The Effect of the Initial Window Size and Limited Transmit Algorithm on the Transient Behavior of TCP Transfers

Urtzi Ayesta
France Telecom R&D
905 rue Albert Einstein
06921 Sophia Antipolis Cedex, France
e-mail: Urtzi.Ayesta@francetelecom.com

Konstantin E. Avrachenkov
INRIA Sophia Antipolis
2004 route des Lucioles, B.P.93
06902, Sophia Antipolis Cedex, France
e-mail: k.avrachenkov@sophia.inria.fr

Abstract

We study the impact of the modifications proposed for TCP in the context of short file transfers. The two most important proposals are the increment of the initial window size (IW) [3] and the Limited Transmit algorithm (LT) [1]. We analyze analytically and by simulations the effect of these proposals on the TCP latency. We demonstrate that the LT proposal reduces the TCP latency similarly to the IW proposal however LT is less aggressive than IW. Nevertheless, in the context of short file transfers we point out a scenario, when upon a single packet loss the sender times out even if IW or LT are enabled. This harmful scenario happens when the very last packets of the file transfer are lost. Therefore to avoid this situation we propose a new modification which is based on reducing the number of duplicate acknowledgements TCP reacts to. In order to avoid the potentially harmful effect of needless retransmissions on the network load we suggest to implement the new modification only for short file transfers.

Key words. TCP/IP, short file transfers, Initial Window, Limited Transmit algorithm.

1 Introduction

It has been observed that TCP loss recovery mechanism does not work properly when the congestion window of the TCP sender is small. This can happen either in transient or steady-state behavior, as for example due to the limit imposed by the receiver advertised window or because of the constraints imposed by a connection with a small bandwidth-delay product link. In order to avoid this harmful effect several suggestions have been proposed recently. Probably the two most important ones are the increase of the initial congestion window (IW proposal [3]) and the Limited Transmit algorithm (LT proposal [1]). In both cases the aim is to decrease the probability of the event that a TCP connection goes into timeout. A timeout is very harmful for the TCP performance since based on conservative approach its value is normally set to several times greater than the average RTT (round-trip time) [24], which implies a reduction of the average throughput in the case of long-live TCP sessions and an increment of the transfer time in the case of short-live TCP sessions.

So far the same TCP algorithm is used regardless the size of the file to be transfered. However, it is known (see, e.g. [15]) that a TCP session typically belongs to one of the following two kinds: "mice" or "elephants". Most TCP sessions are "mice" with a small size, but a smaller amount of "elephants" (in terms of flows) is responsible for the largest amount of transfered data (in terms of bytes). Therefore, it seems to be natural to use a little bit different TCP modifications for different types of flows. In the present paper we would like to initiate the research on this subject. Still we first want to concentrate our efforts on short-live TCP transfers. We evaluate both analytically and by simulations the performance of the IW and LT proposals. We point out two situations when upon a single loss event the sender will inevitably timeout even with the IW and LT proposals. We propose further modifications in order to decrease the probability of timeout events. We suggest to use these modifications only for "mice" type TCP transfers.

The rest of the paper is organized as follows, Section 2 outlines a recursive approach for the calculation of the expected transfer time for short TCP session. The detailed version of this approach can be found in [6]. Section 3 provides the performance analysis for IW and LT proposals. In section 4 the numerical results are presented and further possible TCP modifications are discussed.

2 The expected latency of TCP transfers

In this section we explain the model we use to calculate the expected time of a TCP file transfer, for a complete explanation we refer the reader to [6]. The expected latency L is computed conditioning on the number of losses. The input parameters for the model are the packet loss probability p, the average round trip time RTT, the document size to be transfered $\bar{y}$, the initial window size W_0, the initial slow start threshold W_{SS}, the maximum receiver's advertised window M and the number of packets the receiver acknowledges by one ACK b (delAck option). The fluid model [5] approach is used to represent the evolution of the congestion window. In particular, this means that instead of a discrete number of packets, a continuous volume of data (of course, with the same size) is transfered over the network. It turns out that the fluid model approach is more analytically tractable than the discrete one. As in all related previous works [12, 20, 22] the assumption that packets are lost independently with probability p is used. Thus the expected latency for a file size of $\bar{y}$ bytes is given by:

$$L(\bar{y}, W_0, W_{SS}) = \sum_{k=0}^{\infty} p(k|\bar{y}) L_k(\bar{y}, W_0, W_{SS}), \quad (1)$$

where $L_k(\bar{y}, W_0, W_{SS})$ is the conditional expected latency given that exactly k losses happened during the file transfer.

$$\begin{aligned} & L_k(\bar{y}, W_0, W_{SS}) = \\ & = E[\text{latency}|p, RTT, \bar{y}, W_0, W_{SS}, \text{loss no.} = k]. \end{aligned}$$

In the case of no losses, L_0 can be easily calculated from the analysis of the TCP congestion window evolution. We calculate $L_k(\bar{y}, W_0, W_{SS})$ using the recursive approach as outlined below.

