Network Working Group   S. Bhandarkar
Request for Comments: 4653   A. L. N. Reddy
Category: Experimental   Texas A&M University
                                                            M. Allman
                                                            ICIR/ICSI
                                                            E. Blanton
                                                    Purdue University
                                                          August 2006

            Improving the Robustness of TCP to Non-Congestion Events

Status of This Memo

Copyright Notice

Abstract

   This document specifies Non-Congestion Robustness (NCR) for TCP. In
   the absence of explicit congestion notification from the network, TCP
   uses loss as an indication of congestion. One of the ways TCP
   detects loss is using the arrival of three duplicate acknowledgments.
   However, this heuristic is not always correct, notably in the case
   when network paths reorder segments (for whatever reason), resulting
   in degraded performance. TCP-NCR is designed to mitigate this
   degraded performance by increasing the number of duplicate
   acknowledgments required to trigger loss recovery, based on the
   current state of the connection, in an effort to better disambiguate
   true segment loss from segment reordering. This document specifies
   the changes to TCP, as well as the costs and benefits of these
   modifications.

Table of Contents

1. Introduction

   One strength of TCP [RFC793] lies in its ability to adjust its
   sending rate according to the perceived congestion in the network
   [Jac88, RFC2581]. In the absence of explicit notification of
   congestion from the network, TCP uses segment loss as an indication
   of congestion (i.e., assuming queue overflow). TCP receivers send
   cumulative acknowledgments (ACKs) indicating the next sequence number
   expected from the sender for arriving segments [RFC793]. When
   segments arrive out of order, duplicate ACKs are generated. As
   specified in [RFC2581], a TCP sender uses the arrival of three
   duplicate ACKs as an indication of segment loss. The TCP sender
   retransmits the lost segment and reduces the load imposed on the
   network, assuming the segment loss was caused by resource contention
   within the network path. The TCP sender does not assume loss on the
   first or second duplicate ACK, but waits for three duplicate ACKs to
   account for minor packet reordering. However, the use of this
   constant threshold of duplicate ACKs has several problems that can be
   mitigated with a dynamic threshold.

   The following is an example of TCP's behavior:

     + TCP A is the data sender, and TCP B is the data receiver.

     + TCP A sends 10 segments, each consisting of a single data byte
       (i.e., transmits bytes 1-10 in segments 1-10).

+ Assume segment 3 is dropped in the network.

+ TCP B cumulatively acknowledges segments 1 and 2, making the
  cumulative ACK transmitted to the sender 3 (the next expected
  sequence number). (Note: TCP B may generate one or two ACKs,
  depending on whether delayed ACKs [RFC1122, RFC2581] are
  employed.)

+ The arrival of segments 4-10 at TCP B will each trigger the
  transmission of a cumulative ACK for sequence number 3. (Note:
  [RFC2581] recommends that delayed ACKs not be used when the ACK
  is triggered by an out-of-order segment.)

+ When TCP A receives the third duplicate ACK (or fourth ACK
  overall) for sequence number 3, TCP A will retransmit
  segment 3 and reduce the sending rate by roughly half (see
  [RFC2581] for specifics on the congestion control state
  adjustments).

Alternatively, suppose segment 3 was not dropped by the network, but
rather delayed such that segment 3 arrives at TCP B after segment 10.
The above scenario will play out in precisely the same manner
insomuch as a retransmission of segment 3 will be triggered. In
other words, TCP is not capable of disambiguating this reordering
event from a segment loss, resulting in an unnecessary retransmission
and rate reduction.

The following is the specific motivation behind making TCP robust to
reordered segments:

* A number of Internet measurement studies have shown that packet
  reordering is not a rare phenomenon [Pax97, BPS99, JIDKT03,
  GPL04]. Further, the reordering can be well beyond that required
  for fast retransmit to be falsely triggered.

* [BA02, ZKFP03] show the negative performance implications that
  packet reordering has on current TCP.

