Lost Retransmission Detection for TCP Part 2: TCP Using SACK Option*

Beomjoon Kim[1], Yong-Hoon Choi[2], Jaiyong Lee[3], Min-Seok Oh[1], and Jin-Sung Choi[1]

[1] Standardiztion & System Research Group (SSRG), Mobile Communication Technology Research Lab., CTO, LG Electronics Inc., LG R&D Complex, 533, Hogyel-Dong, Dongan-Gu, Anyang-City, Kyongki-Do, 431-749, Korea, {beom,minoh,jinsungc}@lge.com

[2] RAN S/W Group, System S/W Dept., System Research Lab., Telecommunication Equipment & Handset Company, LG Electronics Inc., LG R&D Complex, 533, Hogyel-Dong, Dongan-Gu, Anyang-City, Kyongki-Do, 431-749, Korea, dearyonghoon@lge.com

[3] Department of Electrical & Electronic Engineering, Yonsei University, 134 Shinchon-Dong Seodaemun-Gu, Seoul, 120-749, Korea, jyl@nasla.yonsei.ac.kr

Abstract. The performance of transmission control protocol (TCP) is largely dependent upon its loss recovery. Therefore, whether packet losses may be recovered without a retransmission timeout (RTO) or not is a very important issue. Although TCP using selective acknowledgement (SACK) option can recover multiple packet losses in a window, it cannot avoid RTO if a retransmitted packet is lost again. In order to cope with this problem, we propose a simple change to TCP SACK, which is called TCP SACK+ in simple. We use a stochastic model to evaluate the performance of TCP SACK+, and analyze its performance comparatively in terms of loss recovery probability. Numerical results evaluated by simulations show that TCP SACK+ can improve the loss recovery performance of TCP SACK significantly in presence of random losses.

1 Introduction

Since the specification of Transmission Control Protocol (TCP) is released [1], implementations of TCP have been enhanced with several mechanisms such as congestion control [4], [5]. TCP congestion control provides a function to detect and recover packet losses by retransmissions, which is called loss recovery in short. If the loss recovery is successful, packet transmission continues without a retransmission timeout (RTO). The loss recovery function of TCP operates using two basic algorithms of fast retransmit and fast recovery [2]–[5]. Because overall TCP performance has close relation to loss recovery efficiency, it has been a common focus to decrease the number of RTOs invoked if it is failed to recover packet losses by retransmissions [6]. Unnecessary RTOs can be divided into

* This work was supported by grant No.R01-2002-000-00531-0 from the interdisciplinary research program of the KOSEF.

N. Mitrou et al. (Eds.): NETWORKING 2004, LNCS 3042, pp. 88–99, 2004.

three classes. First, if multiple packets are lost in a window at the same time, then all of these packet losses cannot be recovered without RTO frequently. Recently, selective acknowledgement (SACK) option[1] [7], [8] is proposed to avoid performance degradation for this reason. Using SACK option, the sender can be informed about all the packets that have arrived successfully, so the sender need retransmit only the packets that have been lost. Second, if a packet is lost in a small window, then three duplicate ACKs may not be received to trigger a fast retransmit. Using Limited Transmit (LT) [10], most of RTOs corresponding to this class can be avoided. The third class of RTOs is caused by retransmission losses. All of existing TCP implementations including TCP SACK cannot avoid RTO when a retransmission is lost again [6], [11]. In order to avoid RTOs belonging to this class, we propose a simple algorithm that makes it possible for a TCP SACK sender to detect a lost retransmission and recover it by a retransmission. The proposed algorithm requires simple changes only to TCP implementation at the sender and is perfectly consistent with TCP specification such as additive increase multiplicative decrease (AIMD) principle [5]. According to [11], only 4% of the timeouts are due to lost retransmissions. However, it contributes to make TCP SACK more robust and perfect in that the final cause of unnecessary RTOs, which is remaining unsolved, can be avoided with no large modifications.

In order to evaluate the performance of the proposed algorithm, we model the sender's behavior during loss recovery. From the results of the modeling, the exact conditions for successful loss recovery can be derived. For derivation of loss recovery probability and stationary distribution of TCP window in steady-state, we mainly adopt the analysis using Markov process in [12]. Consequently, the improvement of the proposed algorithm can be comparatively analyzed with existing TCP versions in terms of the loss recovery probability that is normalized to the stationary distribution.