First we calculate $p(k|\bar{y})$ which is the probability of having k retransmission when $\bar{y}$ bytes are transmitted [6].

$$p(k|\bar{y}) = \frac{e^{-\lambda(\bar{y}+kMSS)}(\lambda(\bar{y}+(k-1)MSS))^k}{k!}$$

The parameter λ corresponds to the amount of data successfully transmitted between two consecutive losses. Given that packets are lost in an independent fashion the number of packets successfully sent between two consecutive losses has a geometric distribution. To adapt this assumption to the fluid model, the standard approximation of the geometric distribution by an exponential distribution is used. Therefore the parameter λ of the exponential distribution can be determined from the following relation

$$\begin{aligned} & \tfrac{1}{\lambda} = E[\text{transmitted bytes} \\ & \text{between two consecutive losses}] = \tfrac{MSS}{p}. \end{aligned}$$

The model is based on the TCP NewReno and SACK versions. Under this assumptions both flavors of TCP would behave in similar way. Let us define as dup_t the number of duplicate ACKs the sender must receive in order to infer a loss has occurred[1] and start running fast recovery and fast retransmission algorithms. It is important to remark that dup_t does not depend on b, since the receiver acknowledges every packet received out of order. Hence upon a packet loss event the number of ACKs the TCP sender will receive does not depend on b.

In general when a loss occurs, the congestion window of the sender will continue sliding forward until the lost packet gets to the left most position. If the value of the congestion window is less than $1 + dup_t$ the TCP session will timeout. Given the initial settings of a TCP session we define the parameter y_{to} as the amount of bytes sent until this value is reached. Therefore the value for y_{to} is simply given by the amount of bytes sent y_f up to the final sending rate W_f reaches the value $(1 + dup_t)MSS/RTT$. However, if the initial window size is greater or equal to $1 + dup_t$ the sender will not timeout. Thus, we have

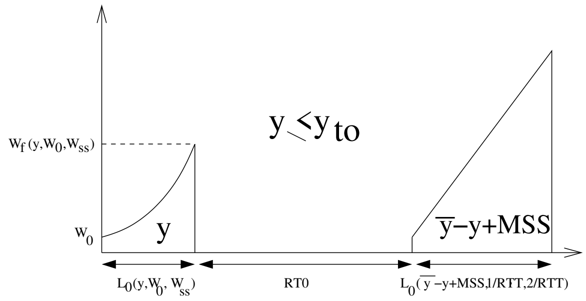

Figure 1: Sending rate evolution when a packet is lost before y_{to} bytes are sent

$$\begin{aligned} y_{to} &= y_f((1+dup_t)MSS/RTT, W_0, W_{SS}) \\ & \quad 1\{W_0 < (1+dup_t)MSS/RTT\} \end{aligned}$$

There is yet another situation when the sender will inevitably timeout. Namely, if a loss occurs when the remaining amount of data is less than $dup_t MSS$, no matter what the actual value of the sending rate is, the sender will not receive three duplicate ACKs and will have to rely on a timeout to detect the loss.

These two scenarios can be very harmful from the performance point of view. The TCP retransmission time RTO is based on measured round-trip times between sender and receiver as specified in [24] and therefore to avoid retransmissions of packets that are only delayed and not lost the minimum RTO is conservatively chosen to be 1 second. For computational purposes we substitute the value of the timeout RTO with the expression $Max(1.0, 4 * RTT)$ as suggested in [14]. Anyhow at some future point it might be suggested that a smaller value minimum leads TCP to a better performance [24].

Let us analyze what can happen during the transmission of a file when a packet is lost. We consider three scenarios depending the amount of bytes y sent before the loss occurs, see (Figures 1, 2 and 3). In Figure 1 we observe that when $y \leq y_{to}$ bytes are sent, the total transfer time will be equal to the sum of the time required to send y bytes, the retransmission time RTO and the time required to send $\bar{y} - y + MSS$ bytes. In the second scenario, Figure 2, the TCP sender will detect the loss event upon the reception of three duplicate ACKs. In this case the TCP sender will halve its sending rate W_f and it will transmit $W_f RTT/2 - MSS$ new bytes in addition to the lost MSS. In the next round upon the reception of all

[1] dup_t is commonly set to three.

