Analysis of RTO Caused by Retransmission Loss to Combat Channel Noise

Bhavika Gambhava
Dharmsinh Desai University,
Nadiad, Gujarat, India

N. J. Kothari
Dharmsinh Desai University,
Nadiad, Gujarat, India

Dr. K. S. Dasgupta
Space Application Centre,
Ahmedabad, India

ABSTRACT
A Retransmission Time Out (RTO) is inevitable, when the retransmission of a packet fails to reach the receiver. An RTO compels TCP to reduce packet flow drastically. However, in case of an RTO resulting from retransmission failure caused by the channel noise, reduction in the flow is inappropriate. The problem is compounded when a TCP sender is forbidden to continue transmission till the occurrence of the timeout. In this paper, we investigate the impact of such RTOs with the help of an empirical mathematical analysis. The analysis presented in the paper calculates the idle period of the sender in terms of number of RTTs, which depends on the value of congestion window before the timeout. The mathematical analysis is supported by the results of simulation based experiments and the evaluation of a scheme that improves TCP performance in case of avoidable timeouts caused by the loss of retransmission on an erroneous wireless link.

General Terms
Algorithm, Experimentation, Verification, Performance.

Keywords
RTO, SACK_OK, cwnd, dupacks

1. INTRODUCTION
TCP [1] congestion control, which is based on principles of packet conservation, slow start and Additive Increase Multiplicative Decrease (AIMD), is responsible for the sustained stability of the Internet despite growth in traffic and topology [7][8]. If availability of resource is not a constraint in wireless environment, the multiplicative decrease to control congestion is inappropriate while facing a packet loss due to corruption with a poor channel quality [3][4]. Additionally, propagation delay is the dominant part in broadband satellite links as compared to other components of latency i.e. transmission delay and queuing delay. The connections traversing this kind of links will be penalized by the bias of TCP congestion control towards flows with high round trip time (RTT) [2]. Therefore, TCP exhibits poor performance over wireless networks with high delay and BER [14] [15].

TCP infers a packet loss based on two indications: (i) a timeout occurrence and (ii) the receipt of duplicate acknowledgments (*dupack*) [16]. To combat packet losses, *fast recovery* is initiated after arrival of three *dupacks*. The *fast recovery* mechanism does not help TCP loss recovery if a retransmitted packet is also dropped. Retransmission Time Out (RTO) becomes indispensable in this situation. Since retransmissions can also be caused by the packets lost on unreliable or error-prone transit paths, error prone transit paths can also lead to loss of retransmission [6]. Hence, the performance of TCP is severely degraded due to an unnecessary timeout, when the cause of failure of retransmission is also the channel noise. This kind of timeouts has been identified as *avoidable timeouts* [6].

TCP may experience unnecessary RTO in any of the following situations: (i) Timeout because of insufficient *dupacks* available for attempting *fast retransmit* [9]. (ii) Spurious timeout, when no packets have been lost [12]. (iii) Timeout due to channel noise causing loss of retransmission. However, the solutions like F-RTO algorithm [12] do not enable TCP to utilize the time period before RTO, when the timeout is avoidable. In addition, the schemes like Eifel [10] [13] and others as mentioned above [9], do not address the issue of retransmission loss caused by channel noise, which in turn leads to RTO needlessly.

RTO reduces packet flow drastically. Throughput of TCP is affected after RTO because of two major reasons: (i) *cwnd* is reset to one. (ii) A TCP sender is prevented from continuing transmission during loss recovery when the number of outstanding packets reaches the maximum window (*Maxwnd*) limit [6]. Without having pending retransmissions and permission to send new packets, a blocked TCP sender may waste considerable amount of time before timeout which could be a multiple of 500ms [9]. In this paper, we find out the idle period of a TCP sender remaining unutilized in terms of number of RTTs with the help of an empirical mathematical analysis.

The mathematical analysis presented in section 2 quantifies the impact of an avoidable timeout in terms of the RTTs wasted by a TCP sender. A scheme to combat the channel errors leading to such timeout is discussed in section 3. We demonstrate improvement in the performance of TCP by utilizing these RTTs with the help of simulations in section 4.