The rest of this paper is organized as follows. Section 2 provides a detailed presentation of the proposed algorithm with simulation results. In Section 3 the detailed analysis of loss recovery behaviors of TCP SACK is presented from the aspect of the loss recovery probability. Section 4 contains the numerical results and their discussion. Finally, some conclusions are summarized in Section 5.

2 Description of the Proposed Algorithm

In this paper, our focus is limited to TCP implementation using SACK option.[2] For TCP SACK, the proposed algorithm works for lost retransmission detection

[1] In the rest of this paper, we denote TCP implementation using SACK option by TCP SACK in simple. We consider "Sack1" presented in [6] as TCP SACK. Recently, the detailed loss recovery behaviors of Sack1 are addressed in terms of maintaining AIMD principle in [9].

[2] We have already proposed an algorithm called duplicate acknowledgement counting (DAC) which can be applied to TCP Reno and NewReno for lost retransmission detection. It cannot be included in this paper due to page limitation, and will appear in another publication as a Part 1 of this paper. Unlike DAC, because SACK information provides the information of well-transmitted packets, there is no need to count the number of duplicate ACKs.

on the basis of the packets transmitted after a retransmission. In the rest of this paper, TCP SACK using the proposed algorithm is indicated by a plus sign such as TCP SACK+.

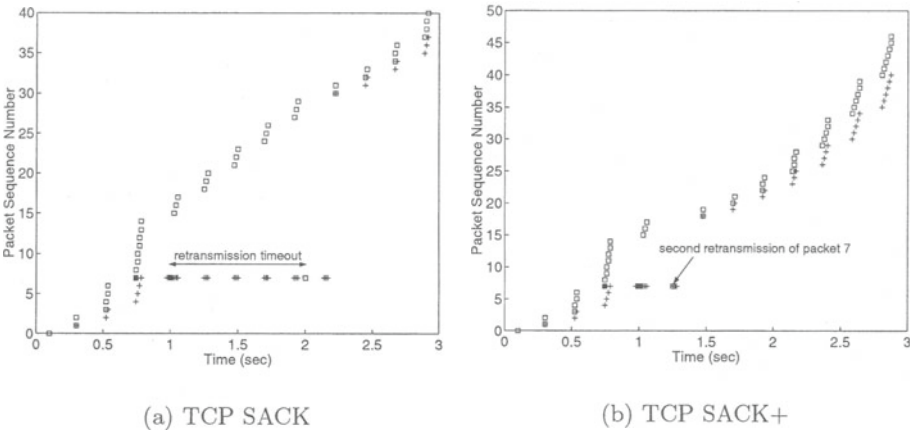
2.1 Description of TCP SACK+

For TCP SACK+ operations, the sender keeps a variable per packet loss when its retransmission is performed. In the variable, the highest sequence number of the packets that are outstanding is stored when a lost packet is retransmitted. We denote the highest sequence number at the time of the retransmission of the hth packet loss by S_h. If the retransmission of the hth packet loss is successful, the right edge of the first block within SACK information is always smaller than or equal to S_h. During fast recovery, every time a duplicate ACK is received, the sender checks the right edge of the first block. If the sender receives any duplicate ACK for the hth lost packet indicating that the right edge of the first block is greater than S_h, it determines that the retransmission is lost again and retransmits it immediately. Consequently, if at least a new packet is transmitted successfully after a retransmission, then the sender can detect whether its retransmission is lost or not. From the aspect of *conservativeness*, TCP SACK+ is perfectly consistent with the AIMD principle specified in [5]. As described above, a lost retransmission is detected based on the packets transmitted after the retransmission. It means that if congestion is so heavy that no packets should be transmitted, TCP SACK+ does not transmit additive packets after detecting a lost retransmission because the packets after the retransmission would be likely to be lost as well in such a congested situation. However, a lost retransmission detected assures that there was a quite heavy congestion so that it should be taken as two indications of congestion, which leads to decreasing *cwnd* twice as specified in [5].

2.2 Simulations

Using *ns* simulations, we implement TCP SACK+ and show the loss recovery behaviors of a TCP SACK+ sender in fig. 1 and fig. 2. In these figures, the well-transmitted packet is indicated by a blank square, lost packet by a black-filled square, and an ACK packet by +. In the simulations, several specific packets are forced to be dropped as in [6].