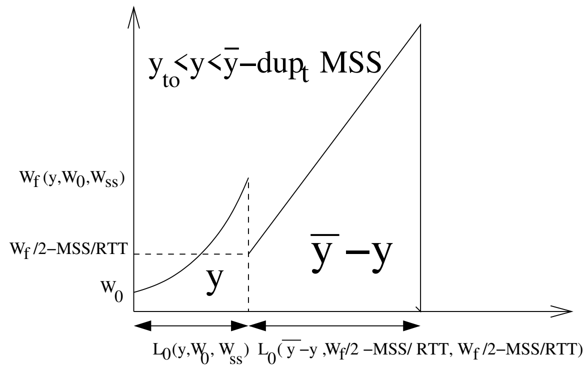

Figure 2: Sending rate evolution when a packet is lost when y ($y_{to} < y < \bar{y} - dup_t MSS$) bytes are sent

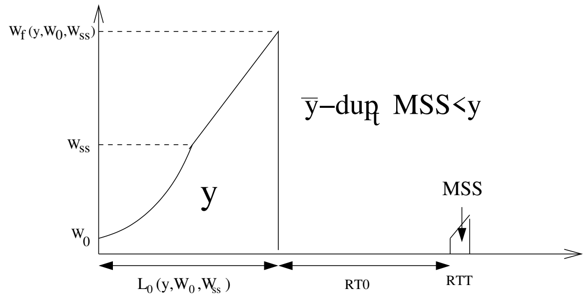

Figure 3: Sending rate evolution when a packet is lost when y ($\bar{y} - dup_t MSS < y$) bytes are sent

the ACKs it will transmit $W_f RTT/2$ new bytes. For the sake of simplicity, in the fluid model we consider that the remaining data is $\bar{y} - y$ and the new sending rate is set to $W_f/2 - MSS/RTT$. This approximation on the remaining data allows us to keep exact track of the evolution of the sending rate. In this scenario the total transfer time will be equal to the sum of the time required to send y bytes with the initial parameters and the time required to send $\bar{y} - y$ after the loss event with the new TCP settings (W_0 and W_{SS}). The third scenario, Figure 3, corresponds to the situation when a loss occurs and the amount of data remaining is less than $dup_t MSS$. In this case the transfer time will be equal to the sum of the time required to send y bytes, the retransmission time RTO and the time required to send the last packet (around one RTT).

After a timeout, the value of W_0 and W_{SS} are changed according to [2]. In particular, the value of W_0, regardless the value of the sending rate just before the timeout W_f, must be set to no more than 1 packet and the new value for the slow-start threshold is

$$W_{SS} \approx \max(W_f/2, 2MSS/RTT).$$

Next, let us define $f(y, k, \bar{y})$ as the density function of the amount of data y that is sent before the first loss occurs given that k loss events happened during the transmission of $\bar{y}$ data. This density is related to the uniform distribution, namely, the distribution of the amount of bytes sent before the first loss y is the minimum of k independent uniformly distributed random variables [13]. Thus the corresponding density function is given by

$$f(y, k, \bar{y}) = f(y|N_{\bar{y}} = k) = \frac{k}{\bar{y}}\left(1 - \frac{y}{\bar{y}}\right)^{k-1}$$

Putting together all above results and using the recursive approach the expected conditional latency of a TCP file transfer can be calculated by

$$\begin{aligned}
&L_k(\bar{y}, W_0, W_{SS}) = \\
&= \int_0^{max(0,min(y_{to},\bar{y}-dup_t MSS))} \big[L_0(y, W_0, W_{SS}) + \\
&+RTO + L_{k-1}(\bar{y} - y + MSS, \tfrac{MSS}{RTT}, \tfrac{W_f}{2})\big] \\
&f(y,k,\bar{y})dy + \\
&+ \int_{max(0,min(y_{to},\bar{y}-dup_t MSS))}^{max(0,\bar{y}-dup_t MSS)} \big[L_0(y, W_0, W_{SS}) + \\
&+L_{k-1}(\bar{y} - y, \tfrac{W_f}{2} - \tfrac{MSS}{RTT}, \tfrac{W_f}{2} - \tfrac{MSS}{RTT})\big] \\
&f(y,k,\bar{y})dy \\
&+ \int_{max(0,\bar{y}-dup_t MSS)}^{\bar{y}} \big[L_{k-1}(y, W_0, W_{SS}) + RTO + \\
&L_{k-1}(MSS, \tfrac{MSS}{RTT}, \tfrac{W_f}{2})\big]\, f(y,k,\bar{y})dy
\end{aligned}$$

Now knowing the expected conditional latency $L_k(\bar{y}, W_0, W_{SS})$ and the probabilities $p(k|\bar{y})$ one can calculate the expected latency by formula (1).

The validation of this result and its comparison with formulas derived in [12, 20] is given in [6]. One of the distinct features of our approach is the fact that it is capable to model accurately and easily the situations when the sender timeouts.

