社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2004－ＭＢＬ－29 (31)
2004／5／14

# Responding to Duplicate ACKs after a Timeout in TCP

Motoharu MIYAKE

Multimedia Laboratories, NTT DoCoMo, Inc.,
3-5 Hikarinooka, Yokosuka, Kanagawa, 236-8536, Japan
Tel: +81-46-840-3310, Fax: +81-46-840-3788
E-mail: miyake@mml.yrp.nttdocomo.co.jp

**Abstract** In a case of handovers or expiration of auto repeat request (ARQ) retransmission in mobile communications, a fast timeout algorithm allows new segment transmission using the fast recovery algorithm as duplicate ACKs following a timeout are implicit segment loss information. However, it is not well handle the retransmitted segment that caused by the retransmission timeout (RTO), and leads to the 2nd RTO. Then, this paper proposes an algorithm based on a loss retransmission algorithm. The proposed algorithm suggests careful retransmission against an oversight of a loss of first unacknowledged segment which was retransmitted by the timeout. The results using *ns*2 simulator show that the proposed algorithm can avoid 2nd RTO and throughput degradation more effectively than the conventional TCP.
**Key words**: TCP, Fast timeout, Loss retransmission, RTO, Mobile Communications

# タイムアウト後のduplicate ACK 到着問題への TCP 改善提案

三宅 基治

NTTドコモ マルチメディア研究所
〒239-8536 神奈川県 横須賀市 光の丘 3-5
Tel: 046-840-3310, FAX: 046-840-3788
E-mail: miyake@mml.yrp.nttdocomo.co.jp

**あらまし** fast timeout アルゴリズムは，移動通信網を含む TCP データ通信でのハンドオーバーや ARQ (Auto repeat request) 未完了時に発生が想定されるタイムアウト後の duplicate ACK 到着に対して，fast recovery アルゴリズムの適用を可能とする．しかし，再送タイムアウト (RTO: Retransmission TimeOut) により送信されたセグメントのロスに対しては，従来の TCP と同様に 2 度目の RTO が発生し，ウインドウサイズを過剰に小さくしてしまう問題が発生する．そこで本稿では，RTO に伴い送信される再送セグメントのロス耐性を強化する loss retransmission アルゴリズムを提案し，不要な 2 度目のタイムアウトを回避する．*ns*2 を用いたシミュレーションにより，提案アルゴリズムの利用によって従来の TCP よりも効率的なデータ伝送が可能となることを明らかにする．
**キーワード**: TCP, Fast timeout, Loss retransmission, 再送タイムアウト, 移動通信

## 1 Introduction

Many communication bearers transfer data as TCP streams. In the case of Internet access over wireless links, such as cellular phones, many more research issues remain to be resolved before a truly effective service can be realized.

Wideband code division multiple access (WCDMA) and general packet radio service (GPRS) are well known 3G and 2.5G mobile communication standards, respectively. Both use the radio link control (RLC) protocol, a selective repeat and sliding window auto repeat request (ARQ) scheme [1]. The ARQ mechanism can provide a packet service that offers a negligibly small probability of undetected error due to the use of RLC frame retransmission [2]. However, the delay jitter 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.

Several algorithm have been proposed to avoid costly retransmission timeouts. The Eifel algorithm [3] and the forward RTO recovery (F-RTO) algorithm [4] provide spurious timeout (STO) detection [5]. STO occurs due to the retransmission ambiguity problem; the sender experiences unnecessary go-back-N retransmission and throughput degradation caused by false congestion control [6]. These algorithms monitor an acceptable ACK's timestamp or a series of acknowledgments to avoid the retransmission ambiguity problem.

In a case of failure of link layer transmission for some period of time or for some number of packets in a base station (BS), ARQ may give up the retransmission attempt which leads to the loss of a TCP segment [2]. 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 [7]. As a result, these STO detection algorithms, such as the Eifel and F-RTO, only work as conventional TCP [8].

To extend the conventional TCP function, the fast timeout algorithm [9] uses the duplicate ACKs raised after a timeout as implicit segment loss information. It 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 is not well handle the retransmitted segment that caused by the retransmission timeout (RTO), and leads to the 2nd RTO.

Then, this paper proposes an algorithm based on the loss retransmission algorithm [10]. The proposed algorithm establishes efficient retransmission against the failure to identify the loss of the first unacknowledged segment that was retransmitted due to timeout. Moreover, it needs only small modification of the sender side TCP, so they can support existing receiver side equipment, such as PCs, PDAs, and Internet access cellular phones without any modification. *ns*2 simulations show that the proposed algorithms can avoid this throughput degradation more effectively than the conventional TCP.

## 2 Related works

Several algorithms have been proposed to avoid the effects of costly retransmission timeout [3], [4]. The Eifel algorithm [3] with the TCP timestamp option can identify if the acknowledgment is in response to the original segment or the retransmitted segment. The timestamp 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 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.