In fig. 1, we compare the loss recovery behaviors of TCP SACK+ with TCP SACK when a single packet and its retransmission are lost. At about 0.7 second, packets 7–14 are transmitted with *cwnd* of 8 and packet 7 is dropped. When the sender receives three duplicate ACKs at about 0.9 second, the sender retransmits packet 7 and sets *cwnd* to $4(= \lfloor 8/2 \rfloor)$ and `pipe` to $5(= 8 - 3)$. The last duplicate ACK decrease `pipe` to one so that three new packets, packet 15, 16, and 17, are transmitted after the retransmission. Because the retransmission of packet 7 is lost again, as can be seen in fig. 1-(a), the sender receives the same repetitive duplicate ACKs per every round-trip time (RTT) until RTO occurs. At about 2.3 second, it can be seen that the sender restarts to transmit in slow-start mode.

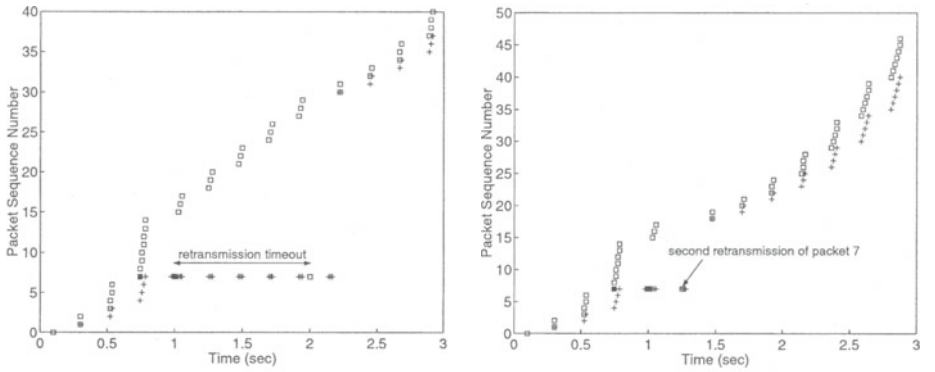


(a) TCP SACK (b) TCP SACK+

Fig. 1. Comparison between TCP SACK and SACK+ for a single packet loss and lost retransmission. (+ indicates an ACK packet.)

For TCP SACK+ shown in fig. 1-(b), the sender sets S_1 to 14 when it retransmits packet 7. The eighth duplicate ACK includes SACK information indicating that the first block starts with packet 8 and ends with packet 15. At this time, because the right edge of the block is greater than the stored value in S_2, the sender transmits the second retransmission of packet 7 at about 1.3 second. After the retransmission, the sender halves *cwnd* again to be $2(= \lfloor 4/2 \rfloor)$. Therefore, even if two more duplicate ACKs by packet 16 and 17 decreases `pipe` to be 2, no packets cannot be transmitted. At about 1.5 second, the second retransmission delivers an ACK that acknowledges packets up to packet 17, which brings the sender out of fast recovery and congestion avoidance starts with *cwnd* of 2.

We perform the same simulation for two packet losses and the results are shown in fig. 2. In this simulation, two packets 7 and 12 are lost and the retransmission of packet 12 is lost again. After fast retransmit of packet 7, the fifth duplicate ACK decreases `pipe` to be 3 so that one packet transmission is allowed. At this time, the sender can be informed by SACK information in the duplicate ACK that packet 12 is also lost. The final duplicate ACK decreases `pipe` again and packet 15 is transmitted. After a RTT, a partial ACK and duplicate ACK for packet 12 are received, which decrease `pipe` by three, and three new packets, packet 16, 17, and 18 are transmitted. However, the retransmission of packet 12 is lost again, the sender cannot complete fast recovery but RTO occurs eventually at about 2.3 second. For TCP SACK+ shown in fig. 2-(b), the sender sets $D_1 = D_2 = 14$ when it retransmits packet 7 and 12. A partial ACK after a RTT means that the retransmission of packet 7 is well-transmitted so that S_1 is cleared and two new packets, packet 16 and 17, are transmitted. After that, the seventh duplicate ACK by packet 15 is received indicating that the first block starts with packet 13 and ends with packet 15, which is greater than S_2. Therefore, the sender transmits the second retransmission of packet 12



(a) TCP SACK (b) TCP SACK+

Fig. 2. Comparison between TCP SACK and SACK+ for two packet losses and a single lost retransmission. (+ indicates an ACK packet.)

instead of packet 18 at about 1.3 second. An ACK that acknowledges packets up to packet 17 brings the sender out of fast recovery and congestion avoidance starts with *cwnd* of 2.