* The requirement imposed by TCP for almost in-order packet
  delivery places a constraint on the design of future technology.
  Novel routing algorithms, network components, link-layer
  retransmission mechanisms, and applications could all be looked
  at with a fresh perspective if TCP were to be more robust to
  segment reordering. For instance, high-speed packet switches
  could cause resequencing of packets if TCP were more robust.
  There has been work proposed in the literature explicitly to
  ensure that packet ordering is maintained in such switches (e.g.,
  [KM02]). Also, link-layer mechanisms that attempt to recover

      from packet corruption by retransmitting could be allowed to
      reorder packets, and thus increase the chances of local loss
      repair rather than rely on TCP to repair the loss (and,
      needlessly reduce its sending rate). Additional examples include
      multi-path routing, high-delay satellite links, and some of the
      schemes proposed for a differentiated services architecture. By
      making TCP more robust to non-congestion events, TCP-NCR may open
      the design space of the future Internet components.

   In this document, we specify a set of TCP sender modifications to
   provide Non-Congestion Robustness (NCR) to TCP. In particular, these
   changes are built on top of TCP with selective acknowledgments
   (SACKs) [RFC2018] and the SACK-based loss recovery scheme given in
   [RFC3517], since SACK is widely deployed at this point ([MAF05]
   indicates that 68% of web servers and 88% of web clients utilize SACK
   as of spring 2004).

   Note that the TCP-NCR algorithm provided in this document could be
   easily adapted to SCTP [RFC2960] since SCTP uses congestion control
   algorithms similar to TCP's (and thus has the same reordering
   robustness issues).

   As noted in several places in the remainder of this document, we
   consider TCP-NCR experimental in that more experience with the
   techniques is required before TCP-NCR should be used on a large scale
   on the Internet. We encourage implementation and experimentation
   with TCP-NCR in the hopes of gaining an understanding of its
   suitability for wide-scale deployment.

   The remainder of this document is organized as follows. Section 2
   provides a high-level description of the TCP-NCR mechanisms. In
   Section 3, we specify the TCP-NCR algorithm. Section 4 provides a
   brief overview of the benefits of TCP-NCR, while Section 5 discusses
   the drawbacks. Section 6 discusses related work. Section 7
   discusses security concerns.

1.1. Terminology

   The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
   "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
   document are to be interpreted as described in [RFC2119].

   Readers should be familiar with the TCP terminology (e.g.,
   FlightSize, Pipe) given in [RFC2581] and [RFC3517].

2. NCR Description

   As discussed above, in the face of packet reordering, three duplicate
   ACKs may not be enough to disambiguate loss from reordering. In this
   section we provide a non-normative sketch of TCP-NCR. The detailed
   algorithms for implementing Non-Congestion Robustness for TCP are
   presented in the next section.

   The general idea behind TCP-NCR is to increase the threshold used to
   trigger a fast retransmission from the current fixed value of three
   duplicate ACKs [RFC2581] to approximately a congestion window of data
   having left the network (but not less than the currently standardized
   value of three duplicate ACKs). Since cwnd represents the amount of
   data a TCP flow can transmit in one round-trip time (RTT), waiting to
   receive notice that cwnd bytes have left the network before deciding
   whether the root cause is loss or reordering imposes a delay of
   roughly one RTT on both the retransmission and the congestion control
   response. The appropriate choice for a new value of the threshold is
   essentially a trade-off between making the best decision regarding
   the cause of the duplicate ACKs and responsiveness. The choice to
   trigger a retransmission only after a cwnd's worth of data is known
   to have left the network represents roughly the largest amount of
   time a TCP can wait before the (often costly) retransmission timeout
   may be triggered. Therefore, the algorithm described in this
   document attempts to make the best decision possible at the expense
   of timeliness.

