Resolving Acknowledgment Ambiguity in non-SACK TCP

Andrei Gurtov, Sally Floyd

Abstract— **In the absence of support for Selective Acknowledgments (SACK) from its peer, TCP suffers from ambiguity regarding Duplicate Acknowledgments after a Retransmit Timeout. In particular, TCP is not able to determine if Duplicate Acknowledgments result from packets that were retransmitted unnecessarily, or from an earlier packet that was lost. Consequently, TCP can either perform unnecessary Fast Retransmits, or suppress the Fast Retransmit and have to wait unnecessarily for a Retransmit Timeout. This paper presents two heuristics that enable non-SACK TCP to correctly determine the reason for Duplicate Acknowledgments in most scenarios. Using these heuristics, a sender in a non-SACK TCP connection can perform significantly better over lossy paths.**

Keywords— **Transport protocol, retransmit timeout, NewReno.**

I. Introduction

TCP is a reliable transport protocol widely used in the Internet. TCP uses both Retransmit Timeouts and the Fast Retransmit procedure to recover lost data. Of the two, the Fast Retransmit procedure is preferable, as the additional delay and slow-start of Retransmit Timeouts significantly decrease performance [7]. The Fast Retransmit procedure is invoked when the TCP sender receives three or more duplicate acknowledgments. In a non-SACK TCP connection, duplicate acknowledgments (DUPACKs) carry no information about received segments other than confirming delivery of all segments below the cumulative acknowledgment number.

After a TCP Retransmit Timeout, duplicate acknowledgments can result either from unnecessarily retransmitted packets or from a loss of a retransmitted packet. If the sender always responded to three or more duplicate acknowledgments with a Fast Retransmit, this could result in unnecessary Fast Retransmits, and unnecessary reductions of the congestion window [2]. A solution called "bugfix" [3] disables Fast Retransmits after a Retransmit Timeout or Fast Retransmit until recovery is completed, and the possibility of three duplicate acknowledgments from unnecessarily-retransmitted packets has been removed. However, because the ambiguity is not resolved, the sender in this case might have to wait unnecessarily for a Retransmit Timeout when a retransmitted packet is lost.

The long-term solution to this problem is to use Selective Acknowledgments [11], [1] to prevent unnecessary retransmissions, and to use the Duplicate SACK extension to the SACK option [5] to help resolve duplicate acknowledgment ambiguity. However, while SACK TCP is fairly widely deployed, it is not ubiquitous. Out of 6700 web servers tested in October 2003, only 47% reported that they support SACK [12].

This paper presents two heuristics for a TCP sender in a non-SACK TCP connection to determine whether duplicate acknowledgments indicate unnecessarily retransmitted packets or a lost packet [4]. The heuristics should be implemented with NewReno TCP, but could be used also with older TCP versions, such as Reno.

The rest of the paper is organized as follows. In Section II we describe the problem of duplicate acknowledgment ambiguity in detail. The two heuristics are presented in Section III. In Section IV, failure scenarios for the two heuristics are discussed. Section V provides an evaluation of heuristics for various packet loss patterns. Section VI concludes the paper.

II. The Ambiguity Problem

When the TCP sender invokes a Fast Retransmit or a Retransmit Timeout, the TCP sender sets a variable "recover" indicating the highest sequence number transmitted so far. If the TCP sender retransmits three consecutive packets that have already been received by the data receiver, then the TCP sender will receive three duplicate acknowledgments below this recovery point. In this case, the duplicate acknowledgments are not an indication of a new instance of congestion. They are simply an indication that the sender has unnecessarily retransmitted at least three packets, as illustrated in the left graph of Figure 1.

On the other hand, if a retransmitted packet is lost, the duplicate acknowledgments indicate a hole in the receive buffer, as illustrated in the right graph of Figure 1. For a non-SACK TCP connection with a sender that implements the algorithm recommended in RFC2582 [3], the sender does not infer a packet drop from duplicate acknowledgments in these circumstances. The retransmit timer is the backup mechanism for inferring packet loss in this case. However, due to the possibility of multiple Retransmit Timeouts, it can take an excessively long time for a sender to reach the recovery point. In addition, in the absence of the timestamp option [8], the retransmit timer is kept backed-off according to Karn's rule [13]. In this case, each new Retransmit Timeout increases the back-off counter. Even when timestamps are used, the sender can be significantly delayed by Retransmit Timeouts due to lost retransmissions.