3 Modeling and Probabilistic Analysis

We adopt the concept of 'loss window' and 'round' defined in [12] and [13], respectively. If we denote a loss window by Ω and the ith packet loss in Ω by l_i, the first packet that Ω includes is always l_1. Additionally, we define Φ_k as the number of new packets that are transmitted in the kth round in loss recovery period. For n packet losses in Ω of u packets, Φ_0 is always equal to $u-n$. For modeling the evolution and obtaining stationary distribution of TCP congestion window, we mainly follow the procedures presented in [12] under the same assumptions such as fixed packet size, random packet losses, no ACK loss, and infinite packet transmission. We also follow some notations in [12] such as W_{max} for receiver's advertised window and K for *slow-start-threshold.*

3.1 TCP Reno

The recovery probability of TCP Reno is derived in terms of the number of packet losses in a loss window in [12]. In [12], it is assumed that the fast recovery of TCP Reno succeeds only if at most two packets are lost in a loss window. For $\Omega = u$ and a single packet loss, l_1 can be recovered without RTO if $\Phi_0 \geq K$ and the retransmission is not lost. Therefore, for $u \geq K+1$, its probability is given by

$$R_R^{(1)}(u) = (1-p)^{\Phi_0}(1-p) = (1-p)^u. \quad (1)$$

In the same way, l_2 can be recovered if $\Phi_1 \geq K$ and there is no packet loss during fast recovery. Since Φ_1 is equal to the number of new packets transmitted after the fast retransmission of l_1, $\Phi_1 = \lfloor u/2 \rfloor + \Phi_0 - u = \lfloor u/2 \rfloor - 2$.[3] For $\Omega = u$, when it contents $\lfloor u/2 \rfloor - 2 \geq K$, the recovery probability of l_2 is given by

$$R_R^{(2)}(u) = \binom{u-1}{1} p(1-p)^{u-2}(1-p)^{\Phi_1}(1-p)^2 = (u-1)p(1-p)^{u+\Phi_2}. \quad (2)$$

Using LT, a packet loss can be fast retransmitted if only a single duplicate ACK can be received. Therefore, even if Ω does not content $u \geq K+1$, a single packet loss may be recovered by a retransmission. For $\Omega = u$ and $2 \leq u \leq 3$, its probability, $R_L(u)$, is given by

$$R_L(u) = (1-p)^{u-1}(1-p)^2(1-p) = (1-p)^{u+2}. \quad (3)$$

It means that exactly one packet is lost out of u packets and three packets including two new packets transmitted by LT and the retransmission itself should not be lost. Note that LT works for only a single packet loss or the first packet loss in case of multiple packet losses per window [14]. Therefore, the recovery probability of TCP Reno using LT is given by $R_{RL}(u) = R_R(u) + R_L(u)$.

3.2 TCP SACK

For loss recovery of TCP SACK, packet transmission may be stalled in the second round due to misbehavior of `pipe`. Suppose that n packet losses are included in a loss window of u packets. After the sender receives all duplicate ACKs for l_1, it sets `pipe` to $n(= u - \Phi_0)$. If n is equal to or greater than $cwnd(= \lfloor u/2 \rfloor)$, no new packet can be transmitted in the first round. When the sender receives a partial ACK by the retransmission of l_1, it decreases `pipe` by two so that it is equal to $n-2$. Again, if $n - 2 \geq \lfloor u/2 \rfloor$, then no more packet can be transmitted but a RTO occurs in the end. Therefore, for successful loss recovery of TCP SACK, the number of packet losses is restricted by $n \leq \lfloor u/2 \rfloor + 1$. Consequently, the loss recovery probability of TCP SACK is given by

$$R_S(u) = \sum_{n=1}^{\mu_1} \binom{u-1}{n-1} p^{n-1}(1-p)^{u-n}(1-p)^n \quad (4)$$

where $\mu_1 = min(u-K, \lfloor u/2 \rfloor + 1)$. When LT is used for TCP SACK, its recovery probability, $R_{SL}(u)$, can be simply derived by replacement $(u-K)$ in μ_1 with $(u-1)$.