2. MATHEMATICAL ANALYSIS
In this section, we derive the number of additional RTTs a TCP sender can utilize during an avoidable timeout, with the help of a mathematical analysis. The analysis also determines the new *cwnd*, which is updated in every RTT as per *congestion avoidance*. It is based on a premise that during entire timeout interval, the first packet transmission and its retransmission are corrupted by channel errors. Hence, RTO occurrence is not due to extreme congestion in the network.

Before the timeout, TCP sender receives *dupacks* equal to one less than *Maxwnd* after a loss of retransmission in absence of any other loss. The number of RTTs utilized in turn, indicates the

number of RTTs remaining unutilized (WRTT) because the total number of RTTs is considered constant (i.e. 25 in this case) [1]. Following analysis is carried out based on the assumption that WRTT is less than the RTTs required for incrementing *cwnd* up to *Maxwnd* limit.

No. of outstanding packets at the end of *fast recovery*

= *Maxwnd – Retrans* (Packets retransmitted = 1)

= *no.* of *dupacks* at the sender.

Time utilized by a sender before the timeout

= *URTT x RTT*

(*URTT* = no. of RTTs the sender utilizes before the timeout)

RTT is assumed constant keeping constant delay of satellite links into consideration [18]. The transmission of new packets in *fast recovery* is governed by *pipe* and *cwnd* relationship.

$cwnd_p$ = *cwnd* before *fast recovery*

$cwnd_f = \frac{cwnd_p}{2}$ = *cwnd* after initiating *fast recovery*

$pipe_t = cwnd_f - 1$ (transmission begins at $pipe_t$)

Once the *pipe* restriction is overcome, the sender receives *dupacks* equal to $pipe_t$. It transmits $pipe_t$ packets in every RTT. *dupacks* arriving at the sender are divided in to two groups. (i) *duprp*: *dupacks* arriving before start of repetitive transmission of $pipe_t$ packets and (ii) *dupacks* arriving in each RTT due to transmission of $pipe_t$ packets (will be same as $pipe_t$).

Number of RTTs utilized by the sender can be calculated by finding RTTs demanded by the above mentioned *dupacks*.

Calculation of RTTs for duprp

The number of *dupacks* received before the beginning of transmission period of $pipe_t$ packets (*duprp*), depends on following factors.

da *dupacks* received from the window facing the loss

db *dupacks* from the packets transmitted in response to normal acknowledgments for the window

dc *dupacks* from the transmission triggered by first two *dupacks*

dd *dupacks* resulting from the packets transmitted after overcoming pipe restriction in RTT immediately after entering *fast recovery*, if any.

pol position of loss in window, $cwnd_p$

Therefore, *duprp* = *da + db + dc + dd*

Where, $da = cwnd_p - pol$

$db = pol - 1$

$dc = 2$

dd = *dupacks in the RTT after loss – dupacks up to* $pipe_t$

= *dupacks in the RTT after loss – [dupacks required to reach* $pipe_t$ *from* $pipe_i$ $(dup_t) + 2]$

$pipe_t$ = *initial pipe* $(pipe_i)$ - dup_t = $cwnd_p - 1 - dup_t$

$cwnd_f - 1 = cwnd_p - 1 - dup_t$

$dup_t = cwnd_f$

dupacks up to $pipe_t$ = $dup_t + 2$

(first two dupacks, before initiation of fast recovery*)*

= *cwndf + 2*

$dd = cwnd_p - pol - (cwnd_f + 2)$

$duprp = (cwnd_p - pol) + (pol - 1) + 2 + [cwnd_p - pol - (cwnd_f + 2)]$

$= cwnd_p + 1 + cwnd_f - pol - 2$

$(cwnd_p - pol) > 1$ and $(cwnd_f - pol - 2) > 0$

$= cwnd_p + 1$

$(cwnd_p - pol) > 1$ and $(cwnd_f - pol - 2) <= 0$

$duprp = 2cwnd_p - (pol + 1)$ $(cwnd_p - pol) <= 1$

Therefore, the number of utilized RTTs before timeout,

$$URTT = \frac{Maxwnd - \text{Re}trans - duprp}{cwnd_f - 1} \quad (cwnd_p - pol) <= 1$$