   Simply increasing the threshold before retransmitting a segment can
   make TCP brittle to packet loss or ACK loss since such loss reduces
   the number of duplicate ACKs that will arrive at the sender from the
   receiver. For instance, if the cwnd is 10 segments and one segment
   is lost, a duplicate ACK threshold of 10 will never be met because
   duplicate ACKs corresponding to at most 9 segments will arrive at the
   sender. To offset the issue of loss, we extend TCP's Limited
   Transmit [RFC3042] scheme to allow for the sending of new data during
   the period when the TCP sender is disambiguating loss and reordering.
   This new data serves to increase the likelihood that enough duplicate
   ACKs arrive at the sender to trigger loss recovery if it is
   appropriate.

   Note that TCP tightly couples reliability and congestion control:
   when a segment is declared lost, a retransmission is triggered, and a
   change to the sending rate is also made on the assumption that the
   drop is due to resource contention [RFC2581]. Therefore, simply by
   changing the retransmission trigger, the congestion control response
   is also changed. However, we lack experience on the Internet as to
   whether delaying the point that a rate reduction takes place is

appropriate for wide-scale deployment. Therefore, the Extended
Limited Transmit mechanism proposed in this document offers two
variants for experimentation.

The first Extended Limited Transmit variant, Careful Limited
Transmit, calls for the transmission of one previously unsent
segment, in response to duplicate acknowledgments, for every two
segments that are known to have left the network. This effectively
halves the sending rate, since normal TCP operation calls for the
sending of one segment for every segment that has left the network.
Further, the halving starts immediately and is not delayed until a
retransmission is triggered. In the case of packet reordering (i.e.,
not segment loss), the congestion control state is restored to its
previous state when reordering is determined.

The second variant, Aggressive Limited Transmit, calls for
transmitting one previously unsent data segment, in response to
duplicate acknowledgments, for every segment known to have left the
network. With this variant, while waiting to disambiguate the loss
from a reordering event, ACK-clocked transmission continues at
roughly the same rate as before the event started. Retransmission
and the sending rate reduction happen per [RFC2581, RFC3517], albeit
with the delayed threshold described above. Although this approach
delays legitimate rate reductions (possibly slightly and temporarily
aggravating overall congestion on the network), the scheme has the
advantage of not reducing the transmission rate in the face of
segment reordering.

Which of the two Extended Limited Transmit variants is best for use
on the Internet is an open question.

3. Algorithm

The TCP-NCR modifications make two fundamental changes to the way
[RFC3517] currently operates, as follows.

First, the trigger for retransmitting a segment is changed from three
duplicate ACKs [RFC2581, RFC3517] to indications that a congestion
window's worth of data has left the network. Second, TCP-NCR
decouples initial congestion control decisions from retransmission
decisions, in some cases delaying congestion control changes relative
to TCP's current behavior as defined in [RFC2581]. The algorithm
provides two alternatives for extending Limited Transmit. The two
variants of extended Limited Transmit are:

Careful Limited Transmit

   This variant calls for reducing the sending rate at
   approximately the same time [RFC2581] implementations reduce
   the congestion window, while at the same time withholding a
   retransmission (and the final congestion determination) for
   approximately one RTT.

Aggressive Limited Transmit

   This variant calls for maintaining the sending rate in the
   face of duplicate ACKs until TCP concludes that a segment is
   lost and needs to be retransmitted (which TCP-NCR delays by
   one RTT when compared with current loss recovery schemes).

A TCP-NCR implementation MUST use either Careful Limited Transmit or
Aggressive Limited Transmit.

A constant MUST be set, depending on which variant of extended
Limited Transmit is used, as follows:

   Careful Limited Transmit

   $LT\_F = 2/3$

   Aggressive Limited Transmit

   $LT\_F = 1/2$

This constant reflects the fraction of outstanding data (including
data sent during Extended Limited Transmit) that must be SACKed
before a retransmission is triggered. Since Aggressive Limited
Transmit sends a new segment for every segment known to have left the
network, a total of roughly cwnd segments will be sent during
Aggressive Limited Transmit, and therefore ideally a total of roughly
2*cwnd segments will be outstanding when a retransmission is
triggered. The duplicate ACK threshold is then set to $LT\_F = 1/2$ of
2*cwnd (or about 1 RTT worth of data). The factor is different for
Careful Limited Transmit because the sender only transmits one new
segment for every two segments that are SACKed and therefore will
ideally have a total of 1.5*cwnd segments outstanding when the
retransmission is to be triggered. Hence, the required threshold is
$LT\_F=2/3$ of 1.5*cwnd to delay the retransmission by roughly 1 RTT.