A. Gurtov is with the University of Helsinki, Finland. E-mail: gurtov@cs.helsinki.fi

S. Floyd is with the ICSI Center for Internet Research, Berkeley, USA. E-mail: floyd@icir.org

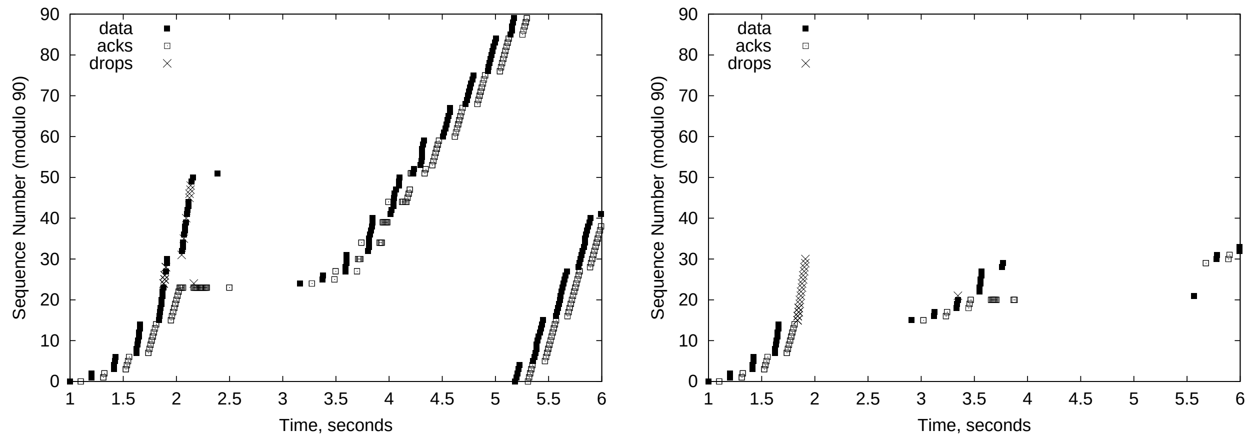


Fig. 1. NewReno TCP with unnecessarily retransmitted packets (left); a retransmitted packet that is lost (right).

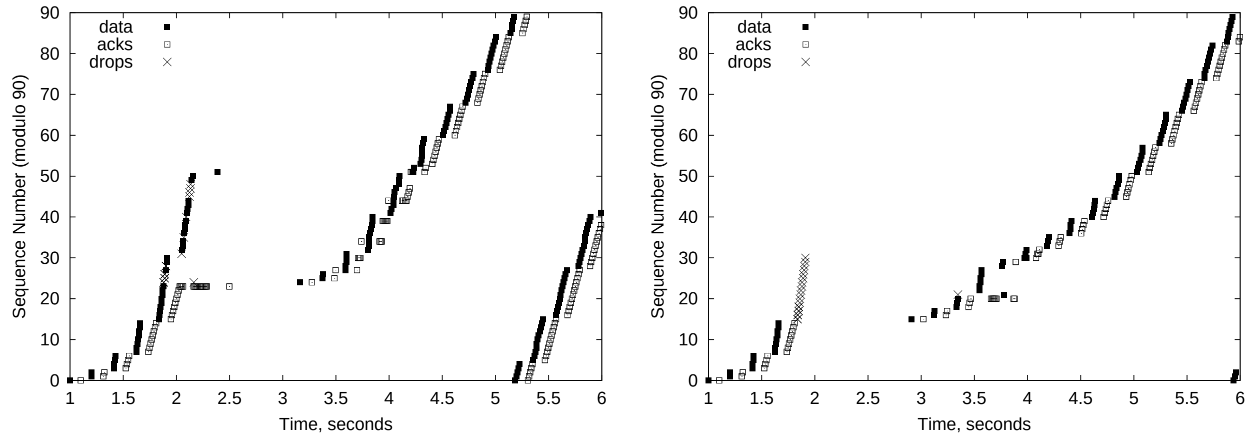


Fig. 2. The acknowledgment heuristic with unnecessarily retransmitted packets (left); a retransmitted packet that is lost (right).

III. Heuristics

The sender in a non-SACK TCP connection is still often able to detect whether the duplicate acknowledgments after a timeout are from unnecessarily retransmitted packets or a lost packet. Below we describe two heuristics that may be used to trigger Fast Retransmit below the recovery point.

