the network can be optimized more on link efficiency. We set our simulation environment as recreating for both unoptimized network operation and users in low quality coverage.

Based on the survey from prior arts, we evaluate and improve the Fast Timeout algorithm with loss retransmission mechanism for strengthen in loss recovery. Lost recovery is just a one reason for RTO in Internet[6], it takes 5% of all RTO occasions. It can be interpreted that the percentage expected when users access Internet via good connection quality of wireless link.

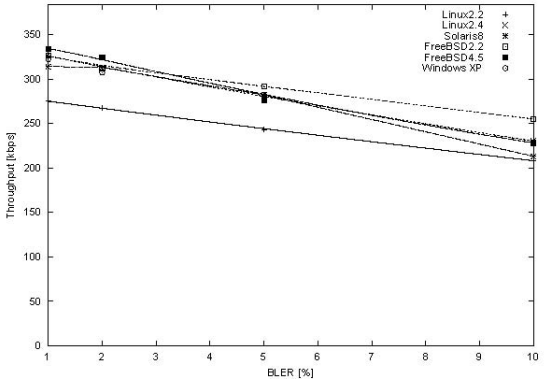


Figure. 6 Throughput vs BLER (according to W-CDMA emulator)

5. RTO on W-CDMA and Eifel evaluation

We evaluated TCP time out behavior of a representative operating system (OS) using an emulator.

5.1 Network assumptions

Figure 4 shows the model of W-CDMA network. We assume the RTT of 300 ms in the wireless subnetwork, and distribute it equally between the up and down links; so that each link has 150 ms delay[1]. The wired subnetwork is high speed and has a bandwidth of 10 Mbps; it has a processing delay of 100 ms, so the end-to-end RTT is 400 ms in our scenario. The bandwidth of the down link is 384 kbps while the uplink offers 64 kbps. The traffic is a 2 Mbyte TCP stream transfer from the server to the terminal via the base station. While the W-CDMA specification

[1] The latency is estimated from actual measurement 300 ms by separating equally for up/down link, since the main cause for short frame is processing delay.

permits a bandwidth allocation of up to 2 Mbps to each terminal, we use 384 kbps to better match the performance of existing commercial services. The wireless channel status is described as the loss rate per PDU, BLER (BLock Error Rate). We examine the PDU error rate in the range of 0 ~ 10%. In this work, we assume an ideal function of physical layer channel coding mechanism, such as turbo coding that is actually implemented in WCDMA. As the result, the error model RLC layer handles is uniform random distribution. For continuous error that caused by long channel blockade in hard handover is not considered in the simulation, if otherwise noted.

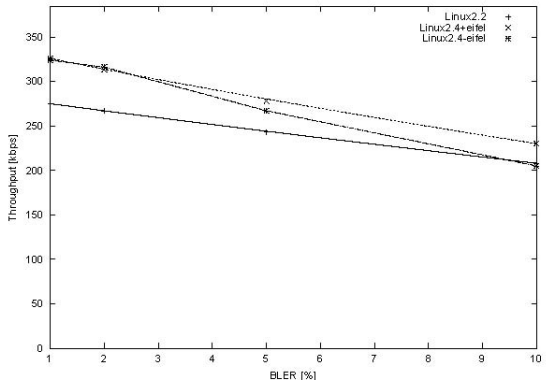


Fig. 7 Contribution of Eifel in Throughput vs BLER in Linux

5.1.1 Emulation Environment

We used a W-CDMA emulator that re-creates the behavior of wireless and wired networks, especially RLC in the link layer. The emulator reproduces the dynamic delay by actually executing RLC, and adds static delay to represent the wired network when forming the pseudo W-CDMA network. Since it recreates the whole W-CDMA network, the emulator provides two Ethernet ports for connecting a terminal and a server. We can connect real PCs to the emulator to run real applications. For generating TCP traffic, we used ftp command/server.

5.2 Evaluation of retransmission time out characteristics in different TCP implementations

We investigated the difference among three TCP implementations and selected one for studying the time out behavior in more detail. That is, we found the TCP implementation that allowed jitter to trigger the greatest number of TCP RTO events.

