Two Schemes to Reduce Latency in Short Lived TCP Flows

Two Schemes to Reduce Latency in Short Lived TCP Flows

D. Ciullo, M. Mellia, and M. Meo

***Abstract*—Motivated by the amount of short-lived TCP traffic due to interactive applications like web browsing, we propose two schemes to reduce the download time of TCP flows. Both schemes explicitly try to overcome the long delay suffered when TCP has to recover from the loss of the last data segment in a flow, loss for which Fast Retransmit cannot be triggered. The first scheme consists in transmitting twice the last segment, while the second one exploits the reception of the duplicate ACK caused by the delivery of the FIN segment. Both schemes allow the transmitter to recover the loss without waiting for a Retransmission Timeout expiration, thus providing quicker recovery with up to 50% gain.**

***Index Terms*—**

I. Introduction

TCP detects that a segment is lost and recovers it at the occurrence of one of these two events: i) a Retransmission Time Out expiration (RTO), or ii) the arrival of three duplicate ACKs that triggers the Fast Retransmit (FR) algorithm [1], [2]. The time to recover a loss through RTO is larger than the one of FR by a factor that is typically between 3 and 5. In case of the last segments of a flow, the RTO is the sole recovery mechanism that can be used, since the number of segments that remain to be transmitted is not large enough to generate duplicate acknowledgments at the receiver. This might result in an excessive penalty for the connection, especially for short-lived ones. Notice also that the quicker recovery of the last segment loss does not affect the congestion control of TCP, being the flow actually completed.

Several studies have shown that most of the TCP traffic is populated by short-lived flows, mainly generated by Web data transfers caused by user interactions [3], [4]. This observation still holds true after the explosion of Peer-To-Peer traffic and Web 2.0. To confirm this, left plot of Fig. 1 reports the Cumulative Distribution Function (CDF) of TCP flow length L measured in segments for both client and server flows. Data have been collected using TSTAT tool [5], and refer to one day-long measurement collected during February 2009 on a ISP BRAS link which aggregate more than 30,000 users. As can be seen, more than 95% (70%) of the client (server) flows are shorter than 10 full-size segments. For this kind of traffic, the paramount performance index is the flow completion time, or "latency", i.e., the time to complete the actual data download and this time can be significantly affected by segment losses. To give the reader the intuition of RTO impact, right plot of Fig. 1 reports the probability density function of actual flow completion time for the same dataset. The plot clearly shows some bumps, the highest one being at 3s which is the default RTO value. These bumps are due to flows that suffered a long delay due to the RTO used to recover a packet loss. Besides, we estimated by measurements the average packet drop probability during peak hour: it is typically above 4% considering downlink traffic, while it is higher than 8% considering uplink traffic.[1] These values can be used as an estimation of the last packet drop probability if we assume packet drop probability is a Bernoulli process. For the sake of completeness, more than 70% of dropped packets are recovered after a RTO expires.

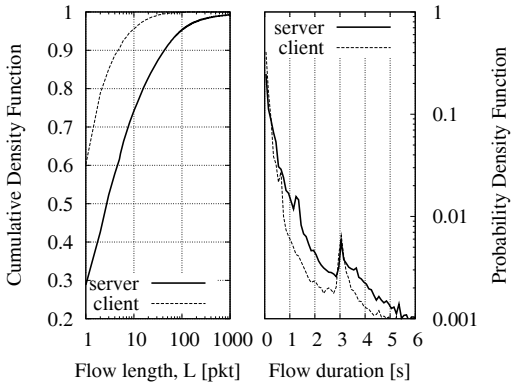

Fig. 1. TCP flow length distribution (on the left), and actual completion time (on the right).

A proposal to reduce the TCP latency by reducing the time to recover segment losses when FR cannot be triggered was made by the authors in [6]; that scheme is based on a proper data segmentation. In this letter, we, instead, focus on the impact of the last segment loss of a short-lived TCP flow, and we propose two schemes to quickly recover from the last segment loss. Both schemes outperform Classic TCP in terms of latency. The proposed schemes were previously discussed in [7].