$$URTT = \left(\frac{Maxwnd - \text{Re}trans - duprp}{cwnd_f - 1}\right) + 1$$

$(cwnd_p - pol) > 1$ ...2.1

Time wasted before Timeout = *(RTO – URTT x RTT)*

No. of RTTs wasted in congestion episode,

$$WRTT = \frac{RTO - URTT \times RTT}{RTT} \quad \text{... 2.2}$$

From equation 2.1 and 2.2,

URTT = f(1/$cwnd_f$*, Retrans)*

Since *Retrans* = 1 i.e. constant,

URTT = f(1/$cwnd_f$*)*

As URTT and *cwnd* are inversely proportional, *URTT* will be large for small values of *cwnd*. On the other hand, for large values of *cwnd*, *WRTT* will be more.

$cwnd_u$ = updated *cwnd* after *fast recovery*

When *fast recovery* is exited without RTO occurrence,

$cwnd_u = cwnd_f$

However, in conventional algorithm after RTO, $cwnd_u = 1$

In an attempt to avoid the negative impact of RTO, the sender should be allowed to continue transmission during the period of WRTT and cwnd should be simultaneously updated.

Expected *cwnd* after RTO,

$cwnd_u$ = $(cwnd_f)$ + increments by WRTT utilization … *2.3*

Since *cwnd* is incremented by one at the end of each RTT, during the period utilized beyond *URTT* (*WRTT*) the additional increment in *cwnd* will be equal to *WRTT*.

Using equation 2.3, $cwnd_u = cwnd_f + WRTT$

[1] Maximum number of RTTs = $\frac{RTO}{RTT} = \frac{1 \text{ sec}}{40\,m \text{ sec}} = 25$

Therefore, $cwnd_u = f(WRTT, cwnd_f)$ ...2.4

If the timeout is considered to be unavoidable because of possibility of congestion,

$WRTT = 0$ and $cwnd_u = 1$.

From 2.4, we can observe that updated cwnd after timeout depends on number of wasted RTTs and $cwnd_f$ means *cwnd* before fast recovery. If *WRTT* is large, more RTTs are wasted and proposed modifications performs better.

3. MODIFICATIONS IN SACK TCP

In wireless networks, channel noise dominates congestion. The TCP sender can "undo" the *cwnd* reduction as a possible response to packet corruption, after finding the fact that a single packet loss had been due to corruption rather than congestion [5]. Various approaches for modifying TCP to differentiate a random loss and avoid unnecessary reduction in *cwnd* have been proposed [5][14]. A new algorithm has been proposed earlier to combat the cases of loss of retransmission due to random losses, restore TCP performance by avoiding reduction of *cwnd* and utilize RTTs before RTO, which would have been wasted unnecessarily otherwise [6]. While SACK TCP is being widely deployed in the Internet, it is also a fact that SACK information is not used in making congestion control decision [17]. The algorithm employed for improving TCP performance while recovering from loss of retransmission over erroneous link is based on inferences made from implicit information conveyed by the SACK blocks, to utilize the RTTs of the interval before RTO when the bandwidth is available.

The modifications shown in Figure 1 are introduced in the original SACK TCP [16] [29].

A new flag SACK_OK is initialized with one on arrival of the first *dupack* [14]. It is continuously updated with every new *dupack*. The SACK_OK flag remains one till SACK blocks arriving with *dupack* report continuous increment in the packets reaching the receiver following the first loss.

If SACK_OK flag remains one throughout the *fast recovery*, the modified TCP increments *cwnd* and continues transmission. The transmission does not cease due to the fact that all packets following retransmitted one are selectively acknowledged and congestion is ruled out. *cwnd* is updated and transmission continued further at the end of every RTT till RTO as long as SACK_OK flag is one. Conversely in timeout, if SACK_OK flag is zero, *cwnd* is reset to one conventionally.