The behavior of TCP time out depends on how the function is implemented in the OSs. Most commercial OSs do not disclose details of their time out algorithm, so hardware tests are needed to study real OS behavior. Accordingly, we investigated the TCP time out behavior against delay-jitter using a WCDMA emulator with the same mechanisms as the software simulator.

We selected Linux-2.2, FreeBSD2.2.8, Solaris8 and Windows XP (SP2) as the sender hosts, since they are often used in Internet servers. We used FreeBSD2.2.8 as the receiver host. In this experiment, we made the link persistence high, to intentionally make jitter large; packet loss was prevented by setting MAX DAT to 10. FreeBSD is the only OS that does not implement SACK, but the absence of SACK is not the issue. This is because the possibility of IP packet loss is negligibly small due to the large MAX DAT value; the possibility of SACK being activated was virtually none.

To select the OS that was most aggressive in terms of triggering retransmission by jitter, we counted the number of TCP RTO events by setting TSP to 500 ms. which is larger than RTT, to increase the possibility of jitter. FTP was used to transfer 1 Mbyte for generating a bulk TCP flow; the sender and receiver OSs used 64 Kbyte buffers. We counted the number of RTO events for each OS during 10 FTP transfers at 4 BLER values; the results are plotted in Table 5. Linux2.2 exhibits RTO events at all BLER values examined. This is also true for Solaris but the number of events is fewer. FreeBSD prevents any RTO event, except at the BLER value of 10%.

Table 5 shows that Windows XP and FreeBSD are resilient against jitter. Linux2.4 changed RTO implementation to conform BSD algorithm that leads its behavior more conservative as BSD does. In addition to that, the retransmission timers have different settings. FreeBSD has larger minimum value than Linux2.2[1], which makes it less aggressive than Linux. Ludwig[15] pointed out that BSD's retransmission timer is restarted by the arrival of a new acknowledgment, not the transmission of the original segment, so the timer is offset by roughly one RTT, which makes BSD less aggressive with regard to the RTO trigger. For Windows XP, The RTO outcome is similar to the FreeBSD as in our investigation that suggests it implements BSD behavior[18]. Figure 6 shows TCP throughput under the same conditions as described above. The results show that the throughput is decreased by the RTO events, especially in Linux2.2. Comparing the difference between FreeBSD2.2.8 and 4.5 is small due to the fact that the incidents of RTO itself are small.

The implementation of Eifel in Linux makes difference. The number of incidents of RTO is almost the same as in Table 5 regardless of Eifel; however the Figure 7 shows that the Eifel improved the throughput by around 25 kbps at BLER 10% sample. The closer look of the traces we take from the experiment, all of RTO is detected

[1] For minimum RTO, Linux2.2 is200ms and FreeBSD2.2.8 is 1s.

and responded by Eifel to cancel the congestion avoidance behavior.

6. Loss retransmission algorithm combined with fast timeout

6.1 Proposed Algorithm

In the case of a retransmission timeout, the sender is not well handled in a conventional TCP using the loss retransmission algorithm, because it can only receive a limited number of the duplicate ACKs in response to the outstanding segments. Thus, the loss of the retransmitted segment raised by the retransmission timeout leads to exponential back off with the smallest values of cwnd and ssthresh. As a result, the sender needs additional time to recover the congestion window size, which leads to severe throughput degradation.

To avoid the above worst case, the fast timeout algorithm[20] enables the retransmitted segment loss to be confirmed using the number of outstanding segments and duplicate ACKs.

If the number of duplicate ACKs, dupacks, equals the number of outstanding segments when loss recovery started, FlightSize, plus DupThresh, the sender retransmits the first unacknowledged segment immediately. It clearly shows that the duplicate ACKs represent not only the response of the original segment, but also the response of the next transmitted segment by the fast recovery algorithm. Note that DupThresh in step (7) is a conservative response to out-of-order segments. After that, the sender just waits for an acceptable ACK while sending the next new segment in response to the duplicate ACK. As a result, the sender can avoid the exponential backoff caused by failure to identify retransmitted segment loss, and increases the probability of segment retransmission without entering costly retransmission timeout.