There are situations whereby the sender cannot transmit new data
during Extended Limited Transmit (e.g., lack of data from the
application, receiver's advertised window limit). These situations
can lead to the problems discussed in the last section when a TCP

does not employ Extended Limited Transmit and is starved for ACKs.
Therefore, TCP-NCR adapts the duplicate ACK threshold on each SACK
arrival to be as robust as possible given the actual amount of data
that has been transmitted, or roughly LT_F times the number of
outstanding segments.

The TCP-NCR modifications specified in this document lend themselves
to incremental deployment. Only the TCP implementation on the sender
side requires modification (assuming both hosts support SACK). The
changes themselves are modest. However, as will be discussed below,
availability of additional buffer space at the receiver will help
maximize the benefits of using TCP-NCR but is not strictly necessary.

The following algorithms depend on the notions provided by [RFC3517],
and we assume the reader is familiar with the terminology given in
[RFC3517]. The TCP-NCR algorithm can be adapted to alternate SACK-
based loss recovery schemes. [BR04, BSRV04] outline non-SACK-based
algorithms; however, we do not specify those algorithms in this
document and do not recommend them due to both the complexity and
security implications of having only a gross understanding of the
number of outstanding segments in the network.

A TCP connection using the Nagle algorithm [RFC896, RFC1122] MAY
employ the TCP-NCR algorithm. If a TCP implementation does implement
TCP-NCR, the implementation MUST follow the various specifications
provided in Sections 3.1 - 3.4. If the Nagle algorithm is not being
used, there is no way to accurately calculate the number of
outstanding segments in the network (and, therefore, no good way to
derive an appropriate duplicate ACK threshold) without adding state
to the TCP sender. A TCP connection that does not employ the Nagle
algorithm SHOULD NOT use TCP-NCR. We envision that NCR could be
adapted to an implementation that carefully tracks the sequence
numbers transmitted in each segment. However, we leave this as
future work.

3.1. Initialization

When entering a period of loss/reordering detection and Extended
Limited Transmit, a TCP-NCR MUST initialize several state variables.
A TCP MUST enter Extended Limited Transmit upon receiving the first
ACK with a SACK block after the reception of an ACK that (a) did not
contain SACK information and (b) did increase the connection's
cumulative ACK point. The initializations are:

(I.1) The TCP MUST save the current FlightSize.

        FlightSizePrev = FlightSize

(I.2) The TCP MUST set a variable for tracking the number of
      segments for which an ACK does not trigger a transmission
      during Careful Limited Transmit.

      Skipped = 0

      (Note: Skipped is not used during Aggressive Limited
      Transmit.)

(I.3) The TCP MUST set DupThresh (from [RFC3517]) based on the
      current FlightSize.

      DupThresh = max (LT_F * (FlightSize / SMSS),3)

      Note: We keep the lower bound of DupThresh = 3 from
      [RFC2581, RFC3517].

In addition to the above steps, the incoming ACK MUST be processed
with the E series of steps in Section 3.3.

3.2. Terminating Extended Limited Transmit and Preventing Bursts

Extended Limited Transmit MUST be terminated at the start of loss
recovery as outlined in Section 3.4.

The arrival of an ACK that advances the cumulative ACK point while in
Extended Limited Transmit, but before loss recovery is triggered,
signals that a series of duplicate ACKs was caused by reordering and
not congestion. Therefore, the receipt of an ACK that extends the
cumulative ACK point MUST terminate Extended Limited Transmit. As
described below (in (T.4)), an ACK that extends the cumulative ACK
point and *also* contains SACK information will also trigger the
beginning of a new Extended Limited Transmit phase.

