On Bursty Packet Loss Model for TCP Performance Analysis

Kaiyu Zhou, Kwan L. Yeung and Victor O.K. Li
Department of Electrical and Electronic Engineering
The University of Hong Kong, Pokfulam, Hong Kong, China
kyzhou@eee.hku.hk, kyeung@eee.hku.hk, vli@eee.hku.hk

Abstract—In this paper, we study the timeout probability of TCP Reno under the bursty packet loss model, which is widely used to represent the loss characteristics of TCP under drop-tail FIFO queues. With a detailed analysis on the three timeout reasons for TCP Reno, we show that the impact of timeout has been underestimated in the existing literature. Surprisingly, we find that this more precise representation of timeout probability does not match the actual performance of TCP under drop-tail FIFO queues. Therefore we conclude that the bursty loss model is incapable of capturing the behavior of drop-tail FIFO queues, and using bursty loss model to analyze TCP performance is flawed.

I. INTRODUCTION

TCP dominates the Internet traffic [12]. Its performance affects the performance of the overall Internet. Analytical models for TCP give us a better understanding of the sensitivity of TCP performance to various network parameters. They also provide insights in the design of active queue management [7] and TCP-friendly rate control schemes [10]. Existing TCP analytical models focus on the steady-state throughput of long-lived TCP flows [1,2,6] or/and the transfer delay of short-lived TCP flows [2-6].

A fundamental issue in modeling TCP performance is to determine the packet loss model to be adopted. Of the existing TCP analytical models, the *bursty loss model* is widely used [1-5], and it is assumed to be an idealization of drop-tail FIFO queueing, i.e. when a FIFO queue is full, all subsequently arrived packets that belong to the same *burst* (to be formally defined later) will be dropped. An important steady-state throughput model based on bursty loss model was proposed in [1]. The throughput equations derived there have also been recommended for adoption by TCP-friendly rate control protocol [10].

In this paper, we determine if the bursty loss model is suitable for analysis of TCP performance over FIFO queue. Although it is inferred in [1] that for TCP Reno the probability of timeout after a bursty loss is high, it does not model fast recovery. In [2], it is found that the probability of timeout with TCP Sack is about 50 percent under bursty loss after modelling fast recovery, and emphasizes the importance of the proper determination of loss model for TCP performance study. After modelling fast recovery, with a detailed analysis on three possible timeout reasons, we show that bursty loss also causes a very high timeout probability in TCP Reno. Although such high timeout probability is consistent with the simulation results based on the exact loss model, it does not match the actual performance of TCP Reno under drop-tail FIFO queues. This shows that the currently prevailing bursty loss model cannot precisely capture the loss characteristics of TCP Reno under drop-tail FIFO queues, so adopting it for TCP performance analysis is flawed.

This research is supported in part by the Research Grants Council of Hong Kong under Grant No. 7044/02E.

The rest of this paper is organized as follows. Section II reviews the important mechanisms of TCP Reno. Section III presents the definition and assumptions adopted in our analysis. Section IV derives the timeout probability of TCP Reno under bursty loss model. Section V validates our analytical model and compares our analysis with simulation results. Section VI concludes this paper.

II. THE MECHANISM OF TCP RENO

TCP [8] is a window-based, connection-oriented, reliable transport layer protocol. Let the next packet to be acknowledged as W_{start}. The sequence numbers of all the outstanding packets is then denoted by $[W_{start}, W_{start} + cwnd - 1]$. With each ACK received, W_{start} increases by a value equal to the number of packets acknowledged. This allows more data packets to be sent. To keep track of the packets sent, the sender maintains a variable $maxseq$, which stores the highest sequence number the sender has sent out. Accordingly, a new packet can be sent out only if