Sarolahti proposed the F-RTO algorithm [4]. It sends two new segments in response to the first acceptable ACK, and monitors the response as part of STO detection. If it receives the second acknowledgment as a series of acknowledgments, it posits an STO event and reverts to the original `cwnd` and `ssthresh` as in the Eifel algorithm. Implementing the F-RTO algorithm needs only sender side modification; no TCP options or receiver side modifications are needed.

If the segment loss is generated by handover or expiration of ARQ retransmission, the sender receives duplicate ACKs following a timeout. 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) [8], [11]. To extend the conventional TCP function so that it can handle timeout, the fast timeout algorithm [9] 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 [10]. 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. 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 backoff 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.

# 3 Proposed loss retransmission algorithm after a timeout

To avoid the above worst case, the fast timeout algorithm [9] enables the retransmitted segment loss to be confirmed using the number of outstanding segments and duplicate ACKs.

Figure 1 shows the proposed loss retransmission algorithm in collaboration with the fast timeout algorithm. 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.

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 the connection between sender and receiver permits the SACK option, the right edge in the SACK block or the score-board in the sender can show the correct timing clearly without resorting to any heuristics.

Figure 2 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

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 $1st$ 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
      $SRTT \leftarrow 2 \times SRTT$,
  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, $2 \times SMSS$)
  cwnd ← ssthresh + $SMSS \times$ 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 1: Proposed loss retransmission algorithm

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.

> 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 2: The loss retransmission algorithm with SACK option

Figure 3 illustrates the relationship between the SND.HIGH and the right edge in a SACK block in a sender-side time-sequence graph. The X-axis and Y-axis plot the time and the sequence number, respectively. Symbols "R" and "S" show the retransmitted segment and the SACK block in duplicate ACK, respectively. In Fig. 3, 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.

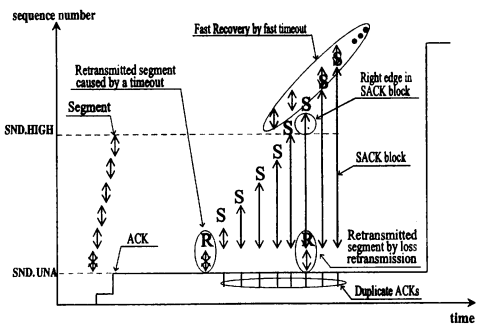



Figure 3: 2nd RTO segment retransmission using SACK option

# 4 Performance evaluation

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

## 4.1 Simulation model

In this paper, we implement the fast timeout and proposed loss retransmission algorithms in the *ns*2 simulator (ns-2.26) developed in the VINT project [12], and evaluate them using a communication model based on Fig. 4. The TCP sender in the wired network communicates with the receiver in the wireless network over a WCDMA protocol.

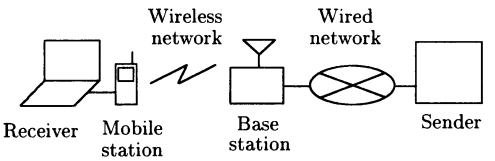



Figure 4: System model

Figure 5 shows the topology used in our experiments. The sender is connected to the BS via a 10 Mbps wired link with 20 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.

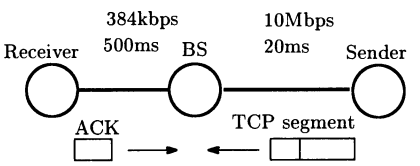



Figure 5: Simulation model

## 4.2 Proposed loss retransmission algorithm evaluation

Figure 6 shows the conventional TCP's time-sequence graph and sender state valuables for the

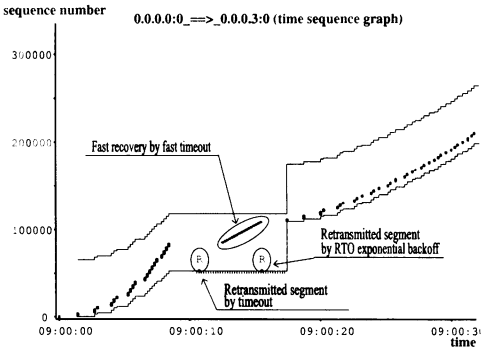
fast timeout algorithm. In Fig. 6 (a), the sender faced the retransmitted segment that caused by the RTO at 9:00:10.7. The conventional TCP has no function that can handle the arrival of the duplicate ACKs after a timeout without the next segment transmission using the fast timeout algorithm until the 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

$$\begin{aligned} \texttt{ssthresh} &\leftarrow \max(\texttt{cwnd}/2, 2) \\ &= 2, \end{aligned}$$

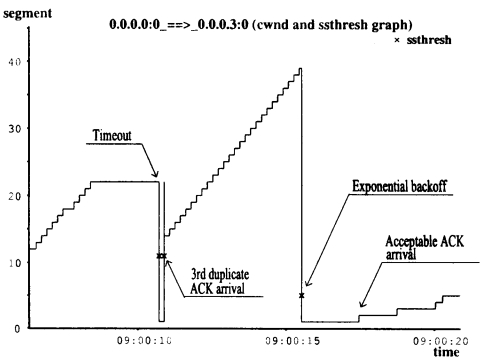
so the sender enters the congestion avoidance phase directly when the sender receives the acceptable ACK at 9:00:16.6. Figure 6 (b) shows the sender state variables, `cwnd` and `ssthresh`. The `ssthresh` is set to 2 at 9:00:15.4, the sender transmits a limited number of segments after that. As a result, the sender also degrades the throughput.