Upon the termination of Extended Limited Transmit, and especially
when using the Careful variant, TCP-NCR may be in a situation where
the entire cwnd is not being utilized, and therefore TCP-NCR will be
prone to transmitting a burst of segments into the network.
Therefore, to mitigate this bursting when a TCP-NCR in the Extended
Limited Transmit phase receives an ACK that updates the cumulative
ACK point (regardless of whether the ACK contains SACK information),
the following steps MUST be taken:

   (T.1) A TCP MUST reset cwnd to:

         cwnd = min (FlightSize + SMSS,FlightSizePrev)

         This step ensures that cwnd is not grossly larger than the
         amount of data outstanding, a situation that would cause a
         line rate burst.

   (T.2) A TCP MUST set ssthresh to:

         ssthresh = FlightSizePrev

         This step provides TCP-NCR with a sense of "history". If step
         (T.1) reduces cwnd below FlightSizePrev, this step ensures that
         TCP-NCR will slow start back to the operating point in effect
         before Extended Limited Transmit.

   (T.3) A TCP is now permitted to transmit previously unsent data as
         allowed by cwnd, FlightSize, application data availability, and
         the receiver's advertised window.

   (T.4) When an incoming ACK extends the cumulative ACK point and also
         contains SACK information, the initializations in steps (I.2)
         and (I.3) from Section 3.1 MUST be taken (but step (I.1) MUST
         NOT be executed) to re-start Extended Limited Transmit. In
         addition, the series of steps in Section 3.3 (the "E" steps)
         MUST be taken.

3.3. Extended Limited Transmit

   On each ACK containing SACK information that arrives after TCP-NCR
   has entered the Extended Limited Transmit phase (as outlined in
   Section 3.1) and before Extended Limited Transmit terminates, the
   sender MUST use the following procedure.

   (E.1) The SetPipe () procedure from [RFC3517] MUST be used to set
         the "pipe" variable (which represents the number of bytes
         still considered "in the network"). Note: the current value
         of DupThresh MUST be used by SetPipe () to produce an accurate
         assessment of the amount of data still considered in the
         network.

   (E.2) If the comparison in equation (1), below, holds and there are
         SMSS bytes of previously unsent data available for
         transmission, then the sender MUST transmit one segment of SMSS
         bytes.

            (pipe + Skipped) <= (FlightSizePrev - SMSS)   (1)

        If the comparison in equation (1) does not hold or no new data
        can be transmitted (due to lack of data from the application
        or the advertised window limit), skip to step (E.6).

   (E.3) Pipe MUST be incremented by SMSS bytes.

   (E.4) If using Careful Limited Transmit, Skipped MUST be incremented
         by SMSS bytes to ensure that the next SMSS bytes of SACKed data
         processed does not trigger a Limited Transmit transmission
         (since the goal of Careful Limited Transmit is to send upon
         receipt of every second duplicate ACK).

   (E.5) A TCP MUST return to step (E.2) to ensure that as many bytes
         as are appropriate are transmitted. This provides robustness
         to ACK loss that can be (largely) compensated for using SACK
         information.

   (E.6) DupThresh MUST be reset via:

            DupThresh = max (LT_F * (FlightSize / SMSS),3)

         where FlightSize is the total number of bytes that have not
         been cumulatively acknowledged (which is different from
         "pipe").

3.4. Entering Loss Recovery

   When a segment is deemed lost via the algorithms in [RFC3517],
   Extended Limited Transmit MUST be terminated, leaving the algorithms
   in [RFC3517] to govern TCP's behavior. One slight change to
   [RFC3517] MUST be made, however. In Section 5, step (2) of [RFC3517]
   MUST be changed to:

       (2) ssthresh = cwnd = (FlightSizePrev / 2)

   This ensures that the congestion control modifications are made with
   respect to the amount of data in the network before FlightSize was
   increased by Extended Limited Transmit.

   Note: Once the algorithm in [RFC3517] takes over from Extended
   Limited Transmit, the DupThresh value MUST be held constant until the
   loss recovery phase is terminated.