$$W_{start} + cwnd - 1 > maxseq . \quad (1)$$

TCP Reno [8] adds *slow-start*, *congestion avoidance*, *fast retransmit* and *fast recovery* algorithms to the original TCP. During the slow-start phase, the sender increases its $cwnd$ by one with each ACK, until the slow-start threshold H is reached and the congestion avoidance phase takes over. In congestion avoidance, the sender increases its $cwnd$ linearly by $1/cwnd$ with each ACK received. If triple duplicate ACKs (TD) are received, the sender infers a packet loss and retransmits the lost packet, i.e. fast retransmit. The sender then sets H to $cwnd/2$, halves its $cwnd$ and activates the fast recovery algorithm. In fast recovery, the sender treats each duplicate ACK as a signal that one packet has left the network. To keep the pipe between the sender and the receiver full, the sender increases its usable congestion window by one for each duplicate ACK received. When the retransmitted packet is acknowledged, the sender exits fast recovery and sets $cwnd$ to H. Then the sender enters the congestion avoidance phase again. Note that TCP Reno can retransmit at most one lost packet per round-trip-time (RTT). Since the arrival of duplicate ACKs implies that the receiver did receive some packets that follow the loss, the value of $maxseq$ is not cleared and this prevents the unnecessary re-sending of data that has already been received.

Retransmission timeout [9] is used as the last resort to recover lost packets. Every time a data packet is sent, if the retransmission timer is not running, a new timer is started to count down with an initial value of t_0 seconds. t_0 is given by

$$t_0 = \min(60, \max(1, RTT + 4 \cdot RTTVAR)) . \quad (2)$$

where RTT is the round trip time and $RTTVAR$ is the variance of the round trip time. In the case that another timeout

occurs before receiving an ACK that acknowledges *maxseq* (when fast recovery starts), the sender *backs off* its retransmission timer by setting it to $2t_0$. From (2), the doubling of the retransmission timer is only effective when t_0 (before doubling) is smaller than 60 seconds.

III. DEFINITION AND ASSUMPTIONS

For ease of our later presentation, this section introduces our assumptions for the analysis in Section IV and our definition of *timeout probabiltiy*.

Our analysis adopts the same set of assumptions about the end systems and network as in [1-3] and [5]. We consider a saturated TCP Reno sender, i.e., a flow with an unlimited amount of data to send. We also assume that a TCP connection is generally able to enter congestion avoidance phase before the next loss event. This helps us to simplify the analysis by neglecting the loss which occurs during the slow start. We assume that for the duration of the data transfer, the sender always sends full-sized packets as fast as its congestion window allows, and the receiver advertises a consistent flow control window. The effects of Nagle algorithm and silly window syndrome avoidance [8] are not considered.

We model the performance of TCP Reno in terms of "*rounds*." A round begins with the transmission of a window of packets and ends on the receipt of one or more ACKs of these packets, which implies that the time needed to send out all the packets in a window is smaller than the duration of a round. Therefore the duration of a round is independent of the window size, and is determined mainly by the round trip propagation delay. Note that with TCP Reno congestion control, this means the congestion window size at the sender must always be smaller than the bandwidth-delay product of the path, so the flow is not fully utilizing the path bandwidth.

The *bursty loss model* [1] is adopted. Let the probability that a packet is lost in a round be p. p is independent of any packet loss in other rounds. In the same round, whether a packet will be lost is determined by whether the previous packet in the same round is lost. If the previous packet is lost, the current packet is lost. Otherwise the packet is lost with probability p. In other words, if a packet is lost in a round, all the packets follow it in the same round (i.e. a *burst* of packets) are lost. Like [1,5], we assume that the probability of packet loss is independent of window size and this is again only valid when flows are not fully utilizing the link. To simplify the discussion, ACK packet losses are ignored as [1-6]. We treat all the correlated packet losses in the same round as a single *loss event*.

From Section II, a TCP sender responds to packet losses with either fast retransmit (followed by fast recovery) or timeout. Given a packet *loss event*, we define the probability that the TCP sender has to recover its data transmission with timeout (i.e. fast retransmit fails to recover it) as the *timeout probability*, or Q.

IV. TIMEOUT PROBABILITY

A. How does timeout happen?

After a loss event, three conditions may trigger the retransmission timeout. We call them the three timeout reasons.

