Network Working Group
Request for Comments: 1185

V. Jacobson
LBL
R. Braden
ISI
L. Zhang
PARC

October 1990

TCP Extension for High-Speed Paths

Status of This Memo

Summary

This memo describes a small extension to TCP to support reliable operation over very high-speed paths, using sender timestamps transmitted using the TCP Echo option proposed in RFC-1072.

1. INTRODUCTION

TCP uses positive acknowledgments and retransmissions to provide reliable end-to-end delivery over a full-duplex virtual circuit called a connection [Postel81]. A connection is defined by its two end points; each end point is a "socket", i.e., a (host,port) pair. To protect against data corruption, TCP uses an end-to-end checksum. Duplication and reordering are handled using a fine-grained sequence number space, with each octet receiving a distinct sequence number.

The TCP protocol [Postel81] was designed to operate reliably over almost any transmission medium regardless of transmission rate, delay, corruption, duplication, or reordering of segments. In practice, proper TCP implementations have demonstrated remarkable robustness in adapting to a wide range of network characteristics. For example, TCP implementations currently adapt to transfer rates in the range of 100 bps to 10**7 bps and round-trip delays in the range 1 ms to 100 seconds.

However, the introduction of fiber optics is resulting in ever-higher transmission speeds, and the fastest paths are moving out of the domain for which TCP was originally engineered. This memo and RFC-1072 [Jacobson88] propose modest extensions to TCP to extend the domain of its application to higher speeds.

There is no one-line answer to the question: "How fast can TCP go?". The issues are reliability and performance, and these depend upon the round-trip delay and the maximum time that segments may be queued in the Internet, as well as upon the transmission speed. We must think through these relationships very carefully if we are to successfully extend TCP's domain.

TCP performance depends not upon the transfer rate itself, but rather upon the product of the transfer rate and the round-trip delay. This "bandwidth*delay product" measures the amount of data that would "fill the pipe"; it is the buffer space required at sender and receiver to obtain maximum throughput on the TCP connection over the path. RFC-1072 proposed a set of TCP extensions to improve TCP efficiency for "LFNs" (long fat networks), i.e., networks with large bandwidth*delay products.

On the other hand, high transfer rate can threaten TCP reliability by violating the assumptions behind the TCP mechanism for duplicate detection and sequencing. The present memo specifies a solution for this problem, extending TCP reliability to transfer rates well beyond the foreseeable upper limit of bandwidth.

An especially serious kind of error may result from an accidental reuse of TCP sequence numbers in data segments. Suppose that an "old duplicate segment", e.g., a duplicate data segment that was delayed in Internet queues, was delivered to the receiver at the wrong moment so that its sequence numbers fell somewhere within the current window. There would be no checksum failure to warn of the error, and the result could be an undetected corruption of the data. Reception of an old duplicate ACK segment at the transmitter could be only slightly less serious: it is likely to lock up the connection so that no further progress can be made and a RST is required to resynchronize the two ends.

Duplication of sequence numbers might happen in either of two ways:

(1) Sequence number wrap-around on the current connection

A TCP sequence number contains 32 bits. At a high enough transfer rate, the 32-bit sequence space may be "wrapped" (cycled) within the time that a segment may be delayed in queues. Section 2 discusses this case and proposes a mechanism to reject old duplicates on the current connection.

(2) Segment from an earlier connection incarnation

Suppose a connection terminates, either by a proper close sequence or due to a host crash, and the same connection (i.e., using the same pair of sockets) is immediately reopened. A delayed segment from the terminated connection could fall within the current window for the new incarnation and be accepted as valid. This case is discussed in Section 3.

TCP reliability depends upon the existence of a bound on the lifetime of a segment: the "Maximum Segment Lifetime" or MSL. An MSL is generally required by any reliable transport protocol, since every sequence number field must be finite, and therefore any sequence number may eventually be reused. In the Internet protocol suite, the MSL bound is enforced by an IP-layer mechanism, the "Time-to-Live" or TTL field.

Watson's Delta-T protocol [Watson81] includes network-layer mechanisms for precise enforcement of an MSL. In contrast, the IP mechanism for MSL enforcement is loosely defined and even more loosely implemented in the Internet. Therefore, it is unwise to depend upon active enforcement of MSL for TCP connections, and it is unrealistic to imagine setting MSL's smaller than the current values (e.g., 120 seconds specified for TCP). The timestamp algorithm described in the following section gives a way out of this dilemma for high-speed networks.