II. Schemes Description

According to the Classic TCP, when the last segment of a TCP flow is lost, the transmitter can only wait the RTO expiration before retransmitting it: the time to recover the loss is given by the sum of one RTO and the Round Trip Time (RTT) time, see Fig. 2a. To reduce this lengthy waiting, we propose the two schemes described below.

The first scheme, named *Transmit Twice* (TT), consists in transmitting twice the last segment of a TCP flow. In this way, while introducing only a marginal increase of the

Manuscript received May 27, 2009. The associate editor coordinating the review of this letter and approving it for publication was F. Theoleyre.

The authors are with the Dipartimento di Elettronica, Politecnico di Torino, Italy (e-mail: {ciullo, mellia, meo}@tlc.polito.it).

Digital Object Identifier 10.1109/LCOMM.2009.091149

[1]Notice that these values are very much dependent on the specific network and settings.

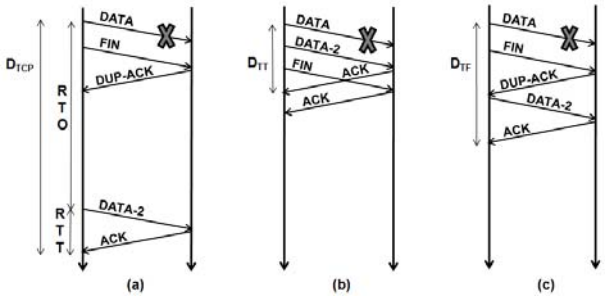

Fig. 2. Last segment loss with TCP (a), TT (b) and TF (c) schemes.

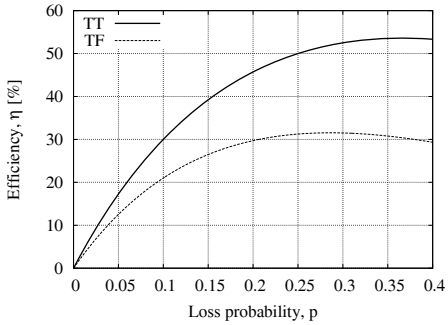

Fig. 3. Average schemes efficiency with respect to Classic TCP.

traffic injected in the network corresponding to one additional segment, the probability that the segment is successfully delivered increases: as far as one of the two last segments reaches the destination, the flow ends at the associated ACK reception. Fig. 2b shows an example of segment loss at the end of the flow: the first copy of the data segment is lost, while the second copy (DATA-2) is received correctly. The connection ends when the ACK of DATA-2 is received.

The second scheme, named *Transmit FIN* (TF), uses the duplicated ACK caused by the FIN reception to detect the possible last segment loss: at this duplicate ACK reception, the last segment is retransmitted without waiting for the RTO expiration. Fig. 2c shows an example of loss recovery under this scheme. This scheme can be implemented also by allowing the sender to immediately transmit a TCP keep-alive message after the last data segment, thus forcing the receiver to generate a duplicate ACK (as triggered by the FIN message). [2]

III. Performance Evaluation

To evaluate the performance of our schemes, we select two parameters: the last segment delivery delay, i.e., the average time needed to successfully transmit the last segment and the overhead, in terms of additional transmitted segments, introduced by the proposed schemes with respect to the Classic TCP. To compare our schemes, we adopt a model similar to [8], where performance indexes are estimated as functions of the loss probability, RTT and RTO.

1) Assumptions: To describe our model, we define the segment loss probability p as $p = p_D + (1 - p_D)p_A$ where p_D and p_A are the probabilities to lose, respectively, a DATA segment and an ACK segment. We assume that the loss rate is the same for the whole flow duration and each segment is dropped with the same probability p that is independent on the segment size.

For simplicity, we compute the retransmission timeout using the following approximation (see [2], [9] for details):

$$RTO \simeq E[RTT] + 4\sigma[RTT] \simeq 4E[RTT].$$

Moreover, we consider the transmission time to be negligible with respect to the propagation time. This also allows us to disregard the dependencies between the RTT and the segment size.

[2]Notice that, a transfer is considered successful when the source receives the ACK for the last segment, i.e., when sender knows that all transmitted data has been successfully delivered to the application. To this extent, we ignore the flow tear-down procedure.