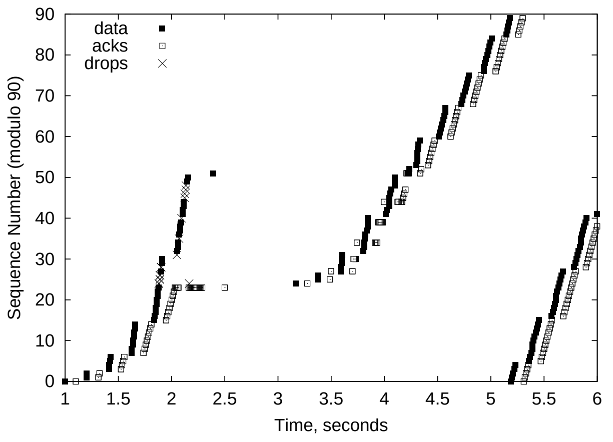
A. *Acknowledgment Heuristic*

The acknowledgment heuristic is based on an observation that if the TCP sender unnecessarily retransmits at least three adjacent packets, there will be a jump by at least four segments in a cumulative acknowledgment field. The sender will have correctly retransmitted at least one packet, to advance the cumulative acknowledgment field, and unnecessarily retransmitted at least three more to result in three duplicate acknowledgments. Following the advancement of the cumulative acknowledgment field, the sender stores the value of the previous cumulative acknowledgment as *prev_highest_ack* and stores the latest cumulative acknowledgment as *highest_ack*. Upon receiving the third duplicate acknowledgment, the sender invokes a Fast Retransmit if its congestion window is greater than one MSS (Maximum Segment Size), and the difference between *highest_ack* and *prev_highest_ack* is at most three MSS. The congestion window check serves to protect against a Fast Retransmit immediately after a Retransmit Timeout, when duplicate acknowledgments from the previous flight of packets might still be arriving. Figure 2 gives examples of applying the acknowledgment heuristic.

B. *Timestamp Heuristic*

The timestamp heuristic uses timestamps echoed by the receiver in acknowledgments. Following RFC1323 [8] or its attempted revisions [9], the receiver echoes different timestamps depending on whether there is a hole in the receive buffer. When the timestamp heuristic is used, the sender stores the timestamp of the last acknowledged segment. Upon receiving the third duplicate acknowledgment, the sender checks if the timestamp echoed in the last non-duplicate acknowledgment equals to the stored timestamp. If so and the congestion window is greater than one MSS, then the duplicate acknowledgments indicate a lost packet, and the sender invokes Fast Retransmit. Otherwise, the duplicate acknowledgments are assumed to be from unnecessary packet retransmissions, and are ignored. Figure 3 gives examples of applying the timestamp heuristic.

If the TCP connection uses timestamps, then the timestamp heuristic is to be preferred over the acknowledgment-based heuristic, because it is more accurate. Before applying either heuristic, the sender



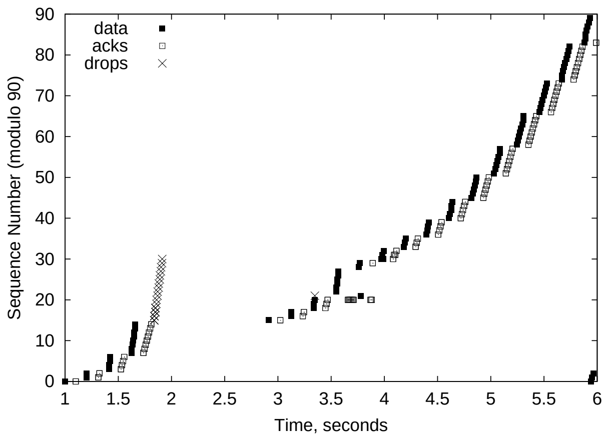


Fig. 3. The timestamp heuristic with unnecessarily retransmitted packets (left); a retransmitted packet that is lost (right).

should check that the timeout was not spurious to avoid using acknowledgments generated in response to the original and not retransmitted segments [10].

IV. Possible Failures

The acknowledgment heuristic can fail to trigger a Fast Retransmit when a packet is lost as in the case shown in Figure 4. If several acknowledgments are lost, the sender can see a jump in the cumulative acknowledgment of more than three segments. Following the acknowledgment heuristic, the sender infers that the duplicate acknowledgments are due to unnecessary retransmissions and ignores them. However, the sender might have been better off by invoking Fast Retransmit.