2. SEQUENCE NUMBER WRAP-AROUND

2.1 Background

Avoiding reuse of sequence numbers within the same connection is simple in principle: enforce a segment lifetime shorter than the time it takes to cycle the sequence space, whose size is effectively 2**31.

More specifically, if the maximum effective bandwidth at which TCP is able to transmit over a particular path is B bytes per second, then the following constraint must be satisfied for error-free operation:

```
    2**31 / B  > MSL (secs)                                 [1]
```

The following table shows the value for Twrap = 2**31/B in seconds, for some important values of the bandwidth B:

Network	B*8 bits/sec	B bytes/sec	Twrap secs
ARPANET	56kbps	7KBps	3*10**5 (~3.6 days)
DS1	1.5Mbps	190KBps	10**4 (~3 hours)
Ethernet	10Mbps	1.25MBps	1700 (~30 mins)
DS3	45Mbps	5.6MBps	380
FDDI	100Mbps	12.5MBps	170
Gigabit	1Gbps	125MBps	17

It is clear why wrap-around of the sequence space was not a problem for 56kbps packet switching or even 10Mbps Ethernets. On the other hand, at DS3 and FDDI speeds, Twrap is comparable to the 2 minute MSL assumed by the TCP specification [Postel81]. Moving towards gigabit speeds, Twrap becomes too small for reliable enforcement by the Internet TTL mechanism.

The 16-bit window field of TCP limits the effective bandwidth B to 2**16/RTT, where RTT is the round-trip time in seconds [McKenzie89]. If the RTT is large enough, this limits B to a value that meets the constraint [1] for a large MSL value. For example, consider a transcontinental backbone with an RTT of 60ms (set by the laws of physics). With the bandwidth*delay product limited to 64KB by the TCP window size, B is then limited to 1.1MBps, no matter how high the theoretical transfer rate of the path. This corresponds to cycling the sequence number space in Twrap= 2000 secs, which is safe in today's Internet.

Based on this reasoning, an earlier RFC [McKenzie89] has cautioned that expanding the TCP window space as proposed in RFC-1072 will lead to sequence wrap-around and hence to possible data corruption. We believe that this is mis-identifying the culprit, which is not the larger window but rather the high bandwidth.

For example, consider a (very large) FDDI LAN with a diameter of 10km. Using the speed of light, we can compute the RTT across the ring as (2*10**4)/(3*10**8) = 67 microseconds, and the delay*bandwidth product is then 833 bytes. A TCP connection across this LAN using a window of only 833 bytes will run at the full 100mbps and can wrap the sequence space in about 3 minutes, very close to the MSL of TCP. Thus, high speed alone can cause a reliability problem with sequence number wrap-around, even without extended windows.

An "obvious" fix for the problem of cycling the sequence space is to increase the size of the TCP sequence number field. For example, the sequence number field (and also the acknowledgment field) could be expanded to 64 bits. However, the proposals for making such a change while maintaining compatibility with current TCP have tended towards complexity and ugliness.

This memo proposes a simple solution to the problem, using the TCP echo options defined in RFC-1072. Section 2.2 which follows describes the original use of these options to carry timestamps in order to measure RTT accurately. Section 2.3 proposes a method of using these same timestamps to reject old duplicate segments that could corrupt an open TCP connection. Section 3 discusses the application of this mechanism to avoiding old duplicates from previous incarnations.

2.2 TCP Timestamps

RFC-1072 defined two TCP options, Echo and Echo Reply. Echo carries a 32-bit number, and the receiver of the option must return this same value to the source host in an Echo Reply option.

RFC-1072 furthermore describes the use of these options to contain 32-bit timestamps, for measuring the RTT. A TCP sending data would include Echo options containing the current clock value. The receiver would echo these timestamps in returning segments (generally, ACK segments). The difference between a timestamp from an Echo Reply option and the current time would then measure the RTT at the sender.