Step (1) RTO timer expires:
Store SND.HIGH (highest sequence number),
FlightSize (the number of outstanding segments),
cwnd_prev ← cwnd and
ssthresh_prev ← ssthresh
Retransmit the 1 st unacknowledged segment

Step (2) Wait for the arrival of either an acceptable or a duplicate ACK:
Update the variable dupacks and proceed to step (3)

Step (3) If an acceptable ACK has arrived
then proceed to step (DONE),
else if dupacks < DupThresh
then return to step (2),
else (dupacks equals DupThresh)
proceed to step (4).

Step (4) Resume transmission from the top
Suppress the fast retransmit, and set
SND.NXT ← SND.MAX

Step (5) Make the RTT estimation more conservative:
Set
SRTT ← 2 SRTT,
recalculate the RTO, and restart retransmission timer.

Step (6) Leave slow start and move to fast recovery algorithm:
Set the parameters as follows:
ssthresh ← max(cwnd_prev/2, 2 SMSS)
cwnd ← ssthresh + SMSS DupThresh

Step (7) If the sender is satisfied with the condition
dupacks == FlightSize + DupThresh
then retransmit the first unacknowledged segment,
else waits for acceptable ACK while sending the next
new segment in response to duplicate ACK.

Step (DONE) Leave the loss retransmission and fast timeout algorithms

Figure 8 shows the proposed loss retransmission algorithm in collaboration with the fast timeout algorithm[16].

Step (7) If the SACK block reports as follows:
Right edge in SACK block > SND.HIGH
then transmit the first unacknowledged segment again,
else transmit the next new segment by the duplicate ACK arrival.

Figure 9 The loss retransmission algorithm with SACK option

7. Performance evaluation

The proposed loss retransmission algorithm may not work well in all cases of lost outstanding segments or acknowledgments. In these case, the sender will retransmit the first unacknowledged segment later than the desired timing. In order to avoid this situation, the proposed algorithm can use the SACK option. If SACK is available, the sender can use the correct timing from internal state, without resorting to any heuristics.

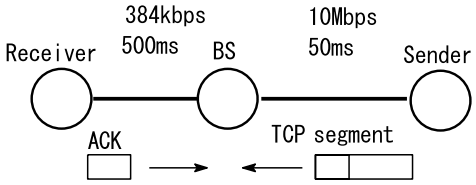


Figure 10 Simulation model

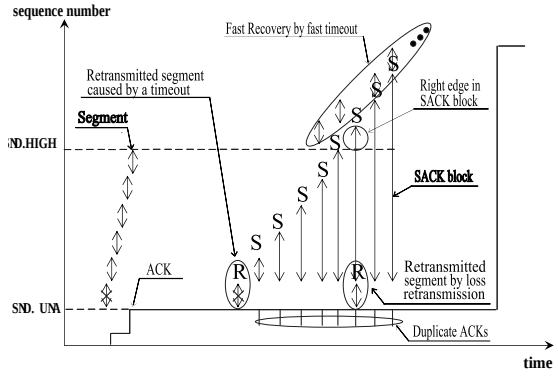


Figure 11 2nd RTO segment retransmission using SACK option

Figure 9 shows the proposed loss retransmission algorithm with SACK option. The difference from the previous basic loss retransmission algorithm is only the evaluation performed in step (7). If the right edge in a SACK block advances the value SND.HIGH, the sender retransmits the first unacknowledged segment immediately. After that, the sender waits for an acceptable ACK while sending the next new segment in response to a duplicate ACK.

Figure 11 illustrates the relationship between the SND.HIGH and the right edge in a SACK block in a sender-side time-sequence graph. Symbols "R" and "S" show the retransmitted segment and the SACK block in duplicate ACK, respectively. In Fig. 11, the beginning of the six outstanding segments is lost, and the retransmitted segment raised by RTO is lost again. Next, the duplicate ACK with the SACK block arrives at the sender. The right edge in the SACK block advances the value SND.HIGH, when the sender receives the 6th duplicate ACK. The sender then retransmits the first unacknowledged segment immediately. In this case, there is no outstanding segment or acknowledgment loss, so that the sender using the SACK enhanced algorithm retransmits the first unacknowledged segment using the same timing as the basic loss retransmission algorithm.