Reason 1: Not enough (i.e. 3) duplicate ACKs to trigger a triple duplicate ACKs (TD) indication after a loss event. This is what has been discussed in detail in [1].

Reason 2: After the lost packet is retransmitted, if the retransmission fails again, a timeout occurs. Note that although the sender may send out new packets during fast recovery, those packets will be acknowledged by the same duplicate ACKs. The retransmission timer is not cleared and a timeout eventually occurs.

Reason 3: Even after the retransmitted packet is successfully acknowledged, timeout *may* still happen. Upon receiving the ACK for the retransmitted packet, the sender exits fast recovery and enters congestion avoidance. The data transfer is now regulated by the conditon stipulated in (1). Compared with when packet loss is detected, the sender now has a decreased *cwnd* and an increased W_{start}. Note there is no guarantee that $W_{start} + cwnd - 1 > maxseq$, and the sender may send no new packets and a timeout occurs. (We will elaborate more on this reason in the next subsection.)

B. What happens after a loss event?

With the understanding of the three timeout reasons, we now focus on how packets are sent in the *rounds* following a loss

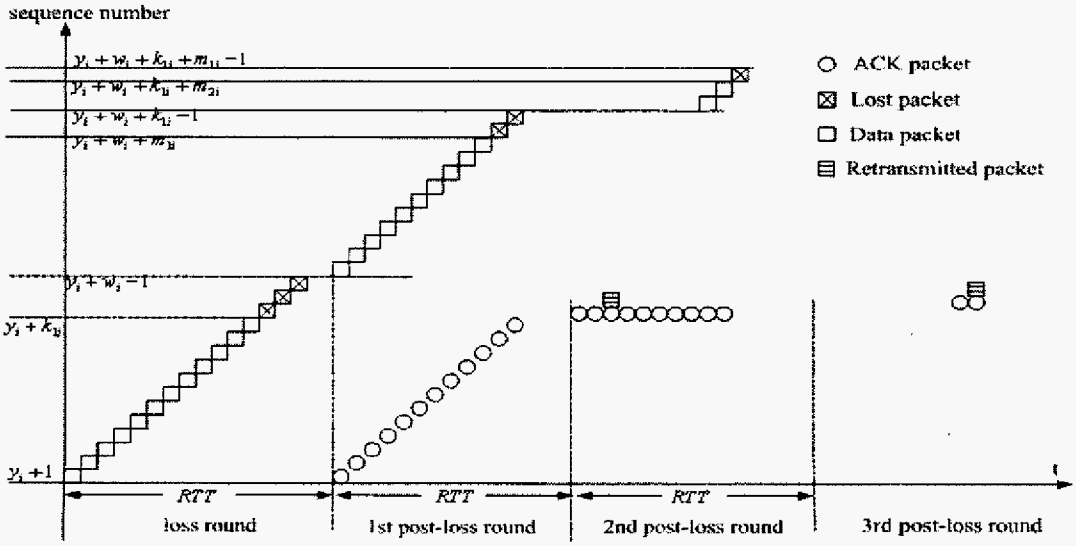

Fig. 1. The rounds following the loss event

event. Our aim is to show that there are seven possible reactions/events of a TCP Reno sender to a loss event. Then by conditioning on these seven events, the timeout probability is found.