This mechanism was designed to solve the following problem: almost all TCP implementations base their RTT measurements on a sample of only one packet per window. If we look at RTT estimation as a signal processing problem (which it is), a data signal at some frequency (the packet rate) is being sampled at a lower frequency (the window rate). Unfortunately, this lower sampling frequency violates Nyquist's criteria and may introduce "aliasing" artifacts into the estimated RTT [Hamming77].

A good RTT estimator with a conservative retransmission timeout calculation can tolerate the aliasing when the sampling frequency is "close" to the data frequency. For example, with a window of 8 packets, the sample rate is 1/8 the data frequency -- less than an order of magnitude different. However, when the window is tens or hundreds of packets, the RTT estimator may be seriously in error, resulting in spurious retransmissions.

A solution to the aliasing problem that actually simplifies the sender substantially (since the RTT code is typically the single biggest protocol cost for TCP) is as follows: the will sender place a timestamp in each segment and the receiver will reflect these timestamps back in ACK segments. Then a single subtract gives the sender an accurate RTT measurement for every ACK segment (which will correspond to every other data segment, with a sensible receiver). RFC-1072 defined a timestamp echo option for this purpose.

It is vitally important to use the timestamp echo option with big windows; otherwise, the door is opened to some dangerous instabilities due to aliasing. Furthermore, the option is probably useful for all TCP's, since it simplifies the sender.

2.3 Avoiding Old Duplicate Segments

Timestamps carried from sender to receiver in TCP Echo options can also be used to prevent data corruption caused by sequence number wrap-around, as this section describes.

2.3.1 Basic Algorithm

Assume that every received TCP segment contains a timestamp. The basic idea is that a segment received with a timestamp that is earlier than the timestamp of the most recently accepted segment can be discarded as an old duplicate. More specifically, the following processing is to be performed on normal incoming segments:

R1) If the timestamp in the arriving segment timestamp is less than the timestamp of the most recently received in-sequence segment, treat the arriving segment as not acceptable:

If SEG.LEN > 0, send an acknowledgement in reply as specified in RFC-793 page 69, and drop the segment; otherwise, just silently drop the segment.*

*Sending an ACK segment in reply is not strictly necessary, since the case can only arise when a later in-order segment has already been received. However, for consistency and simplicity, we suggest treating a timestamp failure the same way TCP treats any other unacceptable segment.

R2) If the segment is outside the window, reject it (normal TCP processing)

R3) If an arriving segment is in-sequence (i.e, at the left window edge), accept it normally and record its timestamp.

R4) Otherwise, treat the segment as a normal in-window, out-of-sequence TCP segment (e.g., queue it for later delivery to the user).

Steps R2-R4 are the normal TCP processing steps specified by RFC-793, except that in R3 the latest timestamp is set from each in-sequence segment that is accepted. Thus, the latest timestamp recorded at the receiver corresponds to the left edge of the window and only advances when the left edge moves [Jacobson88].

It is important to note that the timestamp is checked only when a segment first arrives at the receiver, regardless of whether it is in-sequence or is queued. Consider the following example.

> Suppose the segment sequence: A.1, B.1, C.1, ..., Z.1 has been sent, where the letter indicates the sequence number and the digit represents the timestamp. Suppose also that segment B.1 has been lost. The highest in-sequence timestamp is 1 (from A.1), so C.1, ..., Z.1 are considered acceptable and are queued. When B is retransmitted as segment B.2 (using the latest timestamp), it fills the hole and causes all the segments through Z to be acknowledged and passed to the user. The timestamps of the queued segments are *not* inspected again at this time, since they have already been accepted. When B.2 is accepted, the receivers’s current timestamp is set to 2.

This rule is vital to allow reasonable performance under loss. A full window of data is in transit at all times, and after a loss a full window less one packet will show up out-of-sequence to be queued at the receiver (e.g., up to ~2**30 bytes of data); the timestamp option must not result in discarding this data.

In certain unlikely circumstances, the algorithm of rules R1-R4 could lead to discarding some segments unnecessarily, as shown in the following example:

> Suppose again that segments: A.1, B.1, C.1, ..., Z.1 have

been sent in sequence and that segment B.1 has been lost. Furthermore, suppose delivery of some of C.1, ... Z.1 is delayed until AFTER the retransmission B.2 arrives at the receiver. These delayed segments will be discarded unnecessarily when they do arrive, since their timestamps are now out of date.