The acknowledgment heuristic is more likely to fail if the receiver uses delayed acknowledgments, because then a smaller number of acknowledgment losses are needed to produce a sufficient jump in the cumulative acknowledgment. If the receiver arbitrarily echoes timestamps, the timestamp heuristic can fail. However, such use of timestamps by TCP receivers appears to be rare in the Internet.

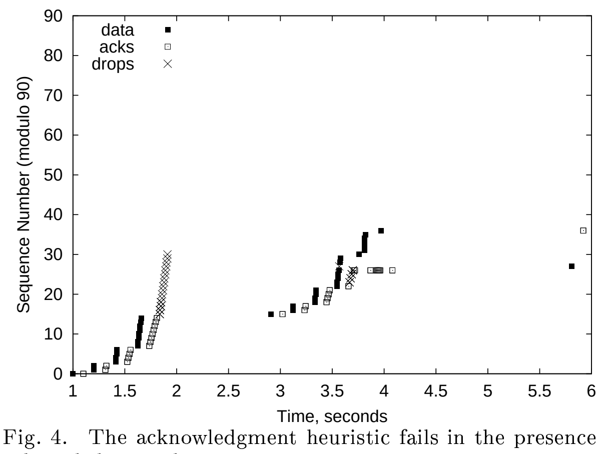


Fig. 4. The acknowledgment heuristic fails in the presence of acknowledgment losses.

V. Evaluation

In this section we evaluate NewReno TCP in the ns2 simulator using bugfix, without bugfix, with the acknowledgment heuristic, and with the timestamp heuristic for various packet loss patterns.

All variants behave the same when the loss rate is uniform and we do not show these results in detail. The difference between variants can only appear when there are Retransmit Timeouts and there is a sufficient flight of packets in the network. These conditions are not met with uniform packet losses, because at lower loss rates there are no timeouts and at high loss rates the flight size is small.

Next we experimented with bursty losses using a three-state model with sequential transition between states. The initial state is loss-free. The second state is short and serves to trigger a Retransmit Timeout. In the "loss" scenario all packets are dropped in the second state. In the "duplicate" scenario, the second state has a 50% loss rate and a fast retransmitted segment is dropped. The third state lasts two seconds and has a 10% loss rate. This state serves to cause drops of a small number of retransmissions. These simulations have been carefully designed to capture some of the behavior discussed earlier in the paper, and therefore explore scenarios where TCP's acknowledgment ambiguity is most likely to affect performance. We used a single short-lived TCP connection in simulations.

The left graph in Figure 5 shows results of simulations for the "loss" scenario. As expected, without bugfix TCP achieves higher throughput than when bugfix is enabled. With bugfix, TCP experiences unnecessary Retransmit Timeouts because of lost retransmissions. Both heuristics successfully determine that duplicate acknowledgments result from lost segments and achieve the same throughput as without bugfix, up to 300% improvement over TCP with bugfix.

The right graph in Figure 5 shows results of simulations for the "duplicate" scenario. As expected, with bugfix TCP achieves higher throughput than