We examined both the time-sequence and the sender state variables in the face of duplicate ACKs arrival following a timeout and retransmitted segment loss.

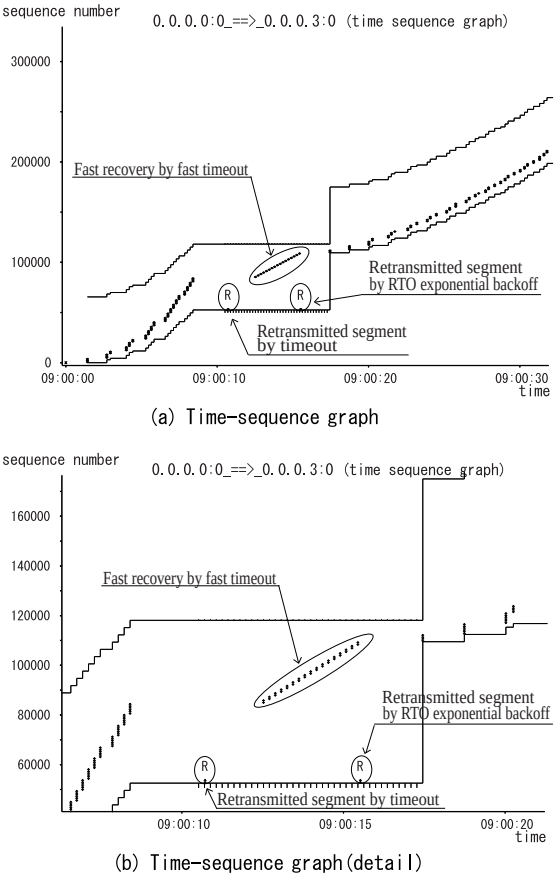
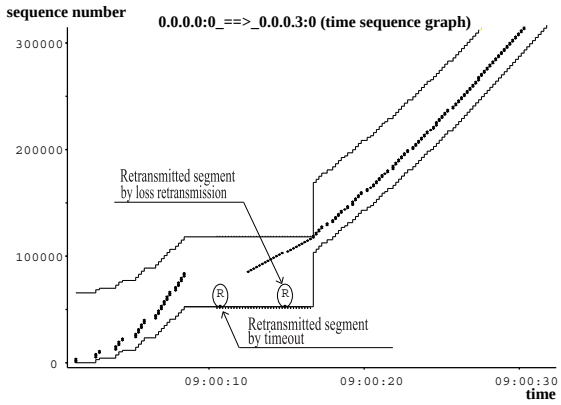


Figure 12 The TCP's time-sequence and sender state variables for the fast timeout algorithm

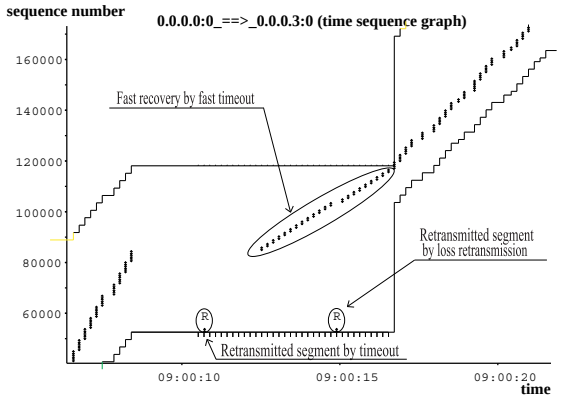
7.1 Simulation model

We implemented the fast timeout and proposed loss retransmission algorithms on the *ns*2 simulator (ns-2.26) developed by the VINT project [12]. Figure 10 shows the topology used in our experiments. The sender is connected to the BS via a 10 Mbps wired link with 50 ms delay, and the receiver is connected to the BS via a 384 kbps wireless link with 500 ms delay. The TCP segments are transmitted

from the sender to the receiver. The BS has enough queue depth and does not drop any original outstanding segments.