2) Delay: We define D_{TCP}, D_{TT}, D_{TF} as, respectively, the average delay of the last segment transmission in the Classic TCP, the Transmit Twice and Transmit FIN schemes.

The average time needed to successfully send the last segment in the Classic TCP is:

$$\begin{aligned} D_{TCP} &= (1-p)RTT + \\ &\quad (1-p)\sum_{i=1}^{\infty} p^i[(2^i-1)RTO + RTT] \\ &= RTT + RTO\left(\frac{p}{1-2p}\right) \end{aligned} \quad (1)$$

Note that this holds only if $p < 1/2$. Moreover, observe that the RTO exponentially grows in case of consecutive losses, according to the Karn's algorithm. In a similar way we compute D_{TT}:

$$\begin{aligned} D_{TT} &= [(1-p)(1-p) + p(1-p) + (1-p)p]RTT + \\ &\quad (1-p)\sum_{i=2}^{\infty} p^i[(2^{i-1}-1)RTO + RTT] \\ &= (1-p^2)RTT + p^2\left(\frac{RTO}{1-2p} + RTT\right) \end{aligned} \quad (2)$$

Finally, D_{TF} is given by:

$$\begin{aligned} D_{TF} &= (1-p)RTT + p(1-p)[RTT + D_{TCP}] + \\ &\quad (1-p)\sum_{i=2}^{\infty} p^i[(2^{i-1}-1)RTO + RTT] \\ &= (1-p)RTT + p(1-p)[RTT + D_{TCP}] + \\ &\quad p^2\left(\frac{RTO}{1-2p} + RTT\right) \end{aligned} \quad (3)$$

To directly compare the two proposed schemes, we compute the efficiency η with respect to Classic TCP as:

$$\eta_i = \left(1 - \frac{D_i}{D_{TCP}}\right) \cdot 100$$

with $i = \{TT, TF\}$. As shown in Fig. 3, Transmit Twice is the best performing scheme, but also the Transmit FIN scheme outperforms Classic TCP for increasing loss probability.

Similarly, we compute the average delay given that the last segment is lost, $\widetilde{D}_i$. This allows to appreciate the improvement

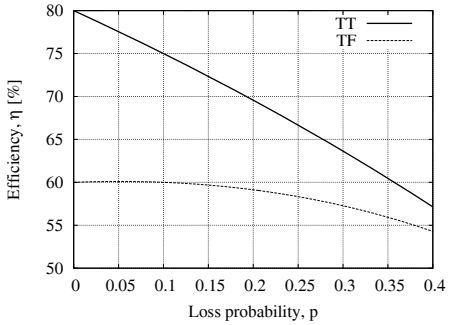

Fig. 4. Average schemes efficiency with respect to Classic TCP given that the last segment is lost.

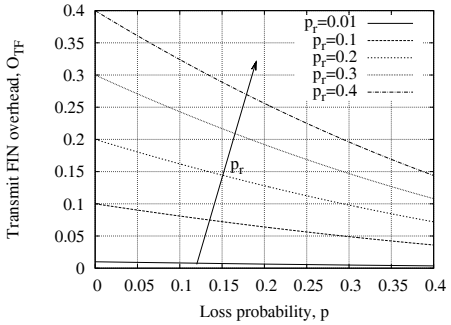

Fig. 5. TF scheme: overhead versus p.

for flows that actually suffer the last segment loss impairment.

$$\widetilde{D}_{TCP} = (1-p)\sum_{i=1}^{\infty} p^{i-1}[(2^i-1)RTO + RTT]$$
$$= RTT + RTO\left(\frac{1}{1-2p}\right) \quad (4)$$

$$\widetilde{D}_{TT} = RTT + RTO\left(\frac{p}{1-2p}\right) \quad (5)$$

$$\widetilde{D}_{TF} = RTT(2-p) + RTO\left(\frac{p}{1-2p}\right) \quad (6)$$

Fig. 4 shows that the efficiency gain η is always above 50%, and that the lower is p, the higher is the gain. This suggests that the proposed schemes can greatly improve the performance for those unlucky situations in which the last segment is lost.