4. SIMULATIONS & RESULTS

The main objective of this paper is to evaluate effectiveness of the modified TCP particularly in presence of errors leading to retransmission loss. Simulations were carried out on ns2.26 simulator [19]. The simulation topology shown in Figure 2 is designed to avoid congestion completely so that negative effect of loss of retransmission can be examined exclusively for random errors.

A single file transfer protocol (FTP) flow is used to make diagnosing problem easier than attempting to diagnose the problem in a dynamic network with other competing traffics [20].

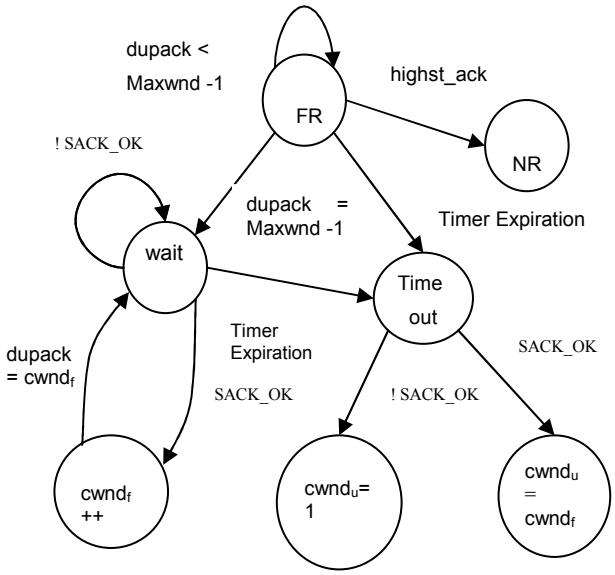

Figure 1 State Transition Diagram for *fast recovery*

NR: Normal State of TCP in absence of loss recovery
FR: TCP in *fast recovery*, $cwnd_f$: *cwnd* in *fast recovery*, $cwnd_u$: *cwnd* after RTO

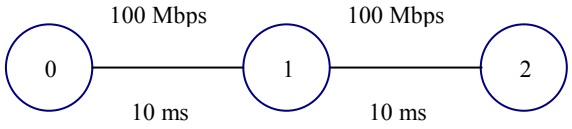

Figure 2 Network with Erroneous Link

It has been pointed out previously that with minimum RTO of 1 sec., TCP is able to avoid bad timeouts without impairing performance significantly [11]. The results of simulation carried out with packet error rates varying from 0.01 to 0.03 in step of 0.01 are discussed in this section. The total time of simulations was 300 seconds with each error rate affecting the link between nodes 1 and 2 for 100 seconds. Observations from the timeouts avoided with the help of modifications are given in Table 1 and Table 2.

The original TCP stops transmission after sending 49 packets beyond the lost packets[2]. It can be seen in Table 1 that transmission continues for a longer time in case of the modified TCP by utilizing additional RTTs during the timeout interval in all cases. The time instant at which transmission stops in the modified TCP is very close to the instant of timeout in case of the *avoidable timeout* occurring at 152.99sec. This in turn results in to utilization of the maximum number of RTTs possible during this timeouts interval.

Number of utilized RTTs in a timeout interval is proportional to cwnd as pointed out in the mathematical analysis presented in Section 2. As a result, all the time instants are very close in case of the timeout occurring at 282.25sec, where the minimum additional RTTs are utilized by the modified TCP. A small *cwnd* also allows

[2] In case of original TCP, the sender will get 49 *dupack*s unless it recovers from a loss by timeout, when the *Maxwnd* is 50 and retransmission is also lost.

the original TCP to utilize more RTTs to reach the limit of maximum outstanding packets decided by *Maxwnd* parameter. This leaves a very little scope for improvement. The corresponding values given in Table 2 confirm the observations.

The maximum number of RTTs, which can be utilized, is 25 in the given topology with RTO interval equal to 1 second[2]. While continuing transmission during timeout interval, occurrence of another loss terminates the transmission and causes return to the conventional timeout-based recovery, as congestion cannot be ruled out. The modified TCP avoids utilizing additional RTTs further, in this situation. Because of this, the utilized RTTs of the modified TCP are usually less than 25 as seen in Table 2.