(a) Time-sequence graph



(b) Time-sequence graph(detail)

Figure 13 The TCP's time-sequence and sender state variables for the proposed algorithms

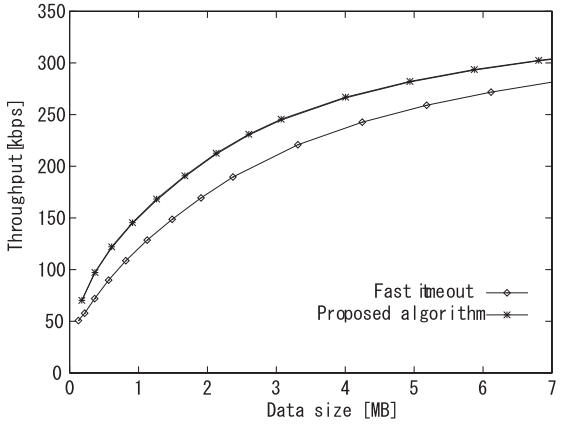


Figure 14 Throughput comparison between the fast timeout and the proposed algorithm

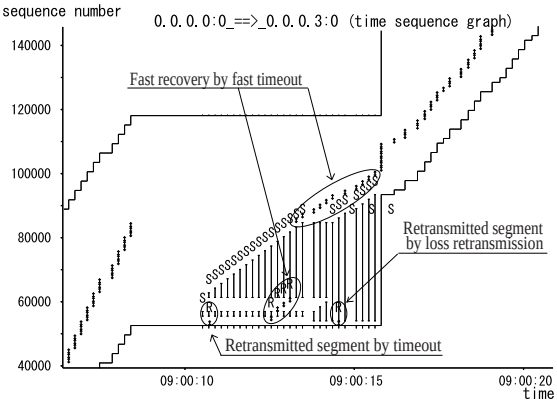


Figure 15 The proposed TCP's time-sequence graph (SACK)

7.2 Evaluation for the proposed loss retransmission algorithm

Figure 12 shows the conventional TCP's time-sequence graph and sender state valuables for the fast timeout algorithm. In Fig. 12 (a), the sender faced the retransmitted segment that caused by the RTO at 9:00:10.7. The conventional TCP has no function that can handle the arrival of the duplicate ACKs after a timeout without the next segment transmission using the fast timeout algorithm until the 2nd RTO. Accordingly, it has to wait a long time, from 9:00:10.7 to 9:00:15.4, following the exponential backoff algorithm. According to RFC2581, the ssthresh is set as

$$ssthresh \leftarrow \max(cwnd/2, 2)$$

$$= 2,$$

so the sender enters the congestion avoidance phase directly when the sender receives the acceptable ACK at 9:00:16.6. Figure 12 (b) shows the sender state variables, cwnd and ssthresh. The ssthresh is set to 2 at 9:00:15.4, the sender transmits a limited number of segments after that. As a result, the sender also degrades the throughput.

In contrast, Fig. 13 shows the time-sequence graph and sender state valuables for the proposed algorithms. It is shown that the sender detects the retransmitted segment loss by examining the 25th duplicate ACK at 9:00:14.7, and it retransmits the unacknowledged segment immediately. Next, the sender waits for an acceptable ACK while sending the next new segment in response to the duplicate ACK. Thus, the sender switches from the fast timeout algorithm to congestion avoidance and transmits a series of segments. In this case, the sender holds one half of the previous cwnd value at 9:00:16.6, see in Fig. 13 (b). This allows it to achieve better throughput than conventional TCP.

Figure 14 shows relationship between data size and throughput among the fast timeout and the proposed algorithms, and it contains the scenario in Figs. 12, and 13. The throughput is obtained by doing division of the amount of data shown in the x-axis in transmission time included in the slow start and the loss retransmission periods. In Fig. 14, the fast timeout algorithm needs excessive data size to recovery the throughput. Then, it spends much time for sending data rather than the others after the loss retransmission. In contrast, the throughput of the proposed algorithms is superior to the fast timeout algorithm. The proposed algorithm overcomes Fast timeout at most around 33 kbps of improvement in throughput for single set of recovery in our simulation settings. Moreover, the proposed algorithm achieves the performance equivalent to the proposed algorithm can with SACK option without any additional overhead and receiver modification in this case.

