

The TCP Maximum Survivable Failure Time

Christopher C. Lord
Information Networking Institute
Carnegie Mellon University
4616 Henry Street, Pittsburgh, PA 15213
clord@cmu.edu

Abstract

TCP connections can tolerate short periods of network outage based on well-known algorithms for computing the roundtrip time and retransmission timeout. There are, however, many other factors that affect how long this survivable failure time can be. This paper examines these factors and develops interrogation and inspection techniques that can estimate the worst-case survivable failure time. These techniques are implemented in a new tool called TProbe and the results reported for TCP implementations on Windows XP, Linux 2.4, FreeBSD 4.7 and SunOS 5.8. Information on the survivable failure time has implications for switching, routing and wireless protocols and other high-availability networking solutions that include failure detection and recovery.

Two modifications to TCP are suggested that may improve connection failure survivability. The first limits the use of delayed acknowledgements; the second limits the minimum retransmission time. The effectiveness and performance of these changes are subjects for further study.

1 Introduction

When a TCP connection fails, many network applications fail gracelessly: errors are passed up to the user for manual restart or recovery. This occurs in POP3, IMAP4, FTP, HTTP/1.1, CIFS, TELNET and other application-layer protocols with typically long-lived connections. These failures could be avoided by switching and routing protocols and by high-availability network solutions provided detection, reconfiguration and recovery occurred below the TCP connection timeout. Network protocol designers would like this timeout to be as long as possible since availability activities have performance costs proportional to the speed of failure detection. However, designers also need to know the minimum possible value for a given environment to balance detection time against the probability of connection failure. Other applications where knowledge of the survivable failure time is useful include wireless protocols such as TCP-Snoop [BSK95]. Additionally, some recovery activities may

schedulable and could benefit from knowing how to shape TCP traffic to increase the connection timeout.

For a steady-state connection, the survivable failure time for a TCP connection is based on the number of allowed retransmissions and the implemented algorithms for computing the retransmission timeout (RTO) and the roundtrip time estimate (RTT). The time is further influenced by clock granularity, initial RTO, upper and lower bounds on the RTO, and the correctness of the implementation.

2 Related Work

The RTO calculation was recommended in RFC 1122 [Bra89] to follow the algorithms in [Jac88]. Much of the behavior was left to the implementation. More recently, RFC 2988 [PA00] clarifies and codifies the algorithm for computing the RTO and removes most of the ambiguity and implementation-dependent behavior.

Paxson and Allman have both conducted studies of TCP retransmission behavior. In [AP99] the authors assess several RTO algorithms and evaluate the effectiveness of these algorithms through trace-driven simulation. The findings show that the performance of estimators is dominated by their minimum values and, to a lesser extent, by clock granularity. The frequency of samples and the weights in the exponentially-weighted moving averages have virtually no effect.

The RTO is dependent on RTT estimates. Since the RTT estimate is internal to the TCP implementation, it is useful to approximate the RTT using passive mechanisms that don't interfere with the host. Two techniques to passively estimate RTTs are presented in [JD02]. These are based on an analysis of connection startup (SYN-ACK exchanges) and slow-start window expansion. These techniques are applied to a number of hosts across the Internet and the resulting calculations compared with