# TCP internal buffers optimization for fast long-distance links

A. Baiocchi*, S. Mascolo†, F. Vacirca*
*Infocom Department,
University of Roma "La Sapienza",
Via Eudossiana 18, 00184 Rome, Italy.
†DEE Politecnico di Bari,
via Orabona 4, 70125 Bari, Italy.

***Abstract*— In recent years, issues regarding the behavior of TCP in high-speed and long-distance networks have been extensively addressed in the networking research community, both because TCP is the most widespread transport protocol in the current Internet and because bandwidth-delay product continues to grow. The well known problem of TCP in high bandwidth-delay product networks is that the TCP Additive Increase Multiplicative Decrease *AIMD* probing mechanism is too slow in adapting the sending rate to the end-to-end available bandwidth. To overcome this problem, many modifications have been proposed such as FAST TCP [1], STCP [2], HSTCP [3], HTCP [4], BIC TCP [5] and CUBIC TCP [6]. However, two key parameters are often omitted in the analysis of new congestion control proposals: they are the TCP retransmission queue size at the sender and the receiver queue that handles out-of-order packets. The first buffer is used by TCP sender to save all the outstanding packets, whereas the second one is used by the receiver entity to backlog all out-of-order received packets that cannot be withdrawn by the receiver application. A wrong choice of these buffer sizes can lead to remarkable underutilization of the link capacity.**

**The goal of this work is to investigate, by using both analytical models and simulation results, optimal sizing of the retransmission and out-of-order buffers in case of modified TCP congestion control settings and to highlight differences between NewReno TCP and SACK TCP packet loss recovery schemes in terms of TCP internal buffers requirements. An important result is that the SACK option turns out to be particularly effective in reducing buffer requirements in the case of very high bandwidth-delay product links.**

## I. Background on TCP congestion control

The basic TCP congestion control is essentially made of a probing phase and a decreasing phase. The probing phase of standard TCP consists of an exponential phase (i.e. the "Slow Start" phase) and a linear increasing phase (i.e. the "Congestion Avoidance" phase). The probing phase stops when congestion is experienced in the form of timeout or reception of $DupThresh$ duplicate acknowledgments (DUPACKs)[1]. The TCP dynamic behaviour in "steady state" condition can be considered with good approximation a sequence of congestion avoidance phases followed by reception of $DupThresh$ DUPACKs. When $DupThresh$ DUPACKs are received, the TCP implements a multiplicative decrease behavior. The generalization of the classic additive increase multiplicative decrease TCP settings can be made as follows:

a) On ACK reception
$cwnd \longleftarrow cwnd + a(cwnd)$
b) When $DupThresh$ DUPACKs are received
$cwnd \longleftarrow cwnd - b \cdot cwnd$

where $a(cwnd)$ is $1/cwnd$ and $b$ is 0.5 in the case of classic TCP. The most of TCP congestion control modifications proposed for high bandwidth-delay networks can be described by modifying $a$ and $b$. Some protocols (e.g. STCP) employ constant values for $a$ and $b$, whereas other protocols, such as HSTCP and CUBIC, modify them dynamically. All these protocols can independently use NewReno or SACK recovery procedure independently of the congestion control algorithm. NewReno TCP and SACK TCP differ in the recovery phase, i.e. when the TCP recovers from packet losses. In particular, NewReno TCP recovery phase is based only on the cumulative ACK information whereas SACK TCP receiver exploits the TCP Selective Acknowledge option to advertise the sender about out-of-order received blocks. This information is employed by the sender to recover from multiple losses more efficiently than NewReno TCP.

## II. Analytical model

The goal of this section is to investigate the effect of different increment and decrement factors used by congestion control algorithms and congestion control parameters in a single connection scenario. For sake of simplicity, we consider a single bottleneck scenario in the case of constant $a$ and $b$ values such as in the case of STCP congestion control. The considered network setting is shown in Figure 1, where $B$ is the bottleneck buffer size, $C$ is the link service rate (in units of packets/s), $T_{fw}$ is the propagation delay from the TCP sender $S$ to the bottleneck buffer, $T_{fb}$ is the propagation delay from the bottleneck buffer to the TCP receiver $R$ and then back to the sender. $RTT_m$ is $T_{fb} + T_{fw}$, and $RTT$ is the sum of $RTT_m$ and the queuing delay in the bottleneck buffer (we are ignoring the packet transmission time $1/C$).