Fig. 1 tracks the packets sent in the rounds after a loss event. We refer to the round where a loss event occurs as the "*loss round*" and all subsequent rounds as the "*n*-th *post-loss round*," where n is the round-distance from the loss round. Let y_i+1 be the sequence number of the first packet in the loss round, and w_i be the *cwnd*. Thus w_i packets are sent in the loss round. Let $y_i+k_{1i}+1$ be the sequence number of the first lost packet. With the bursty loss model, all packets following $y_i+k_{1i}+1$ and in the loss round are also lost. However, since packets y_i+1, y_i+2, ..., y_i+k_{1i} are acknowledged, another k_{1i} packets, y_i+w_i+1, y_i+w_i+2, ..., $y_i+w_i+k_{1i}$ can be sent in the 1st post-loss round. This round of packets may have another loss, say at packet $y_i+w_i+m_{1i}+1$, then packets following it in the 1st post-loss round are also lost. The packets successfully sent in the 1st post-loss round are acknowldged by duplicate ACKs for y_i+k_{1i} in the 2nd post-loss round. The number of duplicate ACKs is equal to the number of successfully received packets in the 1st post-loss round, with our assumption that no ACK packet loss happens. If the number of such ACKs is more than three, then a *Triple Duplicate* (TD) ACKs indication occurs. We denote this event by TD1. Packet $y_i+k_{1i}+1$ is then retransmitted in the 2nd post-loss round. If the number of duplicate ACKs is less than three, a *timeout* (TO) occurs. The associated event is denoted by TO1. Note that TO1 follows the timeout reason 1. Let $P_{NTD}(w_i)$ be the probability that TO1 occurs given that a loss event happened. From [1], $P_{NTD}(w_i)$ is given by

$$P_{NTD}(w_i)=\min(1,\frac{(1-(1-p)^3)(1+(1-p)^3(1-(1-p)^{w_i-3}))}{1-(1-p)^{w_i}})$$

In the 2nd post-loss round, after receiving the third duplicate ACK for y_i+k_{1i}, the sender resends packet $y_i+k_{1i}+1$, set its *cwnd* to $w_i/2$, and enters fast recovery. The sender expands its usable congestion window size with the number of duplicate ACKs received. Since m_{1i} packets are acknowledged in the 2nd post-loss round, the usable congestion window at the end of this round is $w_i/2+m_{1i}$. From (1), another k_{2i} packets may be sent out in this round, k_{2i} is given by

$$k_{2i}=m_{1i}-\frac{w_i}{2},\ \text{where } 3\le m_{1i}\le k_{1i}\le w_i.$$

Among the k_{2i} packets sent, another loss may happen. Let the first loss be $y_i+w_i+k_{1i}+m_{2i}+1$, so m_{2i} packets are acknowledged in the 3rd post-loss round. If the retransmitted packet $y_i+k_{1i}+1$ is lost, a timeout occurs (which follows the timeout reason 2). We denote this event by TO2.

Let $A(n,m)$ be the probability that the first m packets are acknowledged in a round of n packets sent, given there is a loss event in the round. Then

$$A(n,m)=\frac{(1-p)^m p}{1-(1-p)^n}.$$

Let $C(n,m)$ be the probability that n packets are sent out but only the first m packets are acknowledged, and the rest of the packets in the round are lost. Then

$$C(n,m)=\begin{cases}(1-p)^m p, & m\le n-1\\ (1-p)^n, & m=n\\ 0, & n<m\end{cases}.$$

Then $P(m_{2i}\ge i)$ be the probability that more than i packets are acknowledged in the 3rd post-loss round. $P(m_{2i}\ge i)$ is given by

$$P(m_{2i}\ge i)=\sum_{k_{1i}=3}^{w_i}A(w_i,k_{1i})\sum_{m_{1i}=3}^{k_{1i}}C(k_{1i},m_{1i})\sum_{m_{2i}=i}^{m_{1i}}C(m_{1i}-\frac{w_i}{2},m_{2i}).$$

If the retransmitted packet is successfully received, and $m_{2i}<3$, depending on the number of lost packets in the loss event, a timeout may occur (event TO3) following the timeout reason 3, otherwise the data transfer is recovered (event RE1). After receiving the ACK for retransmission, the sender's window size is determined by

$$\begin{aligned}&cwnd=w_i/2\\ &maxseq=y_i+w_i+k_{1i}+k_{2i}\\ &W_{start}=\begin{cases}y_i+k_{1i}+2, & \text{if } k_{1i}\le w_i-2\\ y_i+w_i+k_{2i}+1, & \text{if } k_{1i}=w_i-1 \text{ and } m_{1i}<k_{1i}\\ y_i+w_i+k_{1i}+k_{2i}+1, & \text{if } k_{1i}=w_i-1 \text{ and } m_{1i}=k_{1i}\end{cases}\end{aligned}\quad (3)$$