2.1 Modeling IW and LT modifications

Our model permits to study the impact of the initial window parameter in a straightforward manner since it is one of the input parameters. Some adaptation is however needed to model the limited transmit algorithm. This algorithm is described in detail in [1]. The basic idea is to make TCP react to two duplicate ACKs instead of three duplicate ACKs in an attempt to avoid timeout events without making TCP more aggressive. With LT algorithm the TCP sender will send two new packets upon the arrival of two duplicate ACKs and eventually it will receive a third duplicate which will trigger off fast retransmission and fast recovery phases. Let us define as dup_{LT}[2] the number of ACKs the sender running LT must receive in order to send new data

It is important to note that this modification decreases the value of y_{to}[3] and hence the performance of TCP improves. On the other hand LT does not help avoiding timeouts due to the “no enough data” effect since upon reception of two duplicate ACKs LT only transmits new

[2]The recommended value for dup_{LT} is two in [1].

[3]Under the assumption there is enough data left, upon reception of dup_{LT} duplicate ACKs the sender will end up receiving a third one and therefore avoiding timing out.

data and it does not take the decision to retransmit any packet till dup_t packets are received.

Further, if the TCP sender's congestion window does not allow to receive two duplicate ACKs ($W_f < (1 + dup_{LT})MSS/RTT$) it will timeout upon a loss event, if its congestion window allows it to receive two duplicate ACKs ($W_f = (1 + dup_{LT})MSS/RTT$) the limited transmit algorithm will take over and eventually the sender will get into fast retransmission, and finally we have the scenario when upon a loss event the sender will receive three duplicate ACKs from the beginning ($W_f > (1 + dup_{LT})MSS/RTT$).

For the sake of simplicity in our model we consider two different possibilities, where either the sender will timeout or it will get into fast retransmission directly upon the reception of dup_{LT} duplicate ACKs. For this purpose it is enough to redefine the parameter y_{to} and to use the model described in Section 2. In the case the initial window is less than three packets the value of y_{to} is given by

$$\begin{aligned} y_{to} &= y_f((1 + dup_{LT})MSS/RTT, W_0, W_{SS}) \\ &\quad 1\{W_0 < (1 + dup_{LT})MSS/RTT\} \end{aligned}$$

3 Performance Analysis and Proposed TCP modifications

In the previous section we have introduced a model to calculate the TCP latency and we have explained under which circumstances a single lost packet will make the TCP sender timeout. This model provides several interesting and useful conclusions. For instance, calculating with the theoretic model the expected conditional latency, we have observed a very interesting phenomena – the non monotonicity of the expected conditional latencies, see Figure 4.

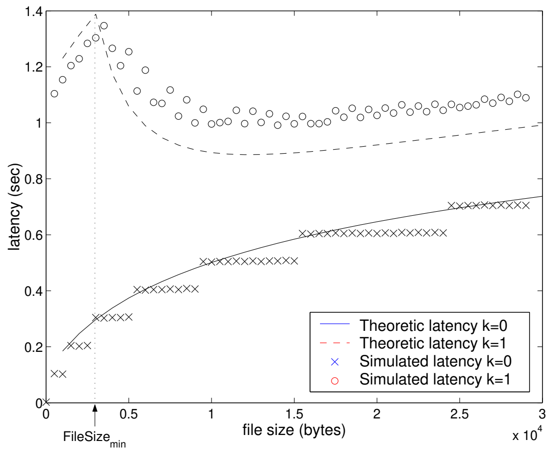

Figure 4: Expected conditional transfer time obtained by simulation and theoretic model for $W_0 = 2$, $W_{SS} = M = 30$, $MSS = 500Bytes$ and $b = 2$.

This behavior is due to the conservative value of the retransmission timer, see Section 1. In fact, there exists a threshold of the file size $FileSize_{min}$, such that if the file size is less than this threshold a loss will inevitably lead to a timeout. The value of $FileSize_{min}$ is given by the sum of y_{to} and $dup_t MSS$ (see Figure 5). In the previous example, we observe in Figure 4 that the maximum conditional latency given one packet is lost corresponds to $FileSize_{min}$. In this particular case $FileSize_{min} = 5pkts$. This value corresponds to the sum of $y_{to} = 2$ and $dup_t = 3$ (see values of y_{to} for parameters $W_0 = 2$, $b = 2$ and Slow-Start phase in Table 2).

