Network Working Group
Request for Comments: 2988
Category: Standards Track

V. Paxson
ACIRI
M. Allman
NASA GRC/BBN

November 2000

Computing TCP's Retransmission Timer

Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice

Abstract

This document defines the standard algorithm that Transmission Control Protocol (TCP) senders are required to use to compute and manage their retransmission timer. It expands on the discussion in section 4.2.3.1 of RFC 1122 and upgrades the requirement of supporting the algorithm from a SHOULD to a MUST.

1 Introduction

The Transmission Control Protocol (TCP) [Pos81] uses a retransmission timer to ensure data delivery in the absence of any feedback from the remote data receiver. The duration of this timer is referred to as RTO (retransmission timeout). RFC 1122 [Bra89] specifies that the RTO should be calculated as outlined in [Jac88].

This document codifies the algorithm for setting the RTO. In addition, this document expands on the discussion in section 4.2.3.1 of RFC 1122 and upgrades the requirement of supporting the algorithm from a SHOULD to a MUST. RFC 2581 [APS99] outlines the algorithm TCP uses to begin sending after the RTO expires and a retransmission is sent. This document does not alter the behavior outlined in RFC 2581 [APS99].

In some situations it may be beneficial for a TCP sender to be more conservative than the algorithms detailed in this document allow. However, a TCP MUST NOT be more aggressive than the following algorithms allow.

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 [Bra97].

2 The Basic Algorithm

To compute the current RTO, a TCP sender maintains two state variables, SRTT (smoothed round-trip time) and RTTVAR (round-trip time variation). In addition, we assume a clock granularity of G seconds.

The rules governing the computation of SRTT, RTTVAR, and RTO are as follows:

(2.1) Until a round-trip time (RTT) measurement has been made for a segment sent between the sender and receiver, the sender SHOULD set RTO <- 3 seconds (per RFC 1122 [Bra89]), though the "backing off" on repeated retransmission discussed in (5.5) still applies.

Note that some implementations may use a "heartbeat" timer that in fact yield a value between 2.5 seconds and 3 seconds. Accordingly, a lower bound of 2.5 seconds is also acceptable, providing that the timer will never expire faster than 2.5 seconds. Implementations using a heartbeat timer with a granularity of G SHOULD not set the timer below 2.5 + G seconds.

(2.2) When the first RTT measurement R is made, the host MUST set

```
SRTT <- R
RTTVAR <- R/2
RTO <- SRTT + max (G, K*RTTVAR)
```

where K = 4.

(2.3) When a subsequent RTT measurement R' is made, a host MUST set

```
RTTVAR <- (1 - beta) * RTTVAR + beta * |SRTT - R'|
SRTT <- (1 - alpha) * SRTT + alpha * R'
```

The value of SRTT used in the update to RTTVAR is its value before updating SRTT itself using the second assignment. That is, updating RTTVAR and SRTT MUST be computed in the above order.

The above SHOULD be computed using alpha=1/8 and beta=1/4 (as suggested in [JK88]).

After the computation, a host MUST update
RTO <- SRTT + max (G, K*RTTVAR)

(2.4) Whenever RTO is computed, if it is less than 1 second then the RTO SHOULD be rounded up to 1 second.

Traditionally, TCP implementations use coarse grain clocks to measure the RTT and trigger the RTO, which imposes a large minimum value on the RTO. Research suggests that a large minimum RTO is needed to keep TCP conservative and avoid spurious retransmissions [AP99]. Therefore, this specification requires a large minimum RTO as a conservative approach, while at the same time acknowledging that at some future point, research may show that a smaller minimum RTO is acceptable or superior.

(2.5) A maximum value MAY be placed on RTO provided it is at least 60 seconds.

3 Taking RTT Samples

TCP MUST use Karn's algorithm [KP87] for taking RTT samples. That is, RTT samples MUST NOT be made using segments that were retransmitted (and thus for which it is ambiguous whether the reply was for the first instance of the packet or a later instance). The only case when TCP can safely take RTT samples from retransmitted segments is when the TCP timestamp option [JBB92] is employed, since the timestamp option removes the ambiguity regarding which instance of the data segment triggered the acknowledgment.

Traditionally, TCP implementations have taken one RTT measurement at a time (typically once per RTT). However, when using the timestamp option, each ACK can be used as an RTT sample. RFC 1323 [JBB92] suggests that TCP connections utilizing large congestion windows should take many RTT samples per window of data to avoid aliasing effects in the estimated RTT. A TCP implementation MUST take at least one RTT measurement per RTT (unless that is not possible per Karn's algorithm).

For fairly modest congestion window sizes research suggests that timing each segment does not lead to a better RTT estimator [AP99]. Additionally, when multiple samples are taken per RTT the alpha and beta defined in section 2 may keep an inadequate RTT history. A method for changing these constants is currently an open research question.