4. Advantages

   The major advantages of TCP-NCR are twofold. As discussed in Section
   1, TCP-NCR will open up the design space for network applications and
   components that are currently constrained by TCP's lack of robustness
   to packet reordering. The second advantage is in terms of an
   increase in TCP performance.

   [BR04] presents ns-2 [NS-2] simulations of a pre-cursor to the TCP-
   NCR algorithm specified in this document, called TCP-DCR (Delayed
   Congestion Response). The paper shows that TCP-DCR aids performance
   in comparison to unmodified TCP in the presence of packet reordering.
   In addition, the extended version of [BR04] presents results based on
   emulations involving Linux (kernel 2.4.24). These results show that
   the performance of TCP-DCR is similar to Linux's native
   implementation that seeks to "undo" wrong decisions according to
   duplicate-SACK (DSACK) [RFC2883] feedback (similar to the schemes
   outlined in [ZKFP03]), when packets are reordered by less than one
   RTT. The advantage of using TCP-DCR over the DSACK-based scheme is
   that the DSACK-based scheme tries to estimate the exact amount of
   reordering in the network using fairly complex algorithms, whereas
   TCP-DCR achieves similar results with less complicated modifications.

   In addition, [BR04,BSRV04] illustrate the ability of TCP-DCR to allow
   for the improvement of other parts of the system. For example, these
   papers show that increasing TCP's robustness to packet reordering
   allows a novel wireless ARQ mechanism to be added at the link-layer.
   The added robustness of the link-layer to channel errors, in turn,
   increases TCP performance by not requiring TCP to retransmit packets
   that were dropped due to corruption (and thus also prevents TCP from
   needlessly reducing the sending rate when retransmitting these
   segments).

5. Disadvantages

   Although all the changes outlined above are implemented in the
   sender, the receiver also potentially has a part to play. In
   particular, TCP-NCR increases the receiver's buffering requirement by
   up to an extra cwnd -- in the case of the TCP sender using Aggressive
   Limited Transmit and actual loss occurring in the network.
   Therefore, to maximize the benefits from TCP-NCR, receivers should
   advertise a large window to absorb the extra out-of-order traffic.
   In the case that the additional buffer requirements are not met, the
   use of the above algorithm takes into account the reduced advertised
   window -- with a corresponding loss in robustness to packet
   reordering.

In addition, using TCP-NCR could delay the delivery of data to the
application by up to one RTT because the fast retransmission point is
delayed by roughly one RTT in TCP-NCR. Applications that are
sensitive to such delays should turn off the TCP-NCR option. For
instance, a socket option could be introduced to allow applications
to control whether NCR would be used for a particular connection.

Finally, the use of TCP-NCR makes the recovery from congestion events
sluggish in comparison to the standard reaction in [RFC2581]. [BR04,
BSRV04] show (via simulation) that the delay in congestion response
has minimal impact on the connection itself and the traffic sharing a
bottleneck. [BBFS01] also indicates (again, via simulation) that
"slowly responsive" congestion control may be safe for deployment in
the Internet. These studies suggest that schemes that slightly delay
congestion control decisions may be reasonable; however, further
experimentation on the Internet is required to verify these results.

6. Related Work

Over the past few years, several solutions have been proposed to
improve the performance of TCP in the face of segment reordering.
These schemes generally fall into one of two categories (with some
overlap): mechanisms that try to prevent spurious retransmits from
happening and mechanisms that try to detect spurious retransmits and
"undo" the needless congestion control state changes that have been
taken.

[BA02,ZKFP03] attempt to prevent segment reordering from triggering
spurious retransmits by using various algorithms to approximate the
duplicate ACK threshold required to disambiguate loss and reordering
over a given network path at a given time. TCP-NCR similarly tries
to prevent spurious retransmits. However, TCP-NCR takes a simplified
approach compared to those in [BA02, ZKFP03], in that TCP-NCR simply
delays retransmission by an amount based on the current cwnd (in
comparison to standard TCP), while the other schemes use relatively
complex algorithms in an attempt to derive a more precise value for
DupThresh that depends on the current patterns of packet reordering.
While TCP-NCR offers simplicity, the other schemes may offer more
precision such that applications would not be forced to wait as long
for their retransmissions. Future work could be undertaken to
achieve robustness without needless delay.