After substituting (3) into (1), we find that the data transfer can be recovered only if $k_{1i}=w_i-1$ and $m_{1i}=k_{1i}$ (no loss with the k_{1i} packets sent in the 2nd post-loss round). Thus the probability that event RE1 occurs given the retransmitted packet in TD1 has been successfully acknowledged is

$$P_1(w_i)=\frac{A(w_i,w_i-1)(1-p)^{w_i-1}}{(1-P_{NTD}(w_i))(1-p)P(m_{2i}<3)}.$$

When the retransmitted packet is successfully received and $m_{2i}\ge 3$, if there are more than two packets lost in the loss event, triple duplicate (TD) ACK will be detected for the second time in the 3rd post-loss round (denote this event by TD2). The sender resends the second lost packet, halves its *cwnd* to $w_i/4$, expands its usable congestion window by m_{2i} and continues in fast recovery phase. In such case, a single loss event triggers two consecutive fast retransmits. The maximum of the usable congestion window in the 3rd post-loss round becomes $w_i/4+m_{2i}$, which is smaller than the maximum of the usable congestion window in the 2nd post-loss round $w_i/2+m_{1i}$. At the same time W_{start} increases only by 1, so no more packets can be sent in the 3rd post-loss round. The retransmission in event TD2 may also be lost. Like the retransmision in event TD1, if the retransmitted packet is lost, a timeout that follows timeout reason 2 occurs (event TO4); if the retransmission is successfully ACKed, a timeout that follows timeout reason 3 *may* occur (event TO5); otherwise the data transfer is recovered (event RE2).

$$\begin{aligned}&W_{start}=\begin{cases}y_i+k_{1i}+3, & \text{if } k_{1i}\le w_i-3\\ y_i+w_i+m_{1i}+j, & \text{if } k_{1i}=w_i-3+j, m_{1i}\le k_{1i}-j\\ y_i+w_i+k_{1i}+m_{1i}+1, & \text{if } k_{1i}=w_i-3+j, m_{1i}>k_{1i}-j\end{cases}\\ &\qquad\qquad \text{where } j=1,2\\ &cwnd=w_i/4\\ &maxseq=y_i+w_i+m_{1i}+k_{1i}\end{aligned}\quad (4)$$

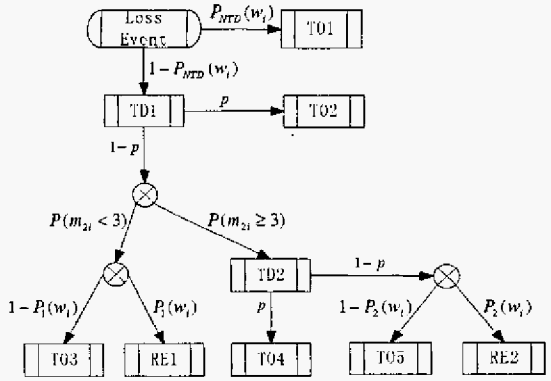

Fig. 2. Different reactions/events after a loss event

With the receiving of the ACK for the retransmission in event TD2, the sender's window size is given by (4). Given that the retransmitted packet of TD2 is successfully ACKed, the probability that event RE2 happens is

$$P_2(w_i)=\frac{\sum_{k=1}^{2}A(w_i,w_i-3+k)\sum_{j=0}^{k-1}C(w_i-3+k,w_i-3+k-j)}{(1-P_{NTD}(w_i))(1-p)^2P(m_{2i}\geq 3)}.$$