4 Clock Granularity

There is no requirement for the clock granularity G used for computing RTT measurements and the different state variables. However, if the K*RTTVAR term in the RTO calculation equals zero, the variance term MUST be rounded to G seconds (i.e., use the equation given in step 2.3).

```
    RTO <- SRTT + max (G, K*RTTVAR)
```

Experience has shown that finer clock granularities (<= 100 msec) perform somewhat better than more coarse granularities.

Note that [Jac88] outlines several clever tricks that can be used to obtain better precision from coarse granularity timers. These changes are widely implemented in current TCP implementations.

5 Managing the RTO Timer

An implementation MUST manage the retransmission timer(s) in such a way that a segment is never retransmitted too early, i.e. less than one RTO after the previous transmission of that segment.

The following is the RECOMMENDED algorithm for managing the retransmission timer:

(5.1) Every time a packet containing data is sent (including a retransmission), if the timer is not running, start it running so that it will expire after RTO seconds (for the current value of RTO).

(5.2) When all outstanding data has been acknowledged, turn off the retransmission timer.

(5.3) When an ACK is received that acknowledges new data, restart the retransmission timer so that it will expire after RTO seconds (for the current value of RTO).

When the retransmission timer expires, do the following:

(5.4) Retransmit the earliest segment that has not been acknowledged by the TCP receiver.

(5.5) The host MUST set RTO <- RTO * 2 ("back off the timer"). The maximum value discussed in (2.5) above may be used to provide an upper bound to this doubling operation.

(5.6) Start the retransmission timer, such that it expires after RTO seconds (for the value of RTO after the doubling operation outlined in 5.5).

Note that after retransmitting, once a new RTT measurement is obtained (which can only happen when new data has been sent and acknowledged), the computations outlined in section 2 are performed, including the computation of RTO, which may result in "collapsing" RTO back down after it has been subject to exponential backoff (rule 5.5).

Note that a TCP implementation MAY clear SRTT and RTTVAR after backing off the timer multiple times as it is likely that the current SRTT and RTTVAR are bogus in this situation. Once SRTT and RTTVAR are cleared they should be initialized with the next RTT sample taken per (2.2) rather than using (2.3).

6 Security Considerations

This document requires a TCP to wait for a given interval before retransmitting an unacknowledged segment. An attacker could cause a TCP sender to compute a large value of RTO by adding delay to a timed packet's latency, or that of its acknowledgment. However, the ability to add delay to a packet's latency often coincides with the ability to cause the packet to be lost, so it is difficult to see what an attacker might gain from such an attack that could cause more damage than simply discarding some of the TCP connection's packets.

The Internet to a considerable degree relies on the correct implementation of the RTO algorithm (as well as those described in RFC 2581) in order to preserve network stability and avoid congestion collapse. An attacker could cause TCP endpoints to respond more aggressively in the face of congestion by forging acknowledgments for segments before the receiver has actually received the data, thus lowering RTO to an unsafe value. But to do so requires spoofing the acknowledgments correctly, which is difficult unless the attacker can monitor traffic along the path between the sender and the receiver. In addition, even if the attacker can cause the sender's RTO to reach too small a value, it appears the attacker cannot leverage this into much of an attack (compared to the other damage they can do if they can spoof packets belonging to the connection), since the sending TCP will still back off its timer in the face of an incorrectly transmitted packet's loss due to actual congestion.

Acknowledgments

The RTO algorithm described in this memo was originated by Van Jacobson in [Jac88].

References

[AP99] Allman, M. and V. Paxson, "On Estimating End-to-End Network Path Properties", SIGCOMM 99.

[APS99] Allman, M., Paxson V. and W. Stevens, "TCP Congestion Control", RFC 2581, April 1999.

[Bra89] Braden, R., "Requirements for Internet Hosts -- Communication Layers", STD 3, RFC 1122, October 1989.

[Bra97] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[Jac88] Jacobson, V., "Congestion Avoidance and Control", Computer Communication Review, vol. 18, no. 4, pp. 314-329, Aug. 1988.

[JK88] Jacobson, V. and M. Karels, "Congestion Avoidance and Control", ftp://ftp.ee.lbl.gov/papers/congavoid.ps.Z.

[KP87] Karn, P. and C. Partridge, "Improving Round-Trip Time Estimates in Reliable Transport Protocols", SIGCOMM 87.

[Pos81] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

Author's Addresses

Vern Paxson
ACIRI / ICSI
1947 Center Street
Suite 600
Berkeley, CA 94704-1198

Phone: 510-666-2882
Fax: 510-643-7684
EMail: vern@aciri.org
http://www.aciri.org/vern/

Mark Allman
NASA Glenn Research Center/BBN Technologies
Lewis Field
21000 Brookpark Rd. MS 54-2
Cleveland, OH 44135

Phone: 216-433-6586
Fax: 216-433-8705
EMail: mallman@grc.nasa.gov
http://roland.grc.nasa.gov/~mallman

Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.