This case is very unlikely to occur. If the retransmission was triggered by a timeout, some of the segments C.1, ... Z.1 must have been delayed longer than the RTO time. This is presumably an unlikely event, or there would be many spurious timeouts and retransmissions. If B's retransmission was triggered by the "fast retransmit" algorithm, i.e., by duplicate ACK's, then the queued segments that caused these ACK's must have been received already.

Even if a segment was delayed past the RTO, the selective acknowledgment (SACK) facility of RFC-1072 will cause the delayed packets to be retransmitted at the same time as B.2, avoiding an extra RTT and therefore causing a very small performance penalty.

We know of no case with a significant probability of occurrence in which timestamps will cause performance degradation by unnecessarily discarding segments.

2.3.2 Header Prediction

"Header prediction" [Jacobson90] is a high-performance transport protocol implementation technique that is is most important for high-speed links. This technique optimizes the code for the most common case: receiving a segment correctly and in order. Using header prediction, the receiver asks the question, "Is this segment the next in sequence?" This question can be answered in fewer machine instructions than the question, "Is this segment within the window?"

Adding header prediction to our timestamp procedure leads to the following sequence for processing an arriving TCP segment:

H1) Check timestamp (same as step R1 above)

H2) Do header prediction: if segment is next in sequence and if there are no special conditions requiring additional processing, accept the segment, record its timestamp, and skip H3.

H3) Process the segment normally, as specified in RFC-793.

This includes dropping segments that are outside the window and possibly sending acknowledgments, and queueing in-window, out-of-sequence segments.

However, the timestamp check in step H1 is very unlikely to fail, and it is a relatively expensive operation since it requires interval arithmetic on a finite field. To perform this check on every single segment seems like poor implementation engineering, defeating the purpose of header prediction. Therefore, we suggest that an implementor interchange H1 and H2, i.e., perform header prediction FIRST, performing H1 and H3 only if header prediction fails. We believe that this change might gain 5-10% in performance on high-speed networks.

This reordering does raise a theoretical hazard: a segment from 2**32 bytes in the past may arrive at exactly the wrong time and be accepted mistakenly by the header-prediction step. We make the following argument to show that the probability of this failure is negligible.

If all segments are equally likely to show up as old duplicates, then the probability of an old duplicate exactly matching the left window edge is the maximum segment size (MSS) divided by the size of the sequence space. This ratio must be less than 2**-16, since MSS must be < 2**16; for example, it will be (2**12)/(2**32) = 2**-20 for an FDDI link. However, the older a segment is, the less likely it is to be retained in the Internet, and under any reasonable model of segment lifetime the probability of an old duplicate exactly at the left window edge must be much smaller than 2**16.

The 16 bit TCP checksum also allows a basic unreliability of one part in 2**16. A protocol mechanism whose reliability exceeds the reliability of the TCP checksum should be considered "good enough", i.e., it won't contribute significantly to the overall error rate. We therefore believe we can ignore the problem of an old duplicate being accepted by doing header prediction before checking the timestamp.

2.3.3 Timestamp Frequency

It is important to understand that the receiver algorithm for timestamps does not involve clock synchronization with the sender. The sender's clock is used to stamp the segments, and the sender uses this fact to measure RTT's. However, the receiver treats the timestamp as simply a monotone-increasing serial number, without any necessary connection to its clock. From the receiver’s viewpoint, the timestamp is acting as a logical extension of the high-order bits of the sequence number.

However, the receiver algorithm dpes place some requirements on the frequency of the timestamp "clock":

(a) Timestamp clock must not be "too slow".

It must tick at least once for each 2**31 bytes sent. In fact, in order to be useful to the sender for round trip timing, the clock should tick at least once per window’s worth of data, and even with the RFC-1072 window extension, 2**31 bytes must be at least two windows.

To make this more quantitative, any clock faster than 1 tick/sec will reject old duplicate segments for link speeds of ~2 Gbps; a 1ms clock will work up to link speeds of 2 Tbps (10**12 bps!).

(b) Timestamp clock must not be "too fast".