Figure 15 shows the time-sequence graph for the proposed loss retransmission algorithm with SACK option under the multiple segment loss; the retransmitted segment is also lost at 9:00:10.7. It is shown that the proposed algorithm can retransmit the unacknowledged segment since the right edge of the SACK block in the sender advances the value SND.HIGH at 9:00:14.5. Moreover, the sender retransmits it at the correct timing even if some outstanding segments are lost. As a result, a sender using the proposed algorithms holds one half of the previous cwnd value and so avoids unnecessary throughput degradation.

8. Conclusion

The coexistence of ARQ and TCP can lead to inefficient interaction. We are facing two major cause of performance degradation in TCP traffic; jitter and loss.

For mitigating the degradations caused by jitter, we recommend the use of STO avoidance mechanism, Eifel in two ways. First, it is established as Internet standard. Second, our evaluation using actual TCP implementation on FreeBSD, Linux, and Solaris showed clear improvements on throughput. The result will have substantially impact for network operation because if all of TCP implementation could have a way of STO avoidance mechanism, the network can be optimized more on link efficiency. Eifel improved the throughput by around 25 kbps at BLER 10% sample, even with unoptimized link parameters.

For mitigating the degradations caused by segment loss, we proposed an improvement for the loss retransmission algorithm for duplicate ACK arrival following a timeout. The proposed algorithm allows careful retransmission against failure to identify the first unacknowledged segment loss. Simulation results yield the following conclusion:

A sender using the proposed algorithms can overcome the loss of a re transmitted segment due to RTO.

A sender using the proposed algorithms holds one half of the previous cwnd value and so avoids unnecessary throughput degradation even if duplicate ACKs arrive following a timeout and the retransmitted segment is lost.

The proposed algorithm overcomes Fast timeout at most around 33 kbps of improvement in throughput for single set of recovery in our simulation settings.

Acknowledgment

The authors would like to thank Mr. Daikichi Osuga for his advice and comments on this manuscript.

References

[1] 3GPP: 3G TS 25.322 v.3.5.0, RLC Protocol Specification (2000).

[2] A. Gurtov: Effect of Delays on TCP Performance, *In Proceedings of IFIP Personal Wireless Conference* (2001).

[3] A. Gurtov, M. Passoja, O. Aalto, and M. Raitola: Multilayer Protocol Tracing in a GPRS Network, *In Proceedings of the IEEE Vehicular Technology Conference (VTC'02)* (2002).

[4] Bai, Y., Ogielski, T. and Wu, G.: Interactions of TCP and Radio Link！ ARQ Protocol, *VTC '99* (1999).

[5] Brown, K. and Singh, S.: M-TCP:！ TCP for Mobile Cellular Networks, *ACM Computer Communication Review*, Vol. 27, No. 5 (1997).

[6] D. Lin and H. T. Kung: TCP Fast Recovery Strategies: Analysis and Improvements, *In Proceedings of IEEE INFOCOM 98* (1998).

[7] E. Blanton and M. Allman: Using TCP DSACKs and SCTP Duplicate TSNs to Detect Spurious Retransmissions, draft-blanton-dsack-use-02.txt (2002).

[8] H. Inamura, G. Montenegro, R. Ludwig, A. Gurtov, F. Khafizov: TCP over Second (2.5G) and Third (3G) Generation Wireless Networks, RFC3481 (2003).

[9] H. Inamura, T. Ishikawa, O. Takahashi, H. Nakano, and H. Shigeno: Impact of Layer Two ARQ on TCP Performance in W-CDMA Networks, *Proceedings of IEEE ICDCS*, pp. 284–291 (2004).

[10] Harri Holma and Antti Toskala(eds.): *WCDMA for UMTS, Revised Ed.*, Wiley (2001).

