TCP with faster recovery

Conference Paper · February 2000

DOI: 10.1109/MILCOM.2000.904968 · Source: IEEE Xplore

5 authors, including:

Claudio Casetti
Politecnico di Torino
222 PUBLICATIONS **4,963** CITATIONS
SEE PROFILE

M.Y. Sanadidi
University of California, Los Angeles
140 PUBLICATIONS **4,737** CITATIONS
SEE PROFILE

TCP WITH FASTER RECOVERY

Claudio Casetti
Mario Gerla
Scott Seongwook Lee
Saverio Mascolo
Medy Sanadidi

Computer Science Department
University of California, Los Angeles, USA

ABSTRACT

Among the problems affecting the current version of TCP are the slow recovery upon a coarse timeout expiration on long, fat pipes, and the reaction to random segment losses. Both problems are known to reduce the throughput of a connection. In our paper, we propose and evaluate the merits of a class of TCP modifications obtained through a source-based estimate of the available bandwidth by measuring the rate of received ACKs. The estimated bandwidth is used to set the slow start threshold and the congestion window after a timeout or 3 duplicate ACKs. The goal is to allow sources to recover quickly after sporadic losses over high bandwidth-delay links. It is worth noting that only a slight modification of the protocol stack at the source is needed. In our algorithms, a TCP source estimates the bandwidth available to it using an exponential averaging. Whenever an ACK is received, the bandwidth estimate is updated based on the amount of data that clears the transmission buffer following the ACK reception, divided by the current RTT estimate. After a timeout or 3 duplicate ACKs, the available bandwidth estimate is used to reset the TCP congestion window and the slow start threshold. Simulation results show that TCP with "faster recovery "exhibits higher goodput than other flavors of TCP, notably TCP Reno and TCP SACK (Selective Acknowledgement) in specific scenarios.

I. INTRODUCTION

The TCP protocol has unquestionably been one of the key contributors to the enormous success of the Internet. However, newer applications and scenarios (such as TCP over satellite or wireless links [1]) call for some revisions of the current implementation of TCP.

Due to the fundamental assumption that the network does not supply any explicit feedback to the sources, today's TCP algorithms, such as Reno [2] and Selective Acknowledgement (Sack) [3], deal with network congestion through an end-to-end control algorithm. A TCP source constantly probes for the available bandwidth by increasing the congestion window as long as no losses are detected. Initially, the increase is exponential during the Slow Start phase. This phase is intended to quickly grab the available bandwidth. When the window size reaches a slow start threshold ($ssthresh$), the increase becomes linear, thus allowing for a gentler probing of the available capacity. Clearly, a key feature is to set the threshold to a value closely related to the network capacity. The optimal value for the slow start threshold is the one that corresponds to the in-flight segments in a pipe with a capacity equal to the available bandwidth [4].

When a loss is detected either through duplicate acknowledgements, or through a coarse timeout expiration, the connection backs off by shrinking its congestion window. If the loss is indicated by duplicate ACKs, TCP Reno attempts to perform a "fast recovery" by retransmitting the lost segments and halving the congestion window. If the loss is followed by a coarse timeout expiration, the congestion window is reset to 1. In either case, after the congestion window is reset, the connection needs several round-trip times before the window-based probing is restored to near-capacity. This problem is exacerbated when random or sporadic losses occur. *Random* losses are here defined as losses not caused by congestion at the bottleneck link, as is common in the presence of wireless channels. In this case a burst of lost segments is wrongfully interpreted by a TCP source as an indication of congestion, and dealt with by shrinking the sender's window. Such action, clearly, does not alleviate the random loss condition and it merely results in reduced throughput. The degradation depends on the bandwidth-delay product.

A similar situation occurs in presence of bursty sources that may be responsible for small, *sporadic losses* due to a flurry of UDP packets shortly congesting intermediate routers. Although a smaller transmission window can help

This research was supported by Cisco, NASA (NAG2-1249), and NSF (ANI-9805436).

lowering the congestion in the short run, it will affect the source's ability to regain speed in the long run.