As the result, after a loss event, a TCP Reno sender may respond with one of the five timeout events (TO1 to TO5) or one of the two recovery events (RE1 and RE2), whereas events TD1 and TD2 serve as *intermediate* events. Therefore the probability that a loss event causes retransmission timeout is given by

$$Q=\sum_{j=1}^{6}E[P_{TOj}(w_i)],$$

where P_{TOj} denotes the probability that the timeout event TOj happens. Similarly, P_{REj} denotes the probability that the recovery event REj happens. Fig. 2 shows the state transition diagram of all the defined TD, TO and RE events. In Fig. 2 each block represents an event and each edge represents the transition probability. From Fig. 2 the probability for each TO or RE event is derived in (5).

$$\begin{cases}P_{TO1}(w_i)=P_{NTD}(w_i)\\ P_{TO2}(w_i)=(1-P_{NTD}(w_i))p\\ P_{TO3}(w_i)=(1-P_{NTD}(w_i))(1-p)P(m_{2i}<3)(1-P_1(w_i))\\ P_{TO4}(w_i)=(1-P_{NTD}(w_i))(1-p)P(m_{2i}\geq 3)p\\ P_{TO5}(w_i)=(1-P_{NTD}(w_i))(1-p)^2P(m_{2i}\geq 3)(1-P_2(w_i))\\ P_{RE1}(w_i)=(1-P_{NTD}(w_i))(1-p)P(m_{2i}<3)P_1(w_i)\\ P_{RE2}(w_i)=(1-P_{NTD}(w_i))(1-p)^2P(m_{2i}\geq 3)P_2(w_i)\end{cases} \quad (5)$$

So the timeout probability is approximately

$$Q\approx\sum_{i=1}^{6}P_{TOi}(w)=\sum_{i=1}^{6}P_{TOi}(E[w_i]). \quad (6)$$

From (5) and (6), the timeout probability Q is expressed as a function of w, the expection of w_i.

V. Comparison and Validation

In this section, the analytical results of Q from (5) and (6) are compared with the simulation results based on the bursty loss model and drop-tail FIFO queues respectively. All the simulations use TCP Reno agent of ns-2.1b9 [11] and all the packets in the simulations are of size 1KBytes.

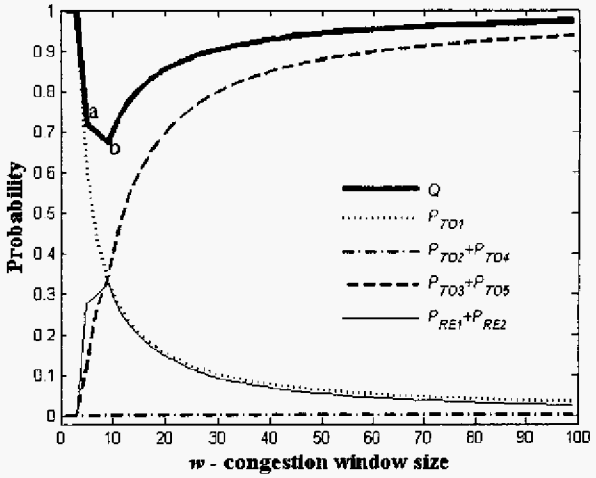

Fig. 3. Timeout probability of different reasons vs. w, p=0.001

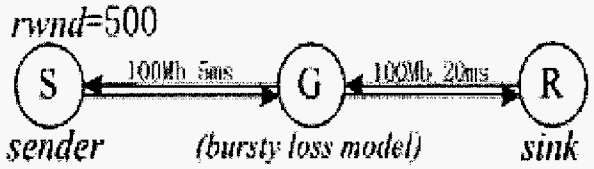

Fig. 4 The simulation network for bursty loss model

Fig. 3 plots Q, obtained from (5) and (6) with $p=0.001$, against congestion window size w. The probabilities that a timeout is triggered by reason 1 only (P_{TO1}), reason 2 only ($P_{TO2}+P_{TO4}$), reason 3 only ($P_{TO3}+P_{TO5}$), and the probability of recovery after a loss event ($P_{RE1}+P_{RE2}$) are shown. Although the probability of timeout triggered by reason 1 drops quickly with increasing w, the probability of timeout triggered by reason 3 increases. The overall timeout probability Q decreases quickly after w reaches 3, and slows down after point *a*, and starts to increase after point *b*. For all the w values Q is always high, with a minimum of about 0.68 as shown in Fig. 3. (Further analysis shows that increasing p results in decreasing w. That part simply follows the method used in [1] and is not introduced here due to space limitation.)