Its cycling time must be greater than MSL seconds. Since the clock (timestamp) is 32 bits and the worst-case MSL is 255 seconds, the maximum acceptable clock frequency is one tick every 59 ns.

However, since the sender is using the timestamp for RTT calculations, the timestamp doesn’t need to have much more resolution than the granularity of the retransmit timer, e.g., tens or hundreds of milliseconds.

Thus, both limits are easily satisfied with a reasonable clock rate in the range 1-100ms per tick.

Using the timestamp option relaxes the requirements on MSL for avoiding sequence number wrap-around. For example, with a 1 ms timestamp clock, the 32-bit timestamp will wrap its sign bit in 25 days. Thus, it will reject old duplicates on the same connection as long as MSL is 25 days or less. This appears to be a very safe figure. If the timestamp has 10 ms resolution, the MSL requirement is boosted to 250 days. An MSL of 25 days or longer can probably be assumed by the gateway system without requiring precise MSL enforcement by the TTL value in the IP layer.

3. DUPLICATES FROM EARLIER INCARNATIONS OF CONNECTION

We turn now to the second potential cause of old duplicate packet errors: packets from an earlier incarnation of the same connection. The appendix contains a review the mechanisms currently included in TCP to handle this problem. These mechanisms depend upon the enforcement of a maximum segment lifetime (MSL) by the Internet layer.

The MSL required to prevent failures due to an earlier connection incarnation does not depend (directly) upon the transfer rate. However, the timestamp option used as described in Section 2 can provide additional security against old duplicates from earlier connections. Furthermore, we will see that with the universal use of the timestamp option, enforcement of a maximum segment lifetime would no longer be required for reliable TCP operation.

There are two cases to be considered (see the appendix for more explanation): (1) a system crashing (and losing connection state) and restarting, and (2) the same connection being closed and reopened without a loss of host state. These will be described in the following two sections.

3.1 System Crash with Loss of State

TCP's quiet time of one MSL upon system startup handles the loss of connection state in a system crash/restart. For an explanation, see for example "When to Keep Quiet" in the TCP protocol specification [Postel81]. The MSL that is required here does not depend upon the transfer speed. The current TCP MSL of 2 minutes seems acceptable as an operational compromise, as many host systems take this long to boot after a crash.

However, the timestamp option may be used to ease the MSL requirements (or to provide additional security against data corruption). If timestamps are being used and if the timestamp clock can be guaranteed to be monotonic over a system crash/restart, i.e., if the first value of the sender's timestamp clock after a crash/restart can be guaranteed to be greater than the last value before the restart, then a quiet time will be unnecessary.

To dispense totally with the quiet time would seem to require that the host clock be synchronized to a time source that is stable over the crash/restart period, with an accuracy of one timestamp clock tick or better. Fortunately, we can back off from this strict requirement. Suppose that the clock is always re-synchronized to within N timestamp clock ticks and that booting (extended with a quiet time, if necessary) takes more than N ticks. This will guarantee monotonicity of the timestamps, which can then be used to reject old duplicates even without an enforced MSL.

3.2 Closing and Reopening a Connection

When a TCP connection is closed, a delay of 2*MSL in TIME-WAIT state ties up the socket pair for 4 minutes (see Section 3.5 of [Postel81]. Applications built upon TCP that close one connection and open a new one (e.g., an FTP data transfer connection using Stream mode) must choose a new socket pair each time. This delay serves two different purposes:

(a) Implement the full-duplex reliable close handshake of TCP.

The proper time to delay the final close step is not really related to the MSL; it depends instead upon the RTO for the FIN segments and therefore upon the RTT of the path.* Although there is no formal upper-bound on RTT, common network engineering practice makes an RTT greater than 1 minute very unlikely. Thus, the 4 minute delay in TIME-WAIT state works satisfactorily to provide a reliable full-duplex TCP close. Note again that this is independent of MSL enforcement and network speed.

The TIME-WAIT state could cause an indirect performance problem if an application needed to repeatedly close one connection and open another at a very high frequency, since the number of available TCP ports on a host is less than 2**16. However, high network speeds are not the major contributor to this problem; the RTT is the limiting factor in how quickly connections can be opened and closed. Therefore, this problem will no worse at high transfer speeds.

(b) Allow old duplicate segements to expire.