Standard TCP congestion control algorithm is made of a probing phase that increases the input rate up to fill the buffer and hit network capacity. At that point packets start to be lost and the receiver sends duplicate acknowledgments.

This work is supported by the Italian Ministry for University and Research (MIUR) under the PRIN project FAMOUS (http://www.tnt.dist.unige.it/famous/)

[1]The default value of $DupThresh$ is 3.

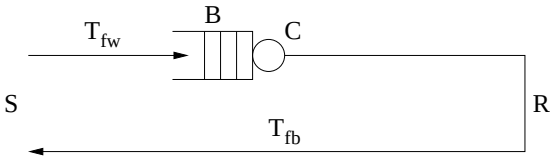



Fig. 1. Schematic of a TCP connection.

After the reception of $DupThresh$ DUPACKs, the sender infers network congestion and multiplicatively reduces the congestion window by $b$. Let $t_0$ be the time when the packet P being lost is transmitted, $t_1$ the time when the packet P is dropped at the bottleneck and $t_2$ the time when $DupThresh$ DUPACKs for the packet P are received by the sender. The congestion window at time $t_0$ is

$$W(t_0) = C \cdot RTT_m + B + 1$$

At $t_1$ the congestion window is increased to

$$W(t_1) = W(t_0 + T_{fw}) = W(t_0) + aCT_{fw}$$

During the interval from $t_1$ to $t_2$, $C \cdot (t_2 - t_1) = C \cdot T_{fb} + B + DupThresh$ ACKs are received and $(C \cdot T_{fb} + B) \cdot (1 + a)$ new packets are transmitted. At $t_2$ the congestion window is:

$$W(t_2) = W(t_1 + T_{fb} + B/C) = W(t_0) + a \cdot (CRTT_m + B)$$

After the reception of $DupThresh$ DUPACKs at time $t_2^+$, the packet P is retransmitted. TCP enters the recovery phase, the congestion window is reduced to:

$$W(t_2^+) = W(t_2) \cdot (1 - b)$$

and the sender stops until it gets acknowledgments for $b \cdot W(t_2)$ packets. Since the link service rate is $C$, the sender will stop for the interval $b \cdot W(t_2)/C$. The number of lost packets between $t_0$ and $t_2$ (excluding the packet P) is the difference between the value of the congestion window before the reception of the $DupThresh$ DUPACKS and the congestion window at $t_0$, i.e.:

$$W(t_2) - W(t_0) = a \cdot (CRTT_m + B)$$

If this number is greater than or equal to 1, the lost packets need to be retransmitted during the fast recovery phase; this phase finishes when all lost packets are eventually recovered. NewReno and SACK differ in the way the lost packets in the same window are recovered.

### A. NewReno

During the fast recovery the sender is allowed to inflate the congestion window is order not to stop the transmissions of new packets while TCP is retransmitting the lost packets. On average, every round trip time, i.e. $RTT_m + \max(0, B - bW(t_2))/C$, (i) the transmitter sends $C \cdot RTT_m + B - bW(t_2)$ new packets and (ii) it receives the ACK for a new retransmitted packet allowing the highest acknowledged packet to be increased of $1/a + 1$ packets. Let $U_n$ (for $n \geq 0$) be the number of unacknowledged packets when the $n$-th packet is retransmitted (the 0-th packet being the first lost packet P that triggered the entire recovery procedure), i.e. the difference between the highest sequence number of the transmitted packets and the highest sequence number of the acknowledged packets[2]. When P is retransmitted at $t_2$, $U$ is:

$$U_0 = W(t_2)$$

and before the retransmission of the $n$-th lost packet (or before the reception of the ACK acknowledging the $(n-1)$-th lost packet), the number of unacknowledged packets is:

$$U_n = W(t_2) + n \cdot (C \cdot RTT_m + B - bW(t_2)) - (n-1) \cdot \left(\frac{1}{a} + 1\right) \quad (1)$$