3) Overhead: We define by N_{TCP}, N_{TT}, N_{TF} the average number of segments that are transmitted to successfully deliver the last segment:

$$N_{TCP} = (1-p)\sum_{i=0}^{\infty} p^i(i+1) = \frac{1}{1-p} \quad (7)$$

$$N_{TT} = N_{TCP} + 1 - p \quad (8)$$

The bytewise overhead can be derived from the overhead in segments considering the average size of the last segment, e.g., half the TCP Maximum Segment Size.

For the TF case, the value of N_{TF} depends on the probability that packets are received out-of-order rather than on p. In particular, if no reordering occurs, N_{TF} is equal to N_{TCP}; while, if the ACK of the FIN arrives earlier than the ACK of the DATA, the transmitter sends a (useless) duplicate DATA segment. Let p_r be the probability that the last segment ACK and the FIN ACK arrive out-of-order. We can compute:

$$N_{TF} = N_{TCP} + p_r(1-p)^2 \quad (9)$$

Let us compute the overheads O_{TT} and O_{TF}, in terms of the average number of additional segments that the scheme has to transmit with respect to Classic TCP, as:

$$O_i = N_i - N_{TCP} \qquad i = \{TT, TF\}$$

Then, for the Transmit Twice scheme, the additional overhead is simply given by $1-p$. For the Transmit FIN, Fig. 5 shows the additional overhead versus p, for different values of p_r. As expected, the overhead actually decreases with p, while it increases with p_r. Nonetheless, for moderate reordering probability, the additional overhead is limited.

To evaluate the additional number of segments the network has to transport, we consider the distribution of actual flow size from Fig. 1 and evaluate the additional overhead averaged over all flow size $\bar{O}_{TT}$. We analyze the Transmit Twice scheme since it performs worse than Transmit FIN in terms of overhead. We have:

$$\bar{O}_{TT} = \frac{\sum_{l=1}^{\infty}(1-p)f(L)}{\sum_{l=1}^{\infty} Lf(L)} \quad (10)$$

where $f(L)$ is the probability that the flow length is equal to L. For example, for $p = 0.2$, we obtain $\bar{O}_{TT} = 0.0214$, that is a quite negligible value.

IV. Conclusions

We have proposed two schemes to improve TCP performance in case of last segment loss: Transmit Twice, based on the duplicate transmission of the the last segment, and Transmit FIN, that uses the ACK triggered by the FIN (or additional keep-alive) segment to retransmit the last segment. Results show that substantial improvement is achieved, while marginally increasing the network load for both schemes. Transmit Twice is the best performing scheme, while Transmit FIN wastes less bandwidth due to protocol overhead.

References

[1] M. Allman, V. Paxson, and W. Stevens, "TCP congenstion control," Request for Comments 2581.
[2] V. Paxson and M. Allman, "Computing TCP's retransmission timer," Request for Comments 2988.
[3] V. Paxson and S. Floyd, "Wide-area traffic: the failure of Poisson modelling," *IEEE/ACM Trans. Networking*, vol. 3, no. 3, pp. 226–244, June 1995.
[4] S. Ebrahimi-Taghizadeh, A. Helmy, and S. Gupta, "Tcp vs. tcp: a systematic study of adverse impact of short-lived tcp flows on long-lived tcp flows," in *Proc. IEEE INFOCOM*, pp. 926–937, Miami, FL, Mar. 2005.
[5] M. Mellia, R. Lo Cigno, and F. Neri, "Measuring IP and TCP behavior on edge nodes with Tstat," *Computer Networks*, vol. 47, no. 1, pp. 1–21, Jan. 2005.
[6] M. Mellia, M. Meo, and C. Casetti, "TCP smart framing: a segmentation algorithm to reduce TCP latency," *IEEE/ACM Trans. Networking*, vol. 13, no. 2, pp. 316–329, Apr. 2005.
[7] P. Ganti, Email discussion, http://www.postel.org/pipermail/end2end-interest/2007-December/007017.html, 2007.
[8] M. Mellia, I. Stoica, and H. Zhang, "TCP model for short lived flows," *IEEE Commun. Lett.*, vol. 6, no. 2, pp. 85-87, Feb. 2002.
[9] M. Handley, S. Floyd, J. Padhye, and J. Widmer, "TCP friendly rate control TFRC: protocol specification," Request for Comments 3448.