Random or sporadic losses (or a combination of the two) cannot be addressed using conventional TCP algorithms because they do not use any bandwidth availability information to set their congestion window. In this paper, we will propose a simple scheme to allow the TCP source to estimate the available bandwidth and use the bandwidth estimation to recover faster, thus achieving higher throughput. The paper is organized as follows: Section II outlines the estimation process, while Section III describes two "faster recovery" schemes; the performance of these schemes are compared to TCP Reno and TCP Sack using simulation in Section IV; Section V concludes the paper and lists topics for further investigation.

II. AVAILABLE BANDWIDTH ESTIMATION AT THE TCP SOURCE

To overcome the lack of information on the actual available bandwidth while the congestion window is still growing, we propose to estimate the available bandwidth by looking at the reception rate of acknowledgements. Several cases arise. Let us assume that the TCP connection has a heavy backlog and that it suddenly experiences congestion at the bottleneck. In such conditions, it likely that a timeout expires or three duplicate acknowledgments are received. In the meantime, the source has been transmitting at a rate greater than the available bandwidth. In that case, the rate of acknowledgements is proportional to the rate of data delivered to the destination, providing a good estimate of the (reduced) available bandwidth.

If a sporadic or random loss has occurred, the rate of received acknowledgements is only marginally affected, and the bandwidth estimation will show little change.

The basic idea that will be exploited in the algorithms described in the following Section is to use such estimate of available bandwidth to set the slow start threshold and to compute the congestion window.

The rate of acknowledgement is estimated through an exponential averaging. The averaging process is run upon the reception of an ACK, including duplicate ACKs (since they signal the reception of data, although out of sequence). It is detailed by the following pseudo-code:

```
if (ACK is received) {
 sample_BWE = pkt_size*8/(now - lastacktime);
 BWE = BWE*alpha + sample_BWE*(1 - alpha);
}
```

where `pkt_size` indicates the segment size in bytes, `now` indicates the current time, and `lastacktime` the time the previous ACK was received. `alpha` determines the smoothing operated by the exponential filtering. In all of our experiment, we have used `alpha` = 0.8. It should be noted that since the segment size is usually not fixed, the value `pkt_size` could be set as the average size of the last n received segments. A similar problem arises with duplicate ACKs, since they do not carry information on the size of the received segment. In this case, we propose to use the average size computed before the reception of the duplicate ACK, and to update the average size only when new data are acked.

III. FASTER RECOVERY ALGORITHMS

A. Faster Recovery TCP

The Faster Recovery TCP algorithm (FR-TCP) behaves like Reno as far as the sequence of actions following a triple duplicate ACK or a coarse timeout expiration are concerned.

The modifications we are proposing utilize the estimated bandwidth (BWE) to set the congestion window ($CWIN$) and the $ssthresh$ as follows:

- triple duplicate ACKS:
  $ssthresh = (BWE * RTT_{min})/a$
  $CWIN = ssthresh$
- coarse timeout expiration:
  $ssthresh = (BWE * RTT_{min})/a$
  $CWIN = 1$

where RTT_{min} is the smallest RTT recorded by TCP for that specific connection and a is a reduction factor. Assuming the minimum RTT excludes queueing delays, thus our scheme (ideally) converges to a situation where the transmittion rate is equal to the actual available bandwidth between source and destination.

The rationale of this strategy is quite simple. TCP Reno, after a repeated ACK or a timeout, sets the *ssthresh* to $CWIN/2$. This translates into an output rate equal to $CWIN/2 * RTT$, i.e., half the output rate when the timer expired or the third duplicate ACK was received. Instead of performing this "blind" reduction of the congestion window, we use the estimate of the available bandwidth to set the *ssthresh* equal to a fraction $1/a$ of $BWE * RTT_{min}$. In our simulations, we used $a = 2$, although we are testing other values. We have experimentally verified that, in the presence of one or few TCP connections, a good choice for a is 1, whereas in the presence of many TCP connections a better choice is $a = 2$ or greater. We believe that increasing a can mitigate the degradation of TCP performance known as "many-flows effect" [5]. This is an area of further investigation and is out of the scope of the present paper.