From the previous discussion, the following observation can be derived:

i) To avoid losses during the "Fast Recovery" phase the number of outstanding packets should be less than the maximum number of packets that fits into the pipe:

$$(1-b)W(t_2) \leq B + CRTT_m \Rightarrow$$

$$(1-b) \leq \frac{1}{1 + a + 1/(CRTT_m + B)} \approx \frac{1}{1+a}$$

ii) In order to provide full link utilization, the buffer depletion time $B/C$ must be greater or equal to the stop interval $b \cdot W(t_2)/C$:

$$\frac{B}{C} \geq \frac{bW(t_2)}{C} = \frac{b(CRTT_m + B + 1 + a \cdot (CRTT_m + B))}{C}$$

Assuming that the number of lost packet is small with respect to $CRTT_m + B$ (i.e. $a \ll 1$), it turns out:

$$b \leq \frac{B}{CRTT_m + B}$$

iii) The retransmission buffer and the out-of-order buffer at the receiver should be big enough to store the maximum number of unacknowledged packets. Supposing that $U_n$ is an increasing function with $n$, the maximum number of unacknowledged packets is reached before the reception of the recovery ACK. This implies that the max value of $U_n$ occurs when $n = a \cdot (CRTT_m + B)$, i.e. before the reception of the ACK acknowledging the last lost packet occurred between $t_0$ and $t_2$. Therefore it holds:

$$Q \geq U_{a(CRTT_m + B)}$$

where $Q$ is the size of the retransmission and out-of-order buffers and the right hand side can be evaluated by letting $n = a(CRTT_m + B)$ in Eq. (1).

### B. SACK TCP

After the reception of $DupThresh$ DUPACKs the SACK TCP enters in a loss recovery phase. In this phase TCP SACK is able to retransmit all the packets in a single round trip time. After the reception of $DupThresh$ DUPACKs, SACK TCP retransmits packet P and before the reception of the acknowledge for packet P, the transmitter receives in the SACK block options all the information about the

[2]We are considering TCP segment sequence number instead of bytes sequence number.

| | a=0.01, b=0.125 | | a=0.005, b=0.2 | |
|---|---|---|---|---|
| | NewReno | SACK | NewReno | SACK |
| $B = 0.05 \cdot CRTT_m$ | 251330 | 9838 | 120460 | 9445 |
| $B = 0.1 \cdot CRTT_m$ | 275310 | 10306 | 131700 | 9895 |
| $B = 0.5 \cdot CRTT_m$ | 506525 | 14054 | 239590 | 13493 |

TABLE I

TCP INTERNAL BUFFER SIZE REQUIREMENTS.

$a \cdot (CRTT_m + B)$ packet losses occurred between $t_0$ and $t_2$. With this information the sender is able to retransmit all the lost packets. In this case the number of unacknowledged packets before P is retransmitted is

$$U_0 = W(t_2)$$

and before the reception of the ACK acknowledging packet P:

$$U_1 = W(t_2) + B + CRTT_m - bW(t_2) - a(CRTT_m + B)$$

because the sender will receive $B+CRTT_m$ duplicate ACKs, but it will not be able to transmit a packet for $bW(t_2)$ ACKs because the congestion window has been closed. In this round trip time the sender is able to transmit $B+CRTT_m-bW(t_2)-a \cdot (CRTT_m + B)$ new packets, since for the first $bW(t_2)$ duplicate ACKs the congestion window does not allow to send new packets, and $a \cdot (CRTT_m + B)$ duplicate ACKs with SACK information will trigger the retransmissions of packets lost after P. After the reception of the ACK acknowledging P, the number of unacknowledged packets will decrease to $(1 - b)W(t_2)$ and TCP will exit from the "Loss Recovery" phase.