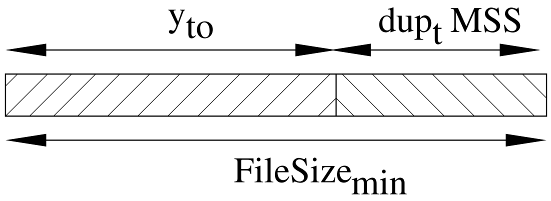

Figure 5: Graphic illustration of the value of $FileSize_{min}$

Let us analyze how much and in which way the two proposed modifications help narrowing these two intervals. The value of y_{to}, as explained in Section 2, is equal to the sum of bytes sent when upon reception of all ACKs due to previous packets the congestion window is smaller than $(1 + dup_t)MSS$. For example, let us calculate its value in a particular case. Let $W_0 = 1$, $W_{SS} = 30$ and $b = 2$ and let us consider the sender is not running the LT algorithm. The evolution of the congestion window (in bytes) at each round follows $cwnd = MSS, 2MSS, 3MSS, 5MSS...$ and since the threshold on the congestion window is given by $1 + dup_t = 4MSS$ the value of y_{to} is $4MSS$. To justify this result we have to explain that the last segment who provokes the sender to timeout corresponds to the first one of the congestion window of size $3MSS$. It is clear that if this segment is lost a timeout will happen since the sender will only receive two duplicate ACKs. If this segment is sent successfully and the next one is lost, the sender will send two new packets upon reception of the ACK corresponding to the successfully transmitted segment. The receiver will receive three out of order packets and thus it will send back three duplicate ACKs who will make the sender enter fast recovery and avoid a timeout. Of course the analysis of delAck is much more complicated in reality and there are many phenomena that have not been taken into account in this simple derivation.

In Tables 1 and 2 we present the values obtained for y_{to} for different initial windows and with/without LT algorithm for the cases of $b = 1, 2$. For the sake of simplicity we consider that the sender stays either in slow-start

	Slow-Start		Congestion Avoidance	
	No LT Alg.	With LT Alg	No LT Alg.	With LT Alg
$W_0 = 1$	3	2	6	3
$W_0 = 2$	2	1	5	2
$W_0 = 3$	1	0	3	1
$W_0 = 4$	0	0	1	0

Table 1: Numerical results for y_{to} in the case of $b = 1$

	Slow-Start		Congestion Avoidance	
	No LT Alg.	With LT Alg	No LT Alg.	With LT Alg
$W_0 = 1$	4	2	11	5
$W_0 = 2$	3	1	10	4
$W_0 = 3$	1	0	6	1
$W_0 = 4$	0	0	1	0

Table 2: Numerical results for y_{to} in the case of $b = 2$

phase[4] or congestion avoidance phase[5]. In reality mixed situations are possible but we believe these two situations are enough to get an idea of the impact of the parameter. From the values presented in Tables 1 and 2 it is clear that the value of y_{to} is not negligible. Considering typical values of $W_0 = 2$ and $b = 1$[6] we see that $FileSize_{min}$ is 4 if we use LT algorithm and 5 if we do not. If consider $b = 2$ $FileSize_{min}$ is 4 and 6 respectively. From the user's point of view, if he wants to transmit a small file and the session experiences no loss the time required to transmit it will be considered as "instantaneous" (few RTT on average) and modifications of TCP such as increasing the initial window do not provide a tremendous benefit. On contrary if the session does experience a loss, it will time out and the performance degrades notably. Thus we think that in the context of short TCP transfers the modification of dup_t and dup_{LT} could be considered.

3.1 Proposed Modification

In actual TCP implementation the value of dup_t is set to three so TCP will avoid spurious retransmissions. With LT algorithm we improve the performance of TCP without being more aggressive since no packet is retransmitted till three duplicate ACKs are received. In the context of short TCP file transfers the sender faces two different harmful scenarios. The first scenario corresponds to the situation when the size of the congestion window size does not allow the sender to receive dup_t duplicate ACKs (dup_{LT} in the case of LT) and the second scenario when any packet lost out of the last dup_t packets will make the sender timeout.

In the case of short files the occurrence of a timeout deteriorates severely the performance since the retransmission timer is typically much greater than the average RTT. Therefore it is necessary to introduce some changes to improve the transfer of short files. Thus we propose to modify TCP for short file transfers as follows:

- In order the reduce the probability of timeouts because of small congestion window size we propose to use the LT algorithm with parameter $dup_{LT} = 1$. This implies that considering a initial value for the window size of $W_0 = 2$ or greater TCP will be able always, regardless the value of b, to recover from a single loss without having to rely on a timeout.
- If the sender receives a duplicate ACK corresponding to one of the last dup_t packets, we propose that TCP retransmits immediately the lost packet. With this modification only the very last packet of the file will provoke a timeout.

In order to allow retransmissions upon reception of a single duplicate ACK it is mandatory to have a small probability of packet reordering in the network. Recent studies [23] point out 2/3's of the Internet paths had routes persisting for either days or weeks.