On the other hand, several schemes have been developed to detect and
mitigate needless retransmissions after the fact. [RFC3522, RFC3708,
BA02, RFC4015, RFC4138] present algorithms to detect spurious
retransmits and mitigate the changes these events made to the
congestion control state. TCP-NCR could be used in conjunction with
these algorithms, with TCP-NCR attempting to prevent spurious

retransmits and some other scheme kicking in if the prevention
failed. In addition, note that TCP-NCR is concentrated on preventing
spurious fast retransmits; some of the above algorithms also attempt
to detect and mitigate spurious timeout-based retransmits.

7. Security Considerations

General attacks against the congestion control of TCP are described
in [RFC2581]. SACK-based loss recovery for TCP [RFC3517] mitigates
some of the duplicate ACK attacks against TCP's congestion control.
This document builds upon that work, and the Extended Limited
Transmit algorithms specified in this document have been designed to
thwart the ACK division problems that are described in [RFC3465].

8. Acknowledgments

Feedback from Lars Eggert, Ted Faber, Wesley Eddy, Gorry Fairhurst,
Sally Floyd, Sara Landstrom, Nauzad Sadry, Pasi Sarolahti, Joe Touch,
Nitin Vaidya, and the TCPM working group have contributed
significantly to this document. Our thanks to all!

9. References

9.1. Normative References

   [RFC793]   Postel, J., "Transmission Control Protocol", STD 7, RFC
              793, September 1981.

   [RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP
              Selective Acknowledgement Options", RFC 2018, October 1996.

   [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
              Requirement Levels", BCP 14, RFC 2119, March 1997.

   [RFC2581] Allman, M., Paxson, V., and W. Stevens, "TCP Congestion
              Control", RFC 2581, April 1999.

   [RFC3042] Allman, M., Balakrishnan, H., and S. Floyd, "Enhancing
              TCP's Loss Recovery Using Limited Transmit", RFC 3042,
              January 2001.

   [RFC3517] Blanton, E., Allman, M., Fall, K., and L. Wang, "A
              Conservative Selective Acknowledgment (SACK)-based Loss
              Recovery Algorithm for TCP", RFC 3517, April 2003.

9.2. Informative References

   [BA02]   E. Blanton and M. Allman, "On Making TCP More Robust to
               Packet Reordering," ACM Computer Communication Review,
               January 2002.

   [BBFS01]   D. Bansal, H. Balakrishnan, S. Floyd and S. Shenker,
               "Dynamic Behavior of Slowly Responsive Congestion Control
               Algorithms", Proceedings of ACM SIGCOMM, Sep. 2001.

   [BPS99]   J. Bennett, C. Partridge, and N. Shectman, "Packet
               reordering is not pathological network behavior," IEEE/ACM
               Transactions on Networking, December 1999.

   [BR04]   Sumitha Bhandarkar and A. L. Narasimha Reddy, "TCP-DCR:
               Making TCP Robust to Non-Congestion Events", In the
               Proceedings of Networking 2004 conference, May 2004.
               Extended version available as tech report TAMU-ECE-2003-04.

   [BSRV04]   Sumitha Bhandarkar, Nauzad Sadry, A. L. Narasimha Reddy and
               Nitin Vaidya, "TCP-DCR: A Novel Protocol for Tolerating
               Wireless Channel Errors", to appear in IEEE Transactions on
               Mobile Computing.

   [GPL04]   Ladan Gharai, Colin Perkins and Tom Lehman, "Packet
               Reordering, High Speed Networks and Transport Protocol
               Performance", ICCCN 2004, October 2004.