It should be noted that this scheme (and the one we are presenting in the following subsection) has two major advantages:

• *Avoiding unnecessarily small windows*: by computing the "target" congestion window using the bandwidth estimation, we avoid the "blind" multiplicative window decrease phase. The latter can lead to excessively small windows and cause underutilization and, in the presence of sporadic loss over wireless links, it can trigger the slow start phase without a real need again leading to underutilization;
• *Source-side implementation*: the implementation is only required at the transmitter, since the receiver, or the intermediate routers have no role in our scheme; therefore, it could be deployed by single TCP sources and it would be readily operational on the current Internet. The source model is particularly indicated in Web browsing applications, at the server side. Tipically, the Web server has large multimedia files to transfer to many clients: it makes perfect sense to invest in an improved TCP software when the returns (on the investment) are the highest (i.e., many receivers).

B. Gradual Faster Recovery TCP

As mentioned earlier, in the slow start phase, TCP grabs the bandwidth rather quickly (exponentially). In contrast, in the congestion avoidance phase, it takes a relatively long time for TCP to reach the maximum available bandwidth. If TCP experiences consecutive segment losses, the slow start threshold becomes very small, and this leads to congestion avoidance with very small congestion window. Subsequently, even though the network capacity may increase, TCP does not detect the bandwidth change and still widens the congestion window linearly. Thus, while there is a need for a bandwidth-aware window-decreasing algorithm (as in FR-TCP), a way to recognize when the output rate can be safely increased is also required. GFR-TCP handles the latter case.

The key idea of the GFR-TCP algorithm is twofold. Firstly, it minimizes the modifications to TCP modules, still achieving the performance equivalent to FR-TCP. Secondly, the algorithm is quite independent from the regular TCP congestion control scheme and can be applied to any TCP flavors, Tahoe, Reno, etc. The following is the pseudo-code for the algorithm:

```
If (CWIN > ssthresh) AND (CWIN < BWE*RTT_min)
then
  ssthresh += (BWE*RTT_min-ssthresh)/2;
```

where `CWIN` is the TCP congestion window in segments, `BWE` is the bandwidth estimation in segment/sec, `rtt` is the round-trip time that TCP keeps monitoring, `ssthresh` is the TCP slow start threshold. We are also considering the use of a slow start threshold margin to curb the aggressiveness of the algorithm and to cushion the effects of imprecisely estimated bandwidth, but its determination still requires further research.

The core of the algorithm is to monitor the bandwidth in the congestion avoidance phase and *periodically* increase the slow start threshold if the conditions allow it. To apply the algorithm *periodically*, the TCP slow timer is used. In our test, we used the standard TCP value of 500 ms.

The theoretical behavior of GFR-TCP is depicted in Figure 1. GFR-TCP deploys the same mechanisms as FR-TCP to set the congestion window and the slow start threshold on segment losses, but periodically checks the condition described in the algorithm. Whenever the condition is met, it simply recomputes the slow start threshold which causes TCP to reenter the slow start phase. Repeatedly carrying it out, TCP can reach the available bandwidth more quickly than any other TCP versions.

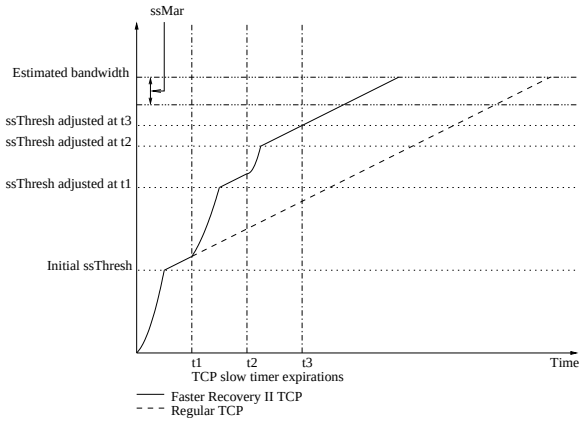

Fig. 1. GFR-TCP behavior.