[11] J.C.R. Bennett, C. Partridge, and N. Shectman: Packet Reordering is Not Pathological Network Behavior, *IEEE/ACM Transactions on Networking*, Vol. 7, No. 6 (1999).

[12] K. Fall and K. Varadhan: ns Note and documentation, *The VINT Project (UC Berkeley, LBL, USC/ISI, and Xerox PARC)* (2003).
[13] Karn, P.: Advice for Internet Subnetwork Designers (2003). internet draft, draft-ietf-pilc-link-design-13.txt.
[14] Khafizov, F. and Yavuz, M.: Running TCP over IS-2000, *International Conference of Communications*, IEEE (2002).
[15] Ludwig, R. and Sklower, K.: The Eifel Retransmission Timer, *Computer Communication Review*, Vol. 30, No. 3, pp. 17–27 (2000).
[16] M. Miyake: Responding to Duplicate ACKs after a Timeout in TCP, *TECHNICAL REPORT OF IEICE* (2004).
[17] M. Miyake, H. Inamura and O. Takahashi: TCP Enhancement using Spurious Timeout Detection and Congestion Window Control Algorithm, *8th International Workshop on MoMuC 2003* (2003).
[18] MacDonald, D. and Barkley, W.: Microsoft Windows 2000 TCP/IP Implementation Details (2000). White Paper, http://www.microsoft.com/technet/itsolutions/network/deploy/depovg/tcpip2k.mspx.
[19] P. Sarolahti and M. Kojo: Forward RTO-Recovery (F-RTO): An Algorithm for Detecting Spurious Retransmission Timeouts with TCP and the Stream Control Transmission Protocol (SCTP), RFC4138 (2005).
[20] R. Ludwig: Responding to Fast Timeouts in TCP, draft-ludwig-tsvwgtcp-fast-timeouts-00.txt (2002).
[21] R. Ludwig and M. Meyer: The Eifel Detection Algorithm for TCP, RFC3522 (2003).
[22] R. Ludwig and R. H. Katz: The Eifel Algorithm: Marking TCP Robust Against Spurious Retransmission, *SIGCOMM Computer Communication Review*, Vol. 30, No. 1 (2000).

Hiroshi Inamura has been working for NTT DoCoMo, Inc. since 1999. He joined DoCoMo Communications Laboratories USA in 2006. His research interests are in the area of Networking including IP architecture and wireless access systems. He participated in the IETF and OMA standardization activities and he received an achievement award from the Information Processing Society of Japan for his contribution to the standardization of mobile multimedia protocol in 2004. From 1994 to 1995, he was a visited researcher in the Department of Computer Science, Carnegie Mellon University. He received B.S. and M.S. degree in Keio University, Japan. He is a member of IPSJ, IEICE, and ACM.

Motoharu Miyake was born in 1973. He received the B.S. and M.S. degrees in electronic engineering from Tokyo University of Engineering, Tokyo, Japan in 1995 and 1997, respectively. He received the Ph.D degree in electrical and electronic engineering from Tokyo Institute of Technology, Tokyo, Japan in 2000. Since 2000, he has been working at NTT DoCoMo, Inc., Yokosuka, Japan. His research interests include a transport protocol for wireless communication and a home network. He is a member of IEICE.

Kenichi Okada received his B.S., M.S. and Ph.D. in instrumentation engineering from Keio University, in 1973, 1975, and 1982, respectively. He is currently a professor in the Department of Information and Computer Science at Keio University. His research interests include CSCW, groupware, human computer interaction, and ubiquitous computing.
He has published 60 journal papers, 70 international conference papers, and 13 books entitled "Collaboration and Communication", "Designing Communication and Collaboration Support Systems", "Introduction to Groupware" and so on. He is a member of IEEE, ACM, IEICE and IPSJ. He was a chair of SIGGW, a chief editor of IPSJ Journal, a chief editor of IPSJ Transactions, and an editor of IEICE Transactions.
Dr. Okada received the IPSJ Best Paper Award in 1995 and 2000, the IPSJ 40th Anniversary Paper Award in 2000, IPSJ Fellow in 2002 and the IEEE SAINT Best Paper Award in 2004.