   [Jac88]   V. Jacobson, "Congestion Avoidance and Control", Computer
               Communication Review, vol. 18, no. 4, pp. 314-329, Aug.
               1988. ftp://ftp.ee.lbl.gov/papers/congavoid.ps.Z.

   [JIDKT03]   S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D.
               Towsley, "Measurement and Classification of Out-of-Sequence
               Packets in a Tier-1 IP Backbone," Proceedings of IEEE
               INFOCOM, 2003.

   [KM02]   I. Keslassy and N. McKeown, "Maintaining packet order in
               twostage switches," Proceedings of the IEEE Infocom, June
               2002

   [MAF05]   A. Medina, M. Allman, S. Floyd. Measuring the Evolution of
               Transport Protocols in the Internet. ACM Computer
               Communication Review, 35(2), April 2005.

   [NS-2]   ns-2 Network Simulator. http://www.isi.edu/nsnam/

   [Pax97]   V. Paxson, "End-to-End Internet Packet Dynamics,"
              Proceedings of ACM SIGCOMM, September 1997.

   [RFC896]   Nagle, J., "Congestion control in IP/TCP internetworks",
              RFC 896, January 1984.

   [RFC1122] Braden, R., "Requirements for Internet Hosts -
              Communication Layers", STD 3, RFC 1122, October 1989.

   [RFC2883] Floyd, S., Mahdavi, J., Mathis, M., and M. Podolsky, "An
              Extension to the Selective Acknowledgement (SACK) Option
              for TCP", RFC 2883, July 2000.

   [RFC2960] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H.
              Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, V.
              Paxson. Stream Control Transmission Protocol. October
              2000.

   [RFC3465] Allman, M., "TCP Congestion Control with Appropriate Byte
              Counting (ABC)", RFC 3465, February 2003.

   [RFC3522] Ludwig, R. and M. Meyer, "The Eifel Detection Algorithm for
              TCP", RFC 3522, April 2003.

   [RFC3708] Blanton, E. and M. Allman, "Using TCP Duplicate Selective
              Acknowledgement (DSACKs) and Stream Control Transmission
              Protocol (SCTP) Duplicate Transmission Sequence Numbers
              (TSNs) to Detect Spurious Retransmissions", RFC 3708,
              February 2004.

   [RFC4015] Ludwig, R. and A. Gurtov, "The Eifel Response Algorithm for
              TCP", RFC 4015, February 2005.

   [RFC4138] Sarolahti, P. and M. Kojo, "Forward RTO-Recovery (F-RTO):
              An Algorithm for Detecting Spurious Retransmission Timeouts
              with TCP and the Stream Control Transmission Protocol
              (SCTP)", RFC 4138, August 2005.

   [ZKFP03]   M. Zhang, B. Karp, S. Floyd, L. Peterson, "RR-TCP: A
              Reordering-Robust TCP with DSACK", in Proceedings of the
              Eleventh IEEE International Conference on Networking
              Protocols (ICNP 2003), Atlanta, GA, November, 2003.

Authors' Addresses

   Sumitha Bhandarkar
   Dept. of Elec. Engg.
   214 ZACH
   College Station, TX 77843-3128

   Phone: (512) 468-8078
   EMail: sumitha@tamu.edu
   URL: http://students.cs.tamu.edu/sumitha/

   A. L. Narasimha Reddy
   Professor
   Dept. of Elec. Engg.
   315C WERC
   College Station, TX 77843-3128

   Phone: (979) 845-7598
   EMail: reddy@ee.tamu.edu
   URL: http://ee.tamu.edu/~reddy/

   Mark Allman
   ICSI Center for Internet Research
   1947 Center Street, Suite 600
   Berkeley, CA 94704-1198

   Phone: (440) 235-1792
   EMail: mallman@icir.org
   URL: http://www.icir.org/mallman/

   Ethan Blanton
   Purdue University Computer Science
   305 North University Street
   West Lafayette, IN 47907

   EMail: eblanton@cs.purdue.edu

Full Copyright Statement

Intellectual Property

Acknowledgement