In the SACK TCP case, points i) and ii) for NewReno case hold, whereas point iii) should be modified as follows:

$$Q \geq U_1 = (CRTT_m + B)(2 - b - ab) + 1 - b$$

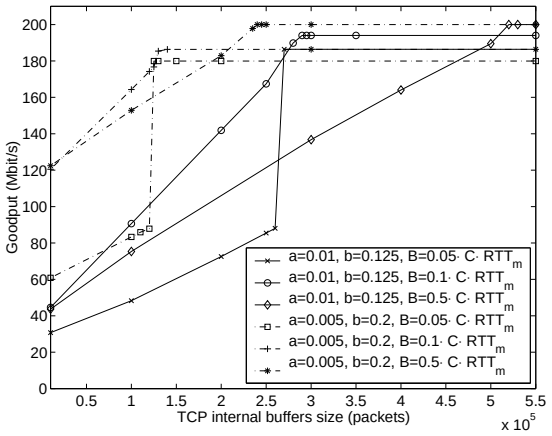
## III. RESULTS

In the previous section, we derived the sizes of the TCP internal buffers that are necessary to store the number of unacknowledged packets $U$ during the recovery phase. These values depend on: (a) the recovery procedure (NewReno or SACK style), (b) the link parameters and (c) the $a$ and $b$ settings.

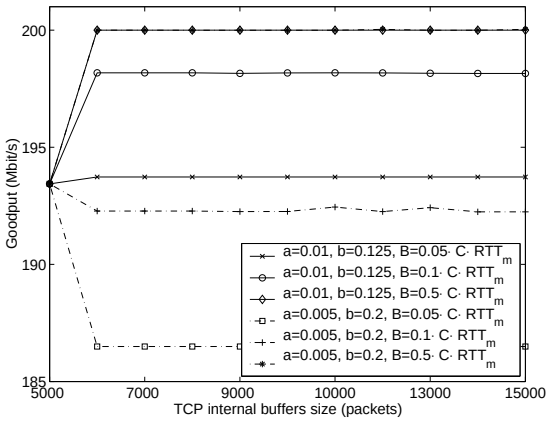
Table I reports the values of the buffer size predicted by the model, whereas Figure 2 depicts simulation results of goodput for one NewReno TCP connection (a) and one SACK TCP connection (b) with modified values of $a$ and $b$ for different values of the internal buffers' size. As far as regards TCP NewReno, the model prediction is very accurate. Moreover it is possible to notice that the goodput strongly depends on the internal buffers. As for SACK TCP, it is possible to observe that the use of the SACK option greatly reduces buffer size requirements. Moreover the impact of the buffer size has a small impact on the goodput performance.

The proposed simple analytical model nicely predicts the minimum internal buffers size that is required to achieve maximum link utilization [8][3].

[3]The achievable throughput is affected by the bottleneck buffer size as well.




(a) NewReno recovery




(b) SACK recovery

Fig. 2. Goodput with $C$=200Mbit/s, $RTT_m$=300ms and packet size of 1500 bytes in the case of NewReno (a) and SACK (b) recovery procedure.

## REFERENCES


[1] Cheng Jin, David X. Wei and Steven H. Low, "FAST TCP: motivation, architecture, algorithms, performance," Proc. of INFOCOM 2004, March 2004, Hong Kong, China.

[2] Tom Kelly, "Scalable TCP: Improving Performance in Highspeed Wide Area Networks," Proc of PFLDnet 2003, February 2003, Geneva, Switzerland.

[3] Sally Floyd, "HighSpeed TCP for Large Congestion Windows," RFC 3649, Experimental, December 2003.

[4] R.N.Shorten, D.J.Leith, "H-TCP: TCP for high-speed and long-distance networks" Proc. PFLDnet, Argonne, 2004.

[5] L. Xu, K. Harfoush, I. Rhee, "Binary Increase Congestion Control for Fast, Long Distance Networks," Proc. of INFOCOM 2004, March 2004, Hong Kong, China.

[6] I. Rhee, L. Xu, "CUBIC: A New TCP-Friendly High-Speed TCP Variant," Proc. of PFLDnet 2005, February 2005, Lyon, France.

[7] S. Mascolo, G. Racanelli, "Testing TCP Westwood+ over Transatlantic links at 10 Gigabit/Second Rate," in proc. of PFLDnet 2005, Lyon, February 2005.

[8] S. Mascolo, F. Vacirca, "Congestion Control and Sizing Router Buffers in the Internet," in proc. of Control on Decision Conference, Sevilla, Spain, December 2005.