Table 2 shows gain in packets delivered during RTO by avoiding *cwnd* reduction and utilizing RTTs for packet transmission with respect to each *avoidable timeout* indicated earlier in Table 1. It also indicates corresponding *cwnd* just before timeouts in the original TCP. The performance gain is enhanced further by updating *cwnd*, which is illustrated in Figure 3. As seen in Table 2, the RTO at 153.53sec utilizes 14 RTTs additionally. Since *cwnd* is updated in each additional RTT as per congestion avoidance scheme, rise in *cwnd* will be identical to additionally utilized RTTs. Thus, *cwnd* increments from 5 to 19 in this case, which can be seen in Figure 3. Note that original SACK TCP stops transmission after sending 49 packets beyond the lost packets, whereas modified scheme allows TCP to transmit 224 packets. This leads to approximately 350% improvement in the performance of the modified TCP, which is reflected in Table 2.

Table 1
RTO Time Instances

Pkt. Error Rate	*cwnd* in *fast rec.*	T_{co} Sec.	T_{cu} Sec.	T_{to} Sec.
0.01	7	60.25248	60.33416	60.97268
0.02	5	152.9974	153.5385	153.5553
0.02	6	181.4254	181.8832	182.1036
0.02	5	194.3459	194.5499	194.9432
0.03	3	282.1015	282.223	282.2568

T_{co} = time instant where transmission stopped in original TCP
T_{cu} = time instant where transmission stopped in modified TCP
T_{to} = time instant of timeout occurrence

Table 2
Relative Performance Improvement

Pkt. Error Rate	Time out Instant	Utilized RTTs in TCP		% imp. (Pkts)
		Org.	Mod.	
0.01	60.97	7	9	34
0.02	153.53	11	25	350
0.02	182.1	8	19	264
0.02	194.94	8	13	70
0.03	282.25	21	24	30

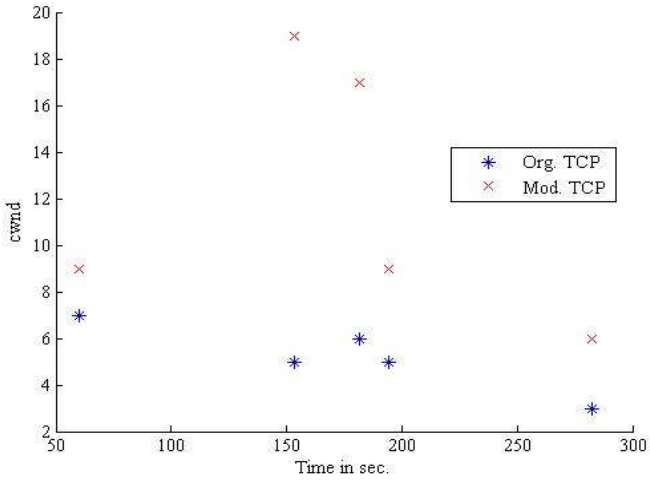

Figure 3 Updated *cwnd* vs Time

The difference in utilized RTTs and consequently performance of both schemes depend on detection of another loss as well as *cwnd* in *fast recovery*. Another loss will prevent the modified TCP from continuing transmission during timeout interval as described in the previous section. The gain in *cwnd* in Figure 3 is directly proportional to the additional RTTs utilized for the reasons discussed earlier.

5. CONCLUSIONS

The wireless links are often mainly characterized by link errors and also large propagation delay. Both factors decrease the acceleration of TCP transmission rate and subsequently, in the overall link utilization. This paper investigates the negative impact of an RTO resulting from a loss of retransmission caused by the channel errors with the help of a mathematical analysis. The analysis determines the RTTs during the timeout interval in which packet transmission ceases and increment in congestion window stops unnecessarily in absence of congestion. The analysis shows that performance enhancement can be higher when *cwnd* is substantially large.