IV. SIMULATION RESULTS

All simulations results presented in this paper were run using LBL network simulator, 'ns' ver.2. New simulation modules for Faster Recovery algorithms were written. Existing modules for simulations involving TCP Reno and TCP Sack were used. Each scenario, involving different capacity, RTT or number of connections, is designed as a single-bottleneck network. The intermediate node buffer is always supposed to hold a number of packets equal to the bandwidth-delay product for that scenario.

A. Losses caused by bursty traffic

Bursty UDP traffic, e.g. from IP telephony, is often the cause for TCP losses because of its uncontrolled nature. In our tests, we compared the behavior of TCP connections in the presence of an On/Off UDP source. On and Off periods are 20 seconds each, and the UDP source consumes 90%

of the bottleneck during On periods. All test lasted 200 simulated seconds. Results are reported on graphs showing the average goodput (i.e., the delivered data rate) attained by connections for each of the 4 TCP versions.

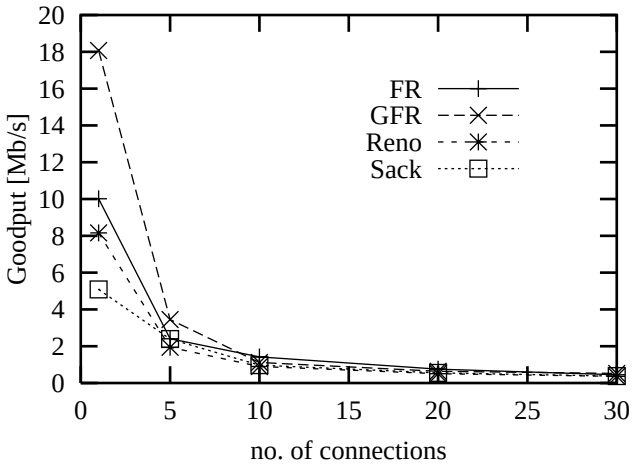

Fig. 2. Goodput as a function of number of concurrent connections

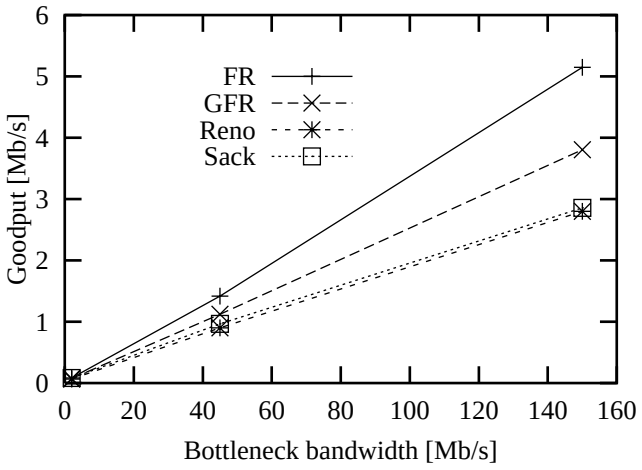

Fig. 3. Goodput as a function of bottleneck bandwidth

Figure 2 compares the scalability of each TCP version running over a 45 Mb/s bottleneck with a 500 ms RTT: results point out the versatility of GFR-TCP, which achieves a very high goodput when few other connections are present, while managing to provide an acceptable performance when more connections share the same bottleneck.

The average goodput of 10 connections is shown in Figure 3 for increasing values of the bottleneck capacity (again the RTT was 500 ms). At the highest tested capacity (150 Mb/s) FR-TCP outperforms GFR-TCP and the standard TCP versions because its aggressiveness pays off especially for long, fat pipes, where it is important to send