In contrast, Fig. 7 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. 7 (b). This allows it to achieve better throughput than conventional TCP.

Figure 8 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.




(a) Time-sequence graph




(b) Sender state variables, `cwnd` and `ssthresh`

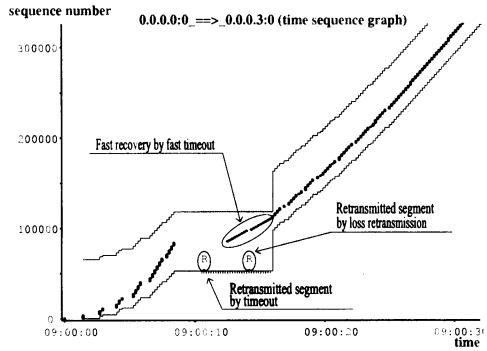
Figure 6: The TCP's time-sequence and sender state variables for the fast timeout algorithm

# 5 Conclusions

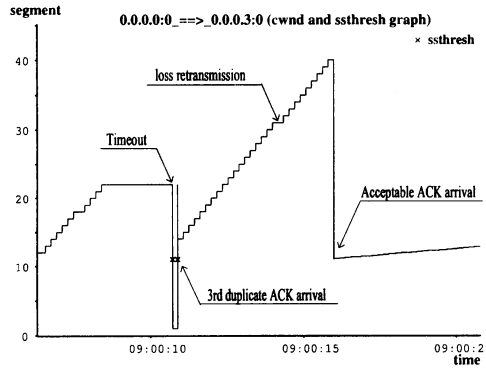
This paper proposed the loss retransmission algorithm for duplicate ACK arrival following a timeout. The proposed algorithm allows careful retransmission against failure to identify the first unacknowledged segment loss. Simulation results yield the following conclusion:

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

2. 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.




(a) Time-sequence graph




(b) Sender state variables, `cwnd` and `ssthresh`

Figure 7: The TCP's time-sequence and sender state variables for the proposed algorithms

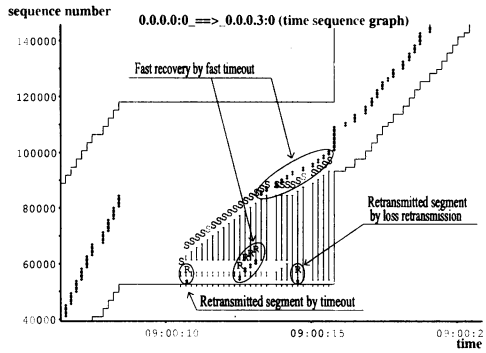



Figure 8: The proposed TCP's time-sequence graph (SACK)

## Acknowledgment

The authors would like to thank Reiner Ludwig and Daikichi Osuga for several useful discussions and comments.

## References

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

[2] 3GPP, "3G TS 25.322 v.3.5.0," RLC Protocol Specification, 2000.

[3] R. Ludwig and M. Meyer, "The Eifel Detection Algorithm for TCP," RFC3522, April 2003.

[4] P. Sarolahti and M. Kojo, "F-RTO: A TCP RTO Recovery Algorithm for Avoiding Unnecessary Retransmissions," draft-sarolahti-tsvwg-tcp-frto-04.txt, June 2003.

[5] M. Miyake, H. Inamura and O. Takahashi, "TCP Enhancement using Spurious Timeout Detection and Congestion Window Control Algorithm," *8th International Workshop on MoMuC 2003*, October 2003.

[6] R. Ludwig and R. H. Katz, "The Eifel Algorithm: Marking TCP Robust Against Spurious Retransmission," *SIGCOMM Computer Communication Review*, vol. 30, no. 1, January 2000.

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

[8] E. Blanton and M. Allman, "Using TCP DSACKs and SCTP Duplicate TSNs to Detect Spurious Retransmissions," draft-blanton-dsack-use-02.txt, October 2002.

[9] R. Ludwig, "Responding to Fast Timeouts in TCP," draft-ludwig-tsvwg-tcp-fast-timeouts-00.txt, July 2002.

[10] D. Lin and H. T. Kung, "TCP Fast Recovery Strategies: Analysis and Improvements," in *In Proceedings of IEEE INFOCOM 98*, March 1998.

[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, December 1999.

[12] K. Fall and K. Varadhan, "ns Note and documentation," *The VINT Project (UC Berkeley, LBL, USC/ISI, and Xerox PARC)*, December 2003.