An approach to overcome this performance degradation by utilizing the RTTs and updating *cwnd,* is evaluated in this paper. Simulation based experiments conducted for different error rates revealed that number of avoidable timeouts increases in case of a higher error rate. However, a very high error rate creates a situation similar to congestion in the network. Since the algorithm depends on implicit information in SACK blocks, it reverts to conventional scheme in this situation. Nevertheless, results of simulations indicate improvement in the overall performance in presence of the modifications. Since this kind of RTO does not occur frequently, the performance gain may be localized.

6. ACKNOWLEDGMENTS

We acknowledge the sincere support provided by Ms. Miral Patel during simulation based experiments. We are also heartily thankful to Mr. N. G. Vasantkumar of Space Application Centre, ISRO, Ahmedabad.

7. REFERENCES

[1] J. Postel. Transmission Control Protocol. *RFC 793*; Sep 1981.

[2] G. Xylomenos. Multi Service Link Layers: An Approach to Enhancing Internet Performance over Wireless Links. *Ph. D. thesis, University of California* ; 1999.

[3] M. Allman, S. Dawkins, D. Glover, J. Griner, D. Tran, T. Henderson, J. Heidemann, J. Touch, H. Kruse, S. Ostermann, K. Scott, J. Sanke. Ongoing TCP Research related to Satellites. *RFC 2760* ; February 2000.

[4] C. Parsa, J. Garcia-Luna-Aceves. Improving TCP Performance over Wireless Networks at the Link Layer. *ACM Mobile Networks and Applications Journal* ; 1999.

[5] K. Pentkousis. TCP in Wired-Cum-Wireless Environments. *IEEE Communications Surveys & Tutorials* ; 2000.

[6] N. J. Kothari, B. M. Gambhava, K. S. Dasgupta. RTT Utilization by Detecting Avoidable Timeouts. *14th IEEE International Conference on Networks* ; September 2006.

[7] W. Stevens. TCP Slow start, Congestion Avoidance, Fast Retransmit, Fast Recovery. *RFC 2001*; January 1997.

[8] M. Allman, V. Paxson, W. Stevens. TCP Congestion Control. *RFC 2581* ; April 1999.

[9] M. Allman, H. Balakrishnan, S. Floyd. Enhancing TCP's Fast Recovery using Limited Transmit. *RFC 3042* ; 2000.

[10] A. Gurtov, R. Ludwig. Evaluating the Eifel Algorithm for TCP in a GPRS Network. *In Proc. of European Wireless.* Florence, Italy, February 2002.

[11] A. Kesselman1, Y. Mansour. Optimizing TCP Retransmission Timeout, *P. Lorenz and P. Dini (Eds.): ICN 2005* ; Springer-Verlag Berlin Heidelberg ; 3421 : 133–140.

[12] P. Sarolahti, M. Kojo. Forward RTO-Recovery (F-RTO): An Algorithm for Detecting Spurious Retransmission Timeouts with TCP and the Stream Control Transmission Protocol (SCTP). *RFC 4138* ; August 2005.

[13] R. Ludwig, R. H. Katz. The Eifel Algorithm: Making TCP Robust Against Spurious Retransmissions. *Computer Communications Review*. January 2000 ; 30 : 30 – 36.

[14] N. J. Kothari, K. S. Dasgupta. Performance Enhancement of SACK TCP Protocol for wireless Network by Delaying Fast Recovery. *IEEE International Conference on Wireless & Optical Communication Networks* ; April 2006.

[15] F. Hu, N. Sharma. Enhancing Wireless internet Performance. *IEEE Communications Surveys*; 2002.

[16] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow. TCP Selective Acknowledgement Options. *RFC 2018*; Oct 1996.

[17] E. Blanton, M. Allman, K. Fall, L. Wang. A Conservative Selective Acknowledgment (SACK)-based Loss Recovery Algorithm for TCP. *RFC 3517*; April 2003.

[18] S. Floyd, J. Mahdavi, M. Mathis, M. Podolsky. An Extension to the Selective Acknowledgement (SACK) Option for TCP. *RFC 2883*; July 2000.

[19] K. Fall ,K. Varadhan. ns Notes and Documentation, 2000.

[20] M. Allman, A. Falk. On the Effective Evaluation of TCP. ACM Computer Communication Review, Oct 1999.