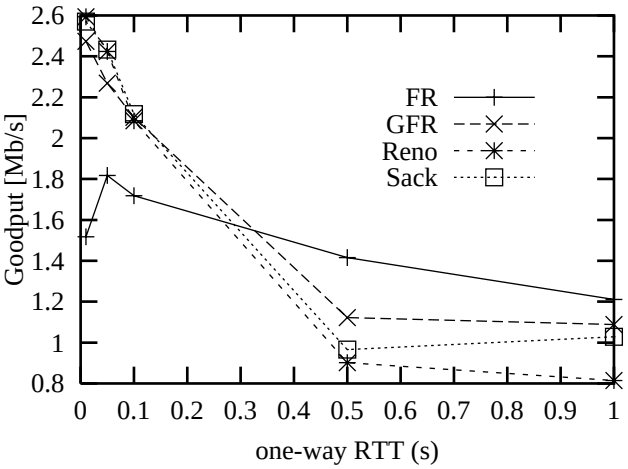

Fig. 4. Goodput as a function of one-way RTT

the largest amount of data (when feasible) in the shortest time.

Faster Recovery algorithms perform better than Reno and Sack if the RTT is higher than 0.2 seconds, as depicted in Figure 4 (10 connections, 45 Mb/s link capacity). This is particularly true for FR-TCP. If the RTT is smaller than that, Reno and Sack manage to recover fast enough so as to make up for their inherent lack of aggressiveness (when compared to Faster Recovery). Faster Recovery, on the other hand, suffers from being too aggressive for small RTTs, and the resulting poor bandwidth estimation forces it into slow start too often. While this property makes Faster Recovery particularly suitable for satellite, and long-distance connections, the shortcomings at small RTTs can easily be overcome by turning off "Faster Recovery" if the RTT is found to be smaller than a certain quantity.

B. Random Losses

In this section, we present results for lossy-link experiments. There is no background UDP, but the link is supposed to be "lossy": loss events are characterized by a burst of n dropped segments. In our simulations, n is a truncated geometric random variable with average=3. The probability of a 'loss event' is 10^{-3}.

As in previous experiments, we ran 200-second tests under several settings: Figure 5 compares results for different number of connections (1 through 30); Figure 6 for different bottleneck bandwidths (2 through 150 Mb/s) and Figure 7 for different RTTs (10ms through 1 sec). The results underscore that Faster Recovery is particularly suited for lossy links, under almost all scenarios, as predicted.

The ability to achieve a sustained throughput is further displayed in Figure 8, for a 45-Mb/s link with a 500-ms

propagation delay. The throughputs of a single (out of 10) connection for each version are plotted as a function of time. It can be seen that Reno does not recover quickly after dropping segments early on, since the ssthresh becomes so small that congestion avoidance is entered at a very early stage. Faster Recovery algorithms, on the other hand, use the bandwidth estimation process to set the ssthresh so that congestion avoidance is entered only when the sending rate is very close to the available link bandwidth.