4 Numerical Results

In this section we use the theoretic model to evaluate the benefit obtained using the different proposals and we validate the results with NS simulations. Let us consider the simple topology a single link with capacity $C = 100Mbps$, propagation delay $d = 50ms$ and independent packet loss probability $p = 0.1$. The value for the parameters C and d are chosen in such a way so

[4] For example in the beginning of the transmission, or after timeouts where initial window is set to 1 packet and the slow-start threshold is set to the half of the current congestion size

[5] For example when upon a loss event the TCP sender enters into fast-recovery or after certain timeouts when the initial window is set to 1 packets and the Slow-Start threshold to 2

[6] Some systems use an adaptive implementation for very short flows than can be modeled well with b=1 [12].

we can model TCP behavior in terms of "rounds", where a round starts with the transmission of the first packet of the congestion window and ends when the sender receives the last acknowledgment. In order for this assumption to hold, the Bandwidth-Delay product has to be greater than the maximum window size. This assumption is the responsible of the stair case pattern for the simulated conditional latency in the case of no losses since all the file sizes that need the same number of "rounds" will require the same amount of time to be transfered.

4.1 Increasing the IW

Increasing the initial window has several advantages [3], as for example avoiding timeouts due to not enough ACKs at the beginning of the transfer, elimination of up to three $RTTs$, elimination of a delayed ACK timeout and hence it reduces in general the transmission time. When the IW is increased TCP becomes more aggressive and it has been observed that under certain conditions TCP sessions would have performed better starting with a small initial window. In the context of short TCP transfers the beneficial effect of increasing the initial window is clear from the observation of Tables 1 and 2 and it is related to the reduction of y_{to}. However it does not have a beneficial effect on reducing timeouts at the end of transmission. Observing Tables 1 and 2 for $b = 1, 2$ we see that the similar benefit can be achieved using the LT algorithm, which is less aggressive modification than IW.

4.2 Limited Transmit Algorithm

Let us study to what extent LT is beneficial. In Figures 6 7 we depict the conditional latency with and without LT algorithm. We observe that the utilization of LT algorithm does not solve completely the problem of timeouts and the value of $FileSize_{min}$ is similar in both cases, 5 using LT in Figure 6 and 4 without LT in Figure 7.

Concerning short TCP transfers, the drawback of the LT algorithm is the same as the one in the case of increasing the initial window. It reduces the value of y_{to} but it produces no benefit in respecting the last dup_t packets. However, as we will show later, LT algorithm will benefit long-live TCP sessions reducing notably timeout probability.

4.3 Reduction of dup_{LT} and dup_t

We have shown that the two proposed modifications does not quite achieve their aim in the case of short TCP transfers, since they reduce the value of y_{to} but do not succeed avoiding timeouts at the end of the transmission. Proposals which intend to improve the performance of TCP in the case of small transfered files should consider both intervals y_{to} and $dup_t MSS$ equally important. In Figure 8 we plot the results for conditional latency obtained in the case when TCP sender adopts the modifications explained

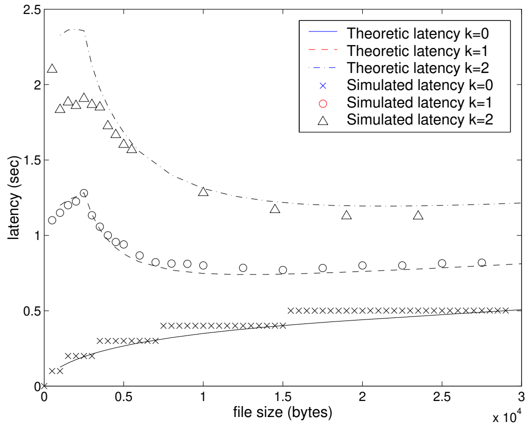

Figure 6: Expected conditional transfer time obtained by simulation and theoretic model for $W_0 = 2$, $W_{SS} = M = 30$, $MSS = 500 Bytes$, $b = 1$ and without LT algorithm.

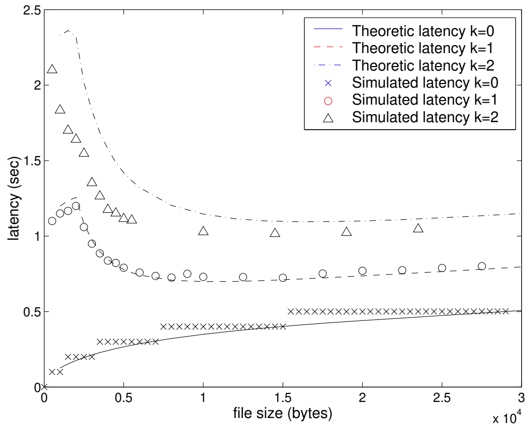

Figure 7: Expected conditional transfer time obtained by simulation and theoretic model for $W_0 = 2$, $W_{SS} = M = 30$, $MSS = 500 Bytes$, $b = 1$ and running LT algorithm.

in Section 3.1. As one can see the transfer time of a file is significantly reduced in the case when the loss event happens during the transfer.