[3] According to our previous work presented in [14], three packet losses may be recovered without a RTO under the strict condition that a loss window size u is large enough to content $\lfloor u/4 \rfloor - 3 \geq K$ and there are at least $u - \lfloor u/4 \rfloor + (K-1)$ packets between l_1 and l_2. However, its probability is so small that we do not include the case in our derivation of the loss recovery probability of TCP Reno.

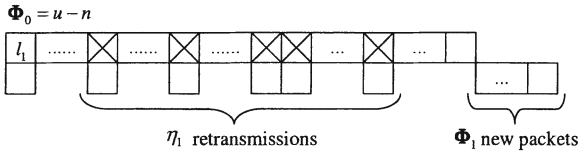


Fig. 3. Loss recovery behaviors of TCP SACK+ when the retransmission of n packet losses is completed in the first round.

3.3 TCP SACK+

We limit the number of lost retransmissions that the proposed algorithm can recover in a loss window to one and the same retransmission loss cannot be recovered twice. Therefore, the recovery probability of TCP SACK+ is given by

$$R_{S+}(u) = R_S(u) + p \cdot \Delta_S(u) \tag{5}$$

where $\Delta_S(u)$ is the probability that a lost retransmission is recovered. The term $p \cdot \Delta_S(u)$, therefore, means the probability that a retransmission is lost again and it is recovered by the proposed algorithm. When SACK option is used, the duration of loss recovery period is dependent on the position as well as the number of packet losses. For simplicity in the derivations of $\Delta_S(u)$, we limit our consideration to the cases that its loss recovery period is completed within a round or two.

First of all, as shown in fig. 3, we consider the case that loss recovery is finished at the first round. In this case, all of packet losses are retransmitted before a partial ACK is received by the retransmission of l_1. For n packet losses in Ω, if we denote the number of retransmissions sent by the decrement of `pipe` in the kth round by η_k, then we have

$$n = \eta_1 + 1. \tag{6}$$

When all of duplicate ACKs for l_1 are received, the values of *cwnd* and `pipe` are equal to $\lfloor u/2 \rfloor$ and n, respectively. Therefore, for $\Omega = u$, total number of packets that can be transmitted during the first round is given by

$$\eta_1 + \Phi_1 = \lfloor u/2 \rfloor - n. \tag{7}$$

For simplicity, as long as `pipe` permits, a retransmission is assumed to be always transmitted first regardless of its position in a loss window. Then, a retransmission loss out of n retransmissions can be detected if $\Phi_1 \geq 1$. From (6) and (7), the condition is given by

$$1 \leq n \leq \lfloor u/4 \rfloor. \tag{8}$$

If the last packet in a loss window is lost, even if the above condition is satisfied, the retransmission of n packet losses cannot be completed in the first round. Hence, we do not include this case in the derivation of $\Delta_S(u)$. As a consequence, the recovery probability of a lost retransmission in the first round, $\Delta_S^{(1)}(u)$, is

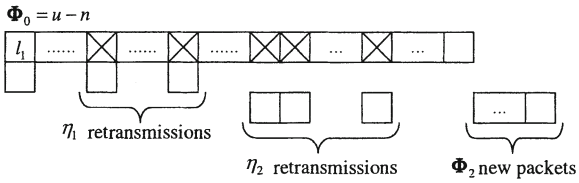


Fig. 4. Loss recovery behavior of TCP SACK+ when the retransmission of n packet losses is completed in the second round.

given by

$$p \cdot \Delta_S^{(1)}(u) = p \cdot \sum_{n=1}^{\lfloor u/4 \rfloor} \binom{u-2}{n-1} \binom{n}{1} p^{n-1}(1-p)^{u+\Phi_1} \tag{9}$$

where it reflects the following two facts:
i) $(n-1)$ packets are lost out of $(u-2)$ packets which do not include the last packet in Ω.
ii) A retransmission is lost out of n retransmissions, and Φ_1 packets and a retransmission by SACK+ should not be lost.