Suppose that a host keeps a cache of the last timestamp received from each remote host. This can be used to reject old duplicate segments from earlier incarnations of the

*Note: It could be argued that the side that is sending a FIN knows what degree of reliability it needs, and therefore it should be able to determine the length of the TIME-WAIT delay for the FIN's recipient. This could be accomplished with an appropriate TCP option in FIN segments.

connection, if the timestamp clock can be guaranteed to have ticked at least once since the old conennection was open. This requires that the TIME-WAIT delay plus the RTT together must be at least one tick of the sender's timestamp clock.

Note that this is a variant on the mechanism proposed by Garlick, Rom, and Postel (see the appendix), which required each host to maintain connection records containing the highest sequence numbers on every connection. Using timestamps instead, it is only necessary to keep one quantity per remote host, regardless of the number of simultaneous connections to that host.

We conclude that if all hosts used the TCP timestamp algorithm described in Section 2, enforcement of a maximum segment lifetime would be unnecessary and the quiet time at system startup could be shortened or removed. In any case, the timestamp mechanism can provide additional security against old duplicates from earlier connection incarnations. However, a 4 minute TIME-WAIT delay (unrelated to MSL enforcement or network speed) must be retained to provide the reliable close handshake of TCP.

4. CONCLUSIONS

We have presented a mechanism, based upon the TCP timestamp echo option of RFC-1072, that will allow very high TCP transfer rates without reliability problems due to old duplicate segments on the same connection. This mechanism also provides additional security against intrusion of old duplicates from earlier incarnations of the same connection. If the timestamp mechanism were used by all hosts, the quiet time at system startup could be eliminated and enforcement of a maximum segment lifetime (MSL) would no longer be necessary.

REFERENCES

[Cerf76] Cerf, V., "TCP Resynchronization", Tech Note #79, Digital Systems Lab, Stanford, January 1976.

[Dalal74] Dalal, Y., "More on Selecting Sequence Numbers", INWG Protocol Note #4, October 1974.

[Garlick77] Garlick, L., R. Rom, and J. Postel, "Issues in Reliable Host-to-Host Protocols", Proc. Second Berkeley Workshop on Distributed Data Management and Computer Networks, May 1977.

[Hamming77] Hamming, R., "Digital Filters", ISBN 0-13-212571-4, Prentice Hall, Englewood Cliffs, N.J., 1977.

[Jacobson88] Jacobson, V., and R. Braden, "TCP Extensions for Long-Delay Paths", RFC 1072, LBL and USC/Information Sciences Institute, October 1988.

[Jacobson90] Jacobson, V., "4BSD Header Prediction", ACM Computer Communication Review, April 1990.

[McKenzie89] McKenzie, A., "A Problem with the TCP Big Window Option", RFC 1110, BBN STC, August 1989.

[Postel81] Postel, J., "Transmission Control Protocol", RFC 793, DARPA, September 1981.

[Tomlinson74] Tomlinson, R., "Selecting Sequence Numbers", INWG Protocol Note #2, September 1974.

[Watson81] Watson, R., "Timer-based Mechanisms in Reliable Transport Protocol Connection Management", Computer Networks, Vol. 5, 1981.

APPENDIX -- Protection against Old Duplicates in TCP

During the development of TCP, a great deal of effort was devoted to the problem of protecting a TCP connection from segments left from earlier incarnations of the same connection. Several different mechanisms were proposed for this purpose [Tomlinson74] [Dalal74] [Cerf76] [Garlick77].

The connection parameters that are required in this discussion are:

Tc = Connection duration in seconds.

Nc = Total number of bytes sent on connection.

B = Effective bandwidth of connection = Nc/Tc.

Tomlinson proposed a scheme with two parts: a clock-driven selection of ISN (Initial Sequence Number) for a connection, and a resynchronization procedure [Tomlinson74]. The clock-driven scheme chooses:

```
   ISN = (integer(R*t)) mod 2**32                 [2]
```

where t is the current time relative to an arbitrary origin, and R is a constant. R was intended to be chosen so that ISN will advance faster than sequence numbers will be used up on the connection. However, at high speeds this will not be true; the consequences of this will be discussed below.