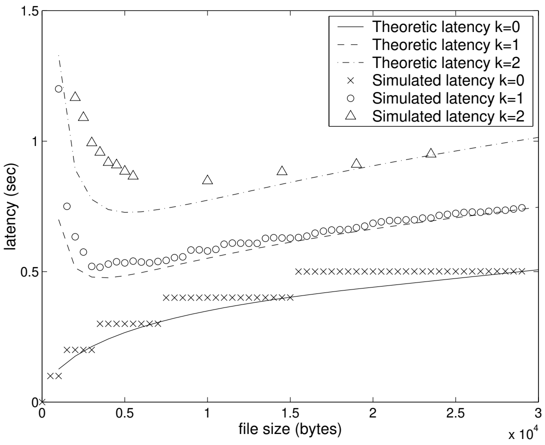

Figure 8: Expected conditional transfer time obtained by simulation and theoretic model when TCP retransmits a packet upon reception of one duplicate acknowledgment. $W_0 = 2, W_{SS} = M = 30, MSS = 500Bytes, b = 1$

In Figures 9 we depict the expected transfer time for TCP NewReno, LT algorithm and our proposal. For our proposal we differentiate between sessions greater and smaller than $10kBytes$. For sessions smaller than $10kBytes$ we consider LT with parameter $dup_{LT} = 1$ and in the case when one of the last dup_t packets is lost the sender would retransmit immediately upon reception of one duplicate ACK. For sessions greater than $10kbytes$ packets we use the standard LT algorithm. From Figure 9 one can conclude that LT reduces the latency significantly, and furthermore, our proposal reduces further the expected transfer time of short TCP transfers.

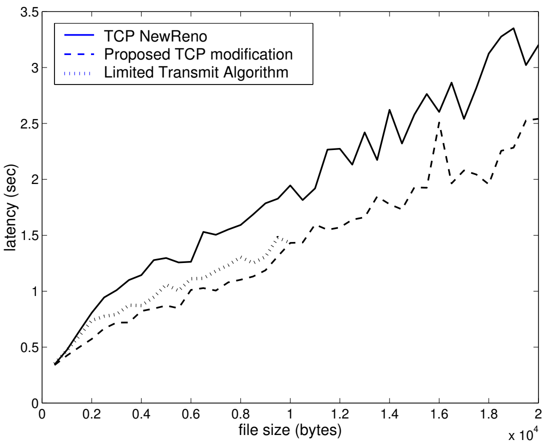

Figure 9: Expected transfer time running different versions of TCP and $W_0 = 2$, $W_{SS} = M = 30$, $MSS = 500Bytes, b = 1$

Finally we would like to note that our proposal brings a dramatic improvement of the TCP latency given the file transfer experiences a loss. For instance, one can see from Figure 10 that, given the file size experiences a loss, the LT proposal decreases the transfer time of a $4kBytes$ file by 10% and our proposal by 40%.

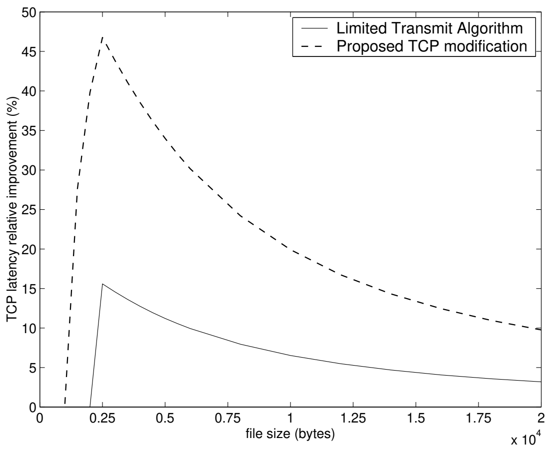

Figure 10: Relative reduction of the conditional TCP latency using LT and our TCP modification for $W_0 = 2$, $W_{SS} = M = 30$, $MSS = 500Bytes, b = 1$

5 Conclusion

In this paper we carry out the analysis of IW and LT proposals which aim to improve the performance of TCP. We demonstrate that the LT proposal brings similar benefits as the IW proposal but it is less aggressive. Then we show neither of these proposals completely achieve their goal in the context of short TCP transfers. In particular, this is because of the fact that they do not take into account the losses at the end of the file transfers which lead inevitably to timeouts. This deficiency could be corrected by reducing the number of duplicate acknowledgments which allow the packet loss detection. In particular, this modification reduces significantly the TCP latency given the file transfer experiences a packet loss. Of course, our proposal is a more aggressive modification than IW and LT, since it might result in unnecessary packet retransmissions. However, if one uses this modification only for the transfer of short files, it will not lead to a large additional network load. We recall that the total contribution of the short-live TCP transfers to the Internet traffic is small in terms of bytes [15]. For instance, in this work we have considered as short, sessions which are smaller than $10kBytes$. The effect of this proposal on the overall network performance is a subject for future research.