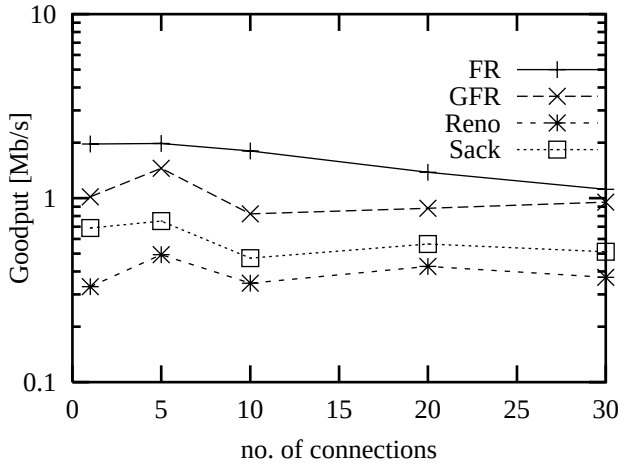

Fig. 5. Goodput as a function of number of concurrent connections

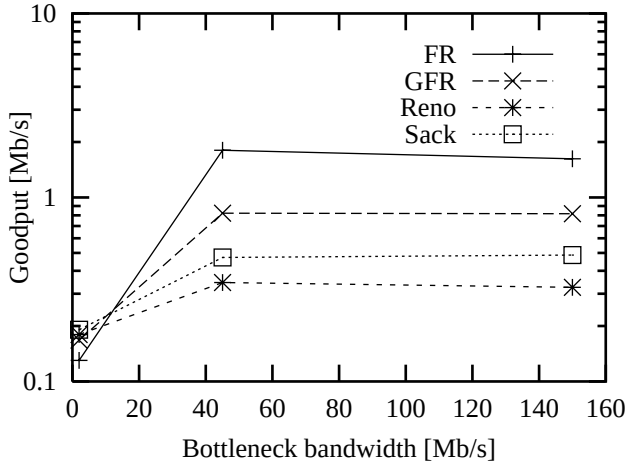

Fig. 6. Goodput as a function of bottleneck bandwidth

V. CONCLUSIONS AND FUTURE WORK

In this paper we have proposed modifications to the TCP protocol to improve its performance under random or sporadic losses conditions. Two new versions of the protocol have been described and tested through simulations, showing considerable gain in terms of goodput in various scenarios.

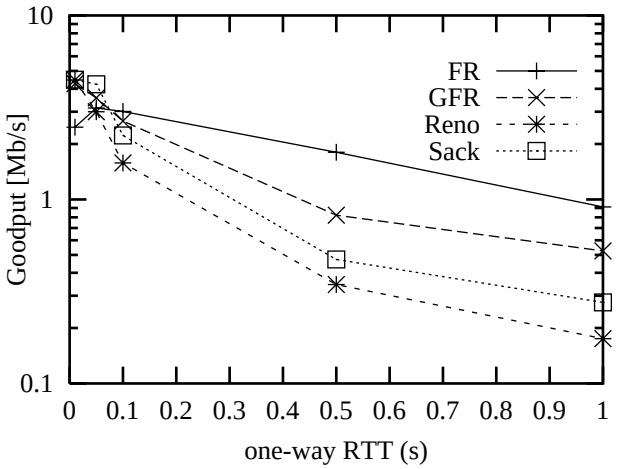

Fig. 7. Goodput as a function of one-way RTT

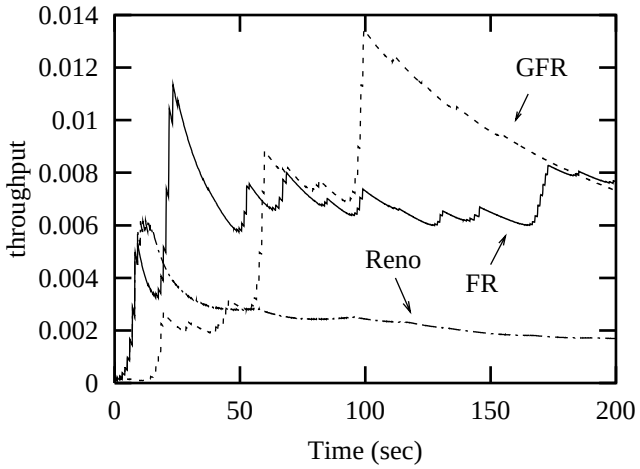

Fig. 8. Throughput vs. time for Reno, FR-TCP and GFR-TCP, on a lossy link

Further investigation is required, especially as regards the friendliness toward other connections not employing the same TCP congestion control method. Also, further refinements of the bandwidth estimation process as well as of some tuning parameters of the algorithms are under study.

REFERENCES

[1] R. Ludwig, A. Konrad, A. D. Joseph, R. H. Katz, ”Optimizing the End-to-End Performance of Reliable Flows over Wireless Links”, ACM/IEEE MOBICOM’99, Seattle, WA, Aug. 1999.

[2] W. Stevens, ”TCP Slow Start, Congestion Avoidance, Fast Retrasmit and Fast Recovery Algorithms”, RFC 2001, Feb. 1997.

[3] M. Mathis, J. Mahadavi, S. Floyd and A. Romanow, ”TCP Selective Acknoledgment Options”, RFC 2018, Oct. 1996.

[4] J. C. Hoe, ”Improving the start-up Behavior of a Congestion Control Scheme for TCP”, ACM SIGCOMM’96, Stanford, CA, USA.

[5] R. Morris, ”TCP Behavior with Many Flows”, IEEE ICNP’97, Atlanta, GA, USA, Oct. 1997.