If $n \geq \lfloor u/4 \rfloor + 1$, the recovery period continues after the first round as shown in fig. 4. If it is assumed that the retransmission of n packet losses is to be completed in the second round, then we have

$$n = 1 + \eta_1 + \eta_2. \tag{10}$$

Recalling the assumption that a retransmission is always transmitted first, it can be inferred that all of packets transmitted during the first round are retransmissions; i.e., $\Phi_1 = 0$. Therefore, the sender is going to receive $(\eta_1 + 1)$ partial ACKs in the second round, so that total number of packets that the sender is allowed to transmit in the second round is given by

$$\eta_2 + \Phi_2 = 2(\eta_1 + 1) = 2(\lfloor u/2 \rfloor - n + 1). \tag{11}$$

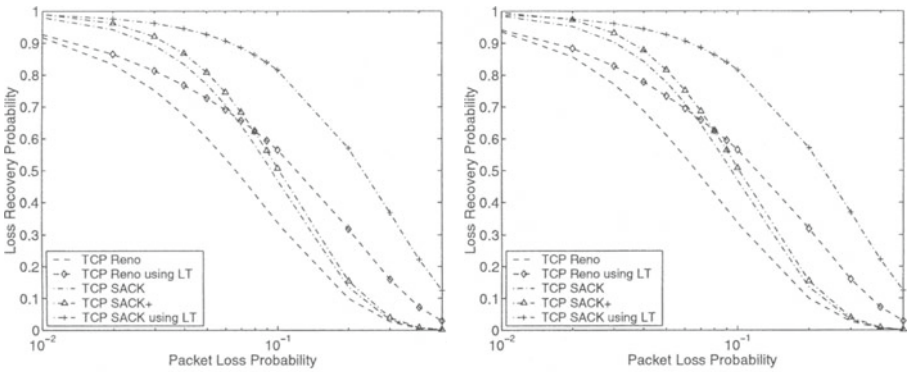
Note that a partial ACK decrements `pipe` by two [6]. From (10) and (11), the condition for $\Phi_2 \geq 1$ is given by

$$n \leq 0.75\lfloor u/2 \rfloor + 0.5. \tag{12}$$

As a consequence, for $\lfloor u/4 \rfloor + 1 \leq n \leq 0.75\lfloor u/2 \rfloor + 0.5$, the recovery probability of a retransmission loss in the second round is given by

$$p \cdot \Delta_S^{(2)}(u) = p \cdot \sum_{n} \binom{u-2}{n-1} \binom{\eta_2}{1} p^{n-1}(1-p)^{u+\Phi_2} \tag{13}$$

where it reflects the following three facts:



(a) $W_{max} = 8$ (b) $W_{max} = 32$

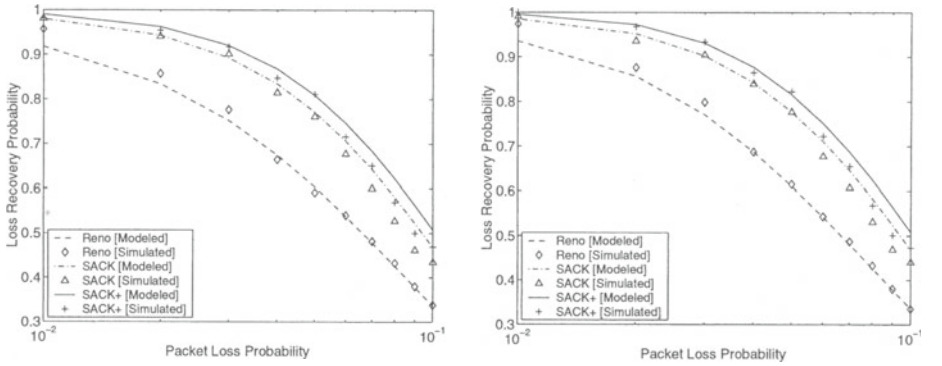
Fig. 5. Comparison of the loss recovery probability predicted by the developed model.

i) $(n-1)$ packets are lost out of $(u-2)$ packets which do not include the last packet in Ω.
ii) $(\eta_1 + 1)$ retransmissions should not be lost.
iii) A retransmission is lost out of η_2 retransmissions, and Φ_2 packets and a retransmission by SACK+ should not be lost.
From (5), (9), and (13), the total loss recovery probability of TCP SACK+ can be derived.

4 Results and Discussion

We calculate the loss recovery probability[4] that is normalized to the stationary distribution of the window, which is obtained from Markov process of the window evolution for packet loss probability as presented in [12]. The x-axis of each graph indicates packet loss probability, which corresponds to p. We assume that K is always three. In fig. 5, the loss recovery probability of each TCP is compared for two different values of W_{max}. When LT is not used, the loss recovery probability of all TCP starts to drop rapidly when packet loss probability exceeds 10^{-2}. For TCP SACK, the recovery of multiple packet losses would likely be successful if only the first lost packet can be recovered by fast retransmit. Therefore, the drop of the loss recovery probability for packet loss probabilities exceeding 10^{-2} can be explained by the fact that fast retransmit of the first lost packet cannot be triggered well due to lack of duplicate ACKs by the small congestion window. As can be seen in this figure, when LT is adopted, the loss recovery probability

[4] Since it has been already proved in [12] that TCP performance such as throughput or goodput is directly proportional to the loss recovery probability, we do not include the results of throughput comparison. Also, it can be inferred intuitively by simulation results presented in Section 2.



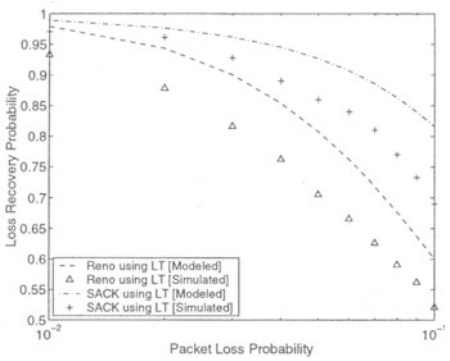
(a) TCP Reno, SACK, and SACK+ ($W_{max} = 8$)

(b) TCP Reno, SACK, and SACK+ ($W_{max} = 32$)

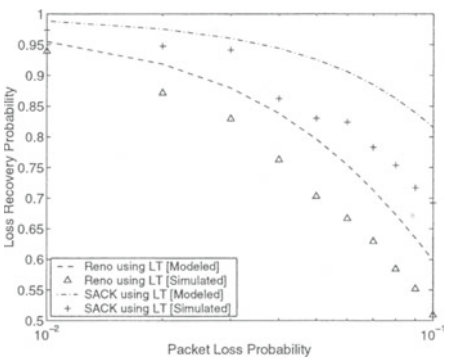
Fig. 6. Comparison of the loss recovery probability predicted by the developed model to the simulated results when the proposed algorithm is applied.

shows a slow decline with the increment of packet loss probability. For packet loss probability over 10^{-1}, the recovery probability of Reno using LT is rather higher than that of TCP SACK and SACK+. It is because, for such a large packet loss probability, the sender cannot keep its size large enough due to the frequent loss recovery events so that the capability of recovery for multiple packet losses makes little difference to the recovery probability. Since LT increases the likelihood of the first fast retransmit, when it is used with SACK option, the sender has more chances to recover multiple packet losses by retransmissions. It is the reason for the higher improvement when LT is used for TCP SACK than Reno.

There is a considerable improvement in the loss recovery probability of TCP SACK compared with TCP Reno. The difference between two lines of SACK and Reno reflects the capability to handle multiple packet losses without RTO or not. The slight difference between TCP SACK and SACK+ reveals the problem of lost retransmissions. As packet loss probability increases, more packet losses may occur and their retransmissions also tend to be lost again. On the other hand, as mentioned earlier, the congestion window is not large enough that multiple packet losses are not likely to be included in a window. That is, the probability for an event that a retransmission is lost again is quite low, which is the reason that the overall values of difference in loss recovery probability between SACK and SACK+ are not quite significant. According to the results measured in the real Internet presented in [11], the largest portion, about 85%, of timeouts are caused by small window, about 11% by multiple packet losses, and 4% by retransmission losses. The results obtained from our model verifies it again in that the largest improvement is made by LT, the second largest by SACK option, and the last by SACK+.



(a) TCP Reno and SACK using LT ($W_{max} = 8$)



(b) TCP Reno and SACK using LT ($W_{max} = 32$)

Fig. 7. Comparison of the loss recovery probability predicted by the developed model to the simulated results when LT is applied.

As shown in fig. 5, further increment of W_{max} to 32 makes no significant differences to the loss recovery probability. For low packet loss probability, it is unlikely that there are more than two packet losses in a window. Note that a single packet loss can be retransmitted if only three duplicate ACKs are received, namely, the window is equal to or greater than four. Even if multiple packet losses occur, they can be retransmitted if the first fast retransmit is successful. Consequently, a large W_{max} may benefit the throughput of TCP but not its loss recovery performance.

In fig. 6, we evaluate the loss recovery probability of TCP SACK and SACK+ using *ns* simulations. A sender and a receiver are connected with a long-fat link of 10Mbps and 100msec where packets are dropped in random. Using FTP, the sender transmits data consist of 10^4 packets whose size is 1kbytes, and the congestion window can grow up to W_{max}. The loss recovery probability is defined as the ratio of the number of the packets recovered by retransmissions to the total number of packet losses. It can be seen that the simulated values and the calculated values of our developed model fit well.

In fig. 7, we also evaluate the loss recovery probability when LT is used. There is a remarkable difference between the modeled results and the simulated results. It is because only the number of duplicate ACKs are considered as the condition for successful loss recovery in our modeling process. However, even if three duplicate ACKs are to be received, a RTO occurs if it expires before the third duplicate ACK arrives. If we suppose the worst case that the second packet is lost in a window of two in congestion avoidance, it takes about four RTTs to receive the third duplicate ACK. Therefore, the practical improvement of LT may be much smaller than the modeled results.

5 Conclusions

In this paper, we have proposed a simple algorithm that enables a TCP sender using SACK option to detect and recover retransmission losses without a RTO. The analysis in this paper assumes an environment where packets are dropped in random with probability p and the losses are independent to each other. Therefore, although the proposed algorithm can recover almost all of retransmission losses, its improvement is not significant since a retransmission loss itself is not such a common event in this loss model. However, according to queue management schemes such as droptail and with changing levels of congestion, the likelihood of a retransmission loss may have some variation. If it may occur more frequently than in our model, the change proposed in this paper is of a great benefit to loss recovery performance of TCP.

References

1. J. Postel: Transmission Control Protocol. RFC 793, (1981)
2. V. Jacobson: Congestion Control and Avoidance. ACM SIGCOMM'88, (1988)
3. V. Jacobson: Modified TCP Congestion Avoidance Algorithm. note sent to end2end-interest mailing list, (1990)
4. W. Stevens: TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms. RFC 2001, (1997)
5. M. Allman, V. Paxson, and W. Stevens: TCP Congestion Control. RFC 2581, (1999)
6. K. Fall and S. Floyd: Simulation-based Comparisons of Tahoe, Reno, and SACK TCP. ACM Computer Communication Review, vol. 26. (1996) 5–21
7. M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow: TCP Selective Acknowledgement Options. RFC 2018, (1996)
8. S. Floyd, J. Madavi, M. Mathis, and M. Podolsky: An Extention to the Selective Acknowledgement (SACK) Option for TCP. RFC 2883, (2000)
9. E. Blanton, M. Allman, K. Fall, and L. Wang: A Conservative Selective Acknowledgement (SACK)-based Loss Recovery Algorithm for TCP. RFC 3517, (2003)
10. M. Allman, H. Balakrishan, and S. Floyd: Enhancing TCP's Loss Recovery Using Limited Transmit. RFC 3042, (2000)
11. Dong Lin and H. T. Kung: TCP Fast Recovery Strategies: Analysis and Improvements. IEEE INFOCOM'98, (1998) 263–271
12. Anurag Kumar: Comparative Performance Analysis of Versions of TCP in a Local Network with a Lossy Link. IEEE/ACM Transactions on Networking, vol. 6. (1998) 485–498
13. J. Padhye, V. Firoiu, D. F. Towsley, and J. F. Kurose: Modeling TCP Reno Performance: A Simple Model and Its Empirical Validation. IEEE/ACM Transactions on Networking, vol. 8. (2000) 133–145
14. Beomjoon Kim and Jaiyong Lee: Analytic Models of Loss Recovery of TCP Reno with Packet Losses. LNCS 2662, (2003) 938–947