References

[1] M. Allman, H. Balakrishnan, S. Floyd, “RFC3042: Enhancing TCP’s Loss Recovery Using Limited Transmit”, January 2001.

[2] M. Allman, V. Paxon, W. Stevens, “RFC2581: TCP Congestion Control”, April 1999.

[3] M. Allman, S. Floyd, C. Partridge, “Increasing TCP’s Initial Window”, September 1998.

[4] E. Altman, K.E. Avrachenkov and C. Barakat, “Impact of bursty losses on TCP performance”, *Performance Evaluation*, v.42, no.2-3, pp.129-147, October 2000.

[5] E. Altman, K.E. Avrachenkov and C. Barakat, “A stochastic model of TCP/IP with stationary random losses”, *ACM SIGCOMM 2000*, Stockholm, Sweden, also in *Computer Communication Review*, v.30, no.4, pp.231-242, 2000.

[6] U.Ayesta, K.E. Avrachenkov, E. Altman, C. Barakat, P. Dube, “Modelling short TCP transfers”, INRIA Technical Report.

[7] C. Barakat, P. Thiran, G. Iannaccone, C. Diot and P. Owezarski, ”A flow-based model for Internet backbone traffic”, Technical Report EPFL/DSC/01/44, Jul. 2001. Also a shorter version appears in *ACM SIGMETRICS 2002*.

[8] J-C. Bolot, “End-to-end packet delay and loss behavior in the Internet”, *Proc. ACM Sigcomm ’93, pp. 289-298*, San Francisco, CA, Sept. 1993.

[9] O. J. Boxma, “Sojourn times in cyclic queues – the influence of the slowest server”, *Computer Performance and Reliability*, G. Iazeolla, P. J. Courtois and O. J. Boxma (Editors), Elsevier Science Publishers B.V. (North-Holland), pp. 13-24, 1988.

[10] T. Bu and D. Towsley, “Fixed point approximation for TCP behaviour in an AQM network”, in Proceedings of SIGMETRICS’2001 conference.

[11] The web site of the Cooperative Association for Internet Data Analysis (CAIDA): http://www.caida.org/.

[12] N. Cardwell, S. Savage, and T. Anderson, “Modeling TCP latency”, *IEEE INFOCOM*, pp.1742-1751, Tel Aviv, Israel, March 2000.

[13] E. Cinlar, “Introduction to stochastic processes”, Prentice-Hall, 1975.

[14] S. Dawkins, G. Montenegro, M. Kojo, V. Magret, N. Vaidya, “End-to-end Performance Implications of Links with Errors”, August 2001.

[15] S. Ben Fredj, T. Bonald, A. Proutiere, G. Regnie, J. Roberts, “Statistical Bandwidth Sharing: A Study of Congestion at Flow Level”, SIGCOMM 2001.

[16] J. Heidemann, “Performance Interactions Between P-HTTP and TCP Implementations” , ACM Computer Communication Review, 27 2, 65-73, April, 1997.

[17] A.A.Kherani, A. Kumar, ”Performance Analysis of TCP with Nonpersistent Sessions,” Workshop on Modeling of Flow and Congestion Control, INRIA, Ecole Normale Superieure, Paris, September 4-6, 2000.

[18] T.V. Lakshman and U. Madhow, “The performance of TCP/IP for networks with high bandwidth-delay products and random loss”, *IEEE/ACM Transactions on Networking*, Jun 1997.

[19] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, “The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm”, *ACM Computer Communication Review*, Jul 1997.

[20] B. Sikdar, S. Kalyanaraman and K. S. Vastola, “An Integrated Model for the Latency and Steady-State Throughput of TCP Connections”, *Performance Evaluation*, v.46, no.2-3, pp.139-154, September 2001.

[21] The web site of the National Laboratory for Applied Network Research (NLANR): http://www.nlanr.net/.

[22] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, “Modeling TCP throughput: A simple model and its empirical validation”, *ACM SIGCOMM*, Sep 1998.

[23] V. Paxson, “End-to-End Routing Behavior in the Internet”, *IEEE/ACM Transactions on Networking* 5(5) pp. 601-615.

[24] V.Paxson, M. Allman, “RFC2988: Computing TCP’s Retransmission Timer”, November 2000.

[25] K. Poduri, K. Nichols, B. Networks, “Simulation Studies of Increased Initial TCP Window Size”, September 1998.

[26] S. Savari and E. Telatar, ”The Behavior of Certain Stochastic Processes Arising in Window Protocols”, *IEEE GLOBECOM*, Dec 1999.