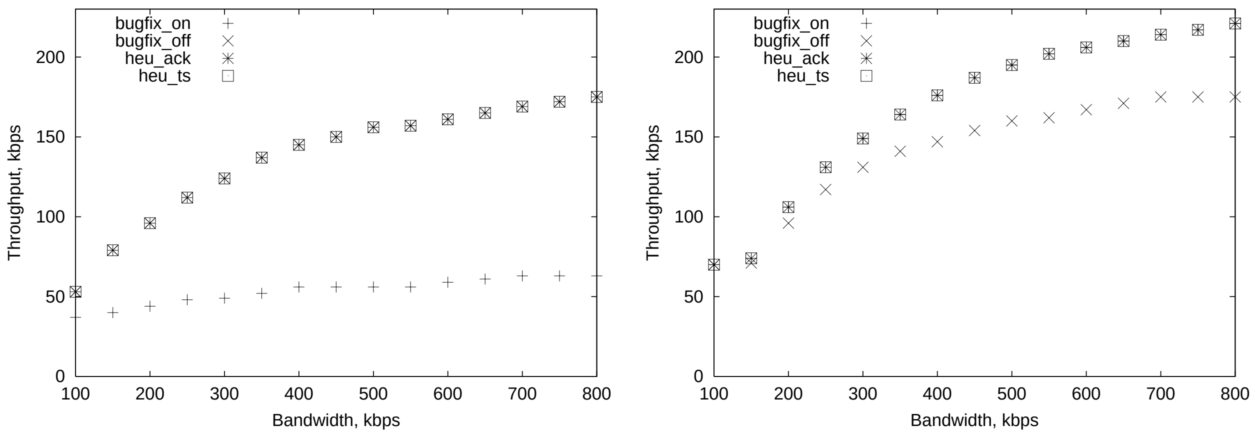


Fig. 5. Heuristics improve performance in "loss" (left) and "duplicate" (right) scenarios.

when bugfix is disabled. Without bugfix, TCP experiences unnecessary Fast Retransmits because of duplicate delivery of some segments. Both heuristics successfully determine that duplicate acknowledgments result from unnecessary retransmissions and achieve the same throughput as with bugfix, up to 30% improvement over TCP without bugfix.

In both scenarios throughput is lower than the bottleneck bandwidth, because the first Retransmit Timeout forces the connection into Congestion Avoidance with a low value of the slow start threshold.

VI. Conclusions

We presented two heuristics that enable a TCP sender in a non-SACK TCP connection to decide whether duplicate acknowledgments result from a lost packet or from unnecessarily retransmitted packets. Using these heuristics TCP can make a more informed choice whether or not to invoke Fast Retransmit.

The acknowledgment heuristic is simple to implement but fails in some scenarios in the presence of acknowledgment losses. The timestamp heuristic is more robust to packet losses but requires the use of the timestamp option and is more difficult to implement. We compared the two heuristics in detail and provided quantitative results illustrating their benefit to NewReno TCP over lossy paths. Using either of the heuristics, TCP can achieve 300% higher throughput than the standard TCP that disables Fast Retransmits in go-back-N.

Simulations for Figure 1-4 are from the "test-all-newreno" validation test in the ns2 simulator [14]. Simulation scripts for Figure 5 are publicly available [6]. We plan to add the heuristics to the Linux TCP implementation.

Acknowledgments

The authors thank Mark Allman, Reiner Ludwig, and anonymous reviewers for useful comments.

References

[1] E. Blanton, M. Allman, K. Fall, and L. Wang. A conservative selective acknowledgment (SACK)-based loss recovery algorithm for TCP. IETF RFC 3517, Apr. 2003.
[2] S. Floyd. TCP and successive fast retransmits. Technical report, Oct. 1994.
[3] S. Floyd and T. Henderson. The NewReno modification to TCP's fast recovery algorithm. IETF RFC 2582, Aug. 1999.
[4] S. Floyd, T. Henderson, and A. Gurtov. The NewReno modification to TCP's fast recovery algorithm. Work in progress, draft-ietf-tsvwg-newreno-02.txt, Nov. 2003.
[5] S. Floyd, J. Mahdavi, M. Mathis, and M. Podolsky. An extension to the selective acknowledgment (SACK) option for TCP. IETF RFC 2883, July 2000.
[6] A. Gurtov. Extensions of ns2 simulator. Available at http://www.cs.helsinki.fi/u/gurtov/ns/, Nov. 2003.
[7] V. Jacobson. Congestion avoidance and control. In *Proc. of ACM SIGCOMM'88*, Aug. 1988.
[8] V. Jacobson, R. Braden, and D. Borman. TCP extensions for high performance. IETF RFC 1323, May 1992.
[9] V. Jacobson, R. Braden, and D. Borman. TCP extensions for high performance. Work in progress, draft-jacobson-tsvwg-1323bis-00.txt, Aug. 2003.
[10] R. Ludwig and M. Meyer. The Eifel detection algorithm for TCP. IETF RFC 3522, Apr. 2003.
[11] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP selective acknowledgement options. IETF RFC 2018, Oct. 1996.
[12] A. Medina and S. Floyd. TBIT experiments, Oct. 2003.
[13] V. Paxson and M. Allman. Computing TCP's retransmission timer. IETF RFC 2988, Nov. 2000.
[14] UCB/LBNL/VINT. The ns2 network simulator, Aug. 2003. http://www.isi.edu/nsnam/ns/.