The clock-driven choice of ISN in formula [2] guarantees freedom from old duplicates matching a reopened connection if the original connection was "short-lived" and "slow". By "short-lived", we mean a connection that stayed open for a time Tc less than the time to cycle the ISN, i.e., Tc < 2**32/R seconds. By "slow", we mean that the effective transfer rate B is less than R.

This is illustrated in Figure 1, where sequence numbers are plotted against time. The asterisks show the ISN lines from formula [2], while the circles represent the trajectories of several short-lived incarnations of the same connection, each terminating at the "x".

Note: allowing rapid reuse of connections was believed to be an important goal during the early TCP development. This requirement was driven by the hope that TCP would serve as a basis for user-level transaction protocols as well as connection-oriented protocols. The paradigm discussed was the "Christmas Tree" or "Kamikazee" segment that contained SYN and FIN bits as well as data. Enthusiasm for this was somewhat dampened when it was observed that the 3-way SYN handshake and the FIN handshake mean that 5 packets are required for a minimum exchange. Furthermore, the TIME-WAIT state delay implies that the same connection really cannot be reopened immediately. No further work has been done in this area, although existing applications (especially SMTP) often generate very short TCP sessions. The reuse problem is generally avoided by using a different port pair for each connection.

```
        |- 2**32       ISN             ISN
        |              *               *
        |             *               *
        |            *               *
        |           *x              *
        |          o               *
    ^   |         *               *
    |   |        *  x            *
        |       * o             *
    S   |      *o              *
    e   |     o               *
    q   |    *               *
        |   *               *
    #   |  * x             *
        | *o              *
        |o_______________*____________
                         ^         Time -->
                       4.55hrs
```

     Figure 1. Clock-Driven ISN avoiding duplication on
                short-Lived, slow connections.

However, clock-driven ISN selection does not protect against old duplicate packets for a long-lived or fast connection: the connection may close (or crash) just as the ISN has cycled around and reached the same value again. If the connection is then reopened, a datagram still in transit from the old connection may fall into the current window. This is illustrated by Figure 2 for a slow, long-lived connection, and by Figures 3 and 4 for fast connections. In each case, the point "x" marks the place at which the original connection closes or crashes. The arrow in Figure 2 illustrates an old duplicate segment. Figure 3 shows a connection whose total byte count Nc < 2**32, while Figure 4 concerns Nc >= 2**32.

To prevent the duplication illustrated in Figure 2, Tomlinson proposed to "resynchronize" the connection sequence numbers if they came within an MSL of the ISN. Resynchronization might take the form of a delay (point "y") or the choice of a new sequence number (point "z").

```
        |- 2**32       ISN               ISN
        |              *                 *
        |             *                 *
        |            *                 *
        |           *                 *
        |          *                 *
    ^   |         *                 *
    |   |        *                 *
        |       *                 *
    S   |      *                 *
    e   |     *                x* y
    q   |    *           o      *
        |   *      o          *z
    #   |  *o                *
        | *                 *
        |*_________________*____________
                           ^         Time -->
                          4.55hrs
```

Figure 2. Resynchronization to Avoid Duplication
on Slow, Long-Lived Connection

```
        |- 2**32       ISN               ISN
        |              *                 *
        |       x   o *                 *
        |            *                 *
        |      o-->o*                 *
        |          *                 *
    ^   |     o   o                 *
    |   |        *                 *
        |    o  *                 *
    S   |      *                 *
    e   |   o *                 *
    q   |    *                 *
        |  o*                 *
    #   |  *                 *
        | o                 *
        |*_________________*____________
                           ^         Time -->
                          4.55hrs
```

Figure 3. Duplication on Fast Connection: Nc < 2**32 bytes

```
        |- 2**32       ISN               ISN
        |      o       *                 *
        |           x *                 *
        |            *                 *
        |     o     *                 *
        |          o                 *
    ^   |         *                 *
    |   |    o   *                 *
        |       * o               *
    S   |      *                *
    e   |   o *                 *
    q   |    *   o             *
        |   *                 *
    #   |  o                 *
        | *     o           *
        |*_________________*____________
                           ^         Time -->
                          4.55hrs
```

Figure 4. Duplication on Fast Connection: Nc > 2**32 bytes

In summary, Figures 1-4 illustrated four possible failure modes for old duplicate packets from an earlier incarnation. We will call these four modes F1 , F2, F3, and F4:

F1: B < R, Tc < 4.55 hrs. (Figure 1)

F2: B < R, Tc >= 4.55 hrs. (Figure 2)

F3: B >= R, Nc < 2**32 (Figure 3)

F4: B >= R, Nc >= 2**32 (Figure 4)

Another limitation of clock-driven ISN selection should be mentioned. Tomlinson assumed that the current time t in formula [2] is obtained from a clock that is persistent over a system crash. For his scheme to work correctly, the clock must be restarted with an accuracy of 1/R seconds (e.g, 4 microseconds in the case of TCP). While this may be possible for some hosts and some crashes, in most cases there will be an uncertainty in the clock after a crash that ranges from a second to several minutes.

As a result of this random clock offset after system reinitialization, there is a possibility that old segments sent before the crash may fall into the window of a new connection incarnation. The solution to this problem that was adopted in the final TCP spec is a "quiet time" of MSL seconds when the system is initialized [Postel81, p. 28]. No TCP connection can be opened until the expiration of this quiet time.

A different approach was suggested by Garlick, Rom, and Postel [Garlick77]. Rather than using clock-driven ISN selection, they proposed to maintain connection records containing the last ISN used on every connection. To immediately open a new incarnation of a connection, the ISN is taken to be greater than the last sequence number of the previous incarnation, so that the new incarnation will have unique sequence numbers. To handle a system crash, they proposed a quiet time, i.e., a delay at system startup time to allow old duplicates to expire. Note that the connection records need be kept only for MSL seconds; after that, no collision is possible, and a new connection can start with sequence number zero.

The scheme finally adopted for TCP combines features of both these proposals. TCP uses three mechanisms:

(A) ISN selection is clock-driven to handle short-lived connections. The parameter R = 250KBps, so that the ISN value cycles in 2**32/R = 4.55 hours.

(B) (One end of) a closed connection is left in a "busy" state, known as "TIME-WAIT" state, for a time of 2*MSL. TIME-WAIT state handles the proper close of a long-lived connection without resynchronization. It also allows reliable completion of the full-duplex close handshake.

(C) There is a quiet time of one MSL at system startup. This handles a crash of a long-lived connection and avoids time resynchronization problems in (A).

Notice that (B) and (C) together are logically sufficient to prevent accidental reuse of sequence numbers from a different incarnation, for any of the failure modes F1-F4. (A) is not logically necessary since the close delay (B) makes it impossible to reopen the same TCP connection immediately. However, the use of (A) does give additional assurance in a common case, perhaps compensating for a host that has set its TIME-WAIT state delay too short.

Some TCP implementations have permitted a connection in the TIME-WAIT state to be reopened immediately by the other side, thus short-circuiting mechanism (B). Specifically, a new SYN for the same socket pair is accepted when the earlier incarnation is still in TIME-WAIT state. Old duplicates in one direction can be avoided by choosing the ISN to be the next unused sequence number from the preceding connection (i.e., FIN+1); this is essentially an application of the scheme of Garlick, Rom, and Postel, using the connection block in TIME-WAIT state as the connection record.

However, the connection is still vulnerable to old duplicates in the other direction. Mechanism (A) prevents trouble in mode F1, but failures can arise in F2, F3, or F4; of these, F2, on short, fast connections, is the most dangerous.

Finally, we note TCP will operate reliably without any MSL-based mechanisms in the following restricted domain:

* Total data sent is less then 2**32 octets, and

* Effective sustained rate less than 250KBps, and

* Connection duration less than 4.55 hours.

At the present time, the great majority of current TCP usage falls into this restricted domain. The third component, connection duration, is the most commonly violated.

Security Considerations

Security issues are not discussed in this memo.

Authors' Addresses

Van Jacobson
University of California
Lawrence Berkeley Laboratory
Mail Stop 46A
Berkeley, CA 94720

Phone: (415) 486-6411
EMail: van@CSAM.LBL.GOV

Bob Braden
University of Southern California
Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292

Phone: (213) 822-1511
EMail: Braden@ISI.EDU

Lixia Zhang
XEROX Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304

Phone: (415) 494-4415
EMail: lixia@PARC.XEROX.COM