Next we implement the bursty loss model based on the network in Fig. 4. Since the bursty loss model is not implemented in the ns simulator, we have derived such a bursty loss model from ns's *basic error model class*. In our simulation, the bursty loss model is placed before the queue of link from nodes G to R, and all the queue buffers are set big enough (5000 packets) to avoid congestion loss. With p=0.001, the timeout probability of a TCP Reno connection is found to be 0.95 with w=41.27 packets. This result matches very well what is shown in Fig. 3.

Fig. 5 compares the estimates of Q and P_{RE1} (the timeout probability in [1]) with simulation results based on the network of Fig. 4. This shows that our proposed analytical model is more accurate. Note that the work in [1] only considers the timeout triggered by reason 1 (P_{TO1}). Therefore it has underestimated the timeout impact on the performance of TCP Reno.

Finally we simulate the actual performance of TCP Reno based on the network in Fig. 5, where drop-tail FIFO routers are

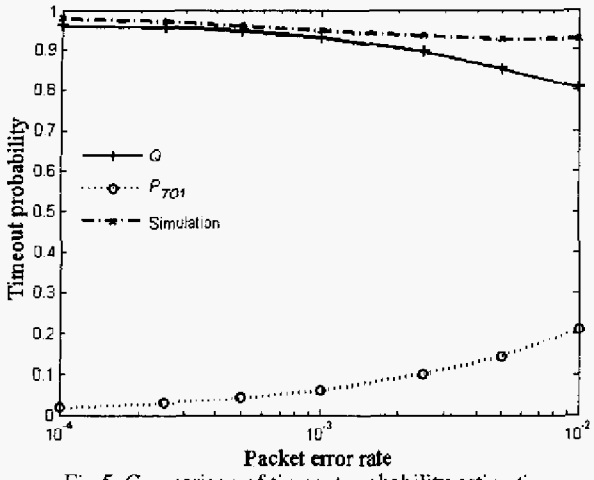

Fig.5. Comparison of timeout probability estimation

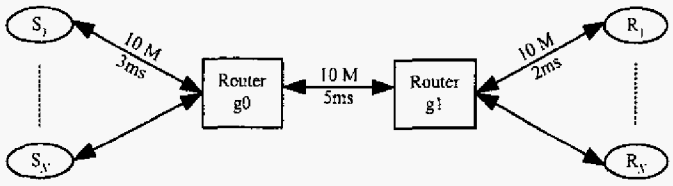

Fig. 6. Simulated network for FIFO drop-tail queue

TABLE 1
Comparison of FIFO queue and bursty loss model simulation results

	FIFO queue			Bursty loss model	
Number of Flows	p	Q	w	Q	w
10	0.010622	0.001	14.093	0.917	12.783
20	0.024579	0.076	9.197	0.935	8.388
30	0.039481	0.116	7.311	0.951	6.653
40	0.051843	0.198	6.648	0.963	5.825

used. Assume there are N concurrent TCP flows. The bottleneck queue buffer size (at the output link from router g0 to g1) is set to 100 packets. Table 1 summarizes the results under "FIFO queue," where the dropping probability p has been normalized to represent the probabiltiy of loss event (loss occurring in the same round are counted as a single loss event). For comparison with the FIFO queue results, another set of simulations based on the bursty loss model are done using the same set of p. The network simulated is still the network in Fig. 5, but there is only one sender/receiver pair. The bursty error model is placed before the output queue of the link from g0 to g1, and all the queue buffers are set large enough to avoid congestion loss. The results are shown under "Bursty-loss model" in Table 1. We see that there are significant differences between the values of Q in the two tables. The above comparison assumes that the influence of drop-tail FIFO queue to TCP connections can be represented solely by loss probability. Although the drop-tail FIFO queue also adds queue delay, its influence to timeout probability should be negligible.

The discrepancy in the timeout probability results shows that the bursty loss model does not capture the loss characteristics of TCP under drop-tail FIFO queues well. Considering the significant influence of timeout on TCP performance, using this bursty loss model to analyze the performance of TCP is indeed problematic.

We notice that the throughput model in [1] shows good match in the reported performance. However, [13] indicates that the correctness of the model in [1] may be due to error cancellation between the model for TCP and the model for packet losses. Comparing with the modelling methods proposed in this paper, the method used in [1] underestimates the timeout probability. Their results may match a loss pattern in which the dropping of successive packets in the same round is of a probability less than 100%. This again emphasizes that a proper determination of the loss model is of utmost importance in TCP performance study.

We also notice that E. Altman *et al.* [13] have proposed a throughput model based on the general stationary ergodic loss. Their analysis, as in [1-7], does not consider the particular behavior that TCP Reno shows in fast recovery. Our future work in this area will focus on combining their work with the methods presented in this paper for a more precise TCP performance model.

VI. Conclusion

This paper studies the timeout probability of a TCP Reno connection under the bursty loss model. We find that the impact of timeout has been underestimated. With a detailed analysis on three timeout reasons, our work captures the impact of timeout more precisely than previous works. Surprisingly, we find that this more precise representation of timeout probability does not match the actual performance of TCP under drop-tail FIFO queues. Therefore we conclude that the bursty loss model is incapable of capturing the behavior of drop-tail FIFO queues, and using bursty loss model to analyze TCP performance is flawed.

References

[1] J. Padhye, V. Firoiu, D. F. Towsley and J. F. Kurose, "Modeling TCP Reno Performance: A Simple Model and Its Empirical Validation," IEEE/ACM Trans on Networking, pp.133-145, vol.8 , I.2 , Apr. 2000.

[2] B. Sikdar, S. Kalyanaraman and K.S. Vastola, "Analytic models for the latency and steady-state throughput of TCP Tahoe, Reno, and SACK," IEEE/ACM Trans. on Networking, vol.11, I.6, Dec. 2003.

[3] D. Zheng, G.Y. Lazarou and R. Hu, "A stochastic model for short-lived TCP flows," in Proc. IEEE ICC2003, pp. 76-81, vol. 1, May 2003.

[4] S. Fortin and B. Sericola,"A model of TCP in wide area networks," in Proc. MASCOTS2002, pp.453-462, Oct. 2002.

[5] N. Cardwell, S. Savage and T. Anderson, "Modeling TCP latency," in Proc INFOCOM2000, pp. 1742-1751, vol.3, Mar. 2000.

[6] B. Sikdar, S. Kalyanaraman and K.S. Vastola, "An integrated model for the latency and steady state throughput of TCP connecitons," Perform. Eval., vol.46, no. 2-3, pp.139-154, Sep. 2001.

[7] T. J. Ott, T. V. Lakshman, and L. H. Wong, "SRED: Stabilized RED", in Proc. INFOCOM1999, pp. 1346-1355, vol.3, March 1999.

[8] M. Allman, V. Paxon and W. Stevens, "TCP Congesiton Control," in RFC2581, Apr. 1999.

[9] V. Paxson and M. Allman, "Computing TCP's Retransmission Timer," in RFC2988, Nov. 2000.

[10] H. Handley, S. Floyd, J. Padhye and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification," in RFC3448, Jan. 2003.

[11] UCB/LBNL/VINT network simulator – ns (version 2).

[12] K. Claffy et al., "The nature of the beast: Recent traffic measurements from an Internet Backbone," In Proc. of INET'98, July 1998.

[13] E. Altman, K. Avrachenkov, and C. Barakat, "A Stochastic Model of TCP/IP with stationary random losses," ACM Computer Communication Review, vol.30, no. 4, pp. 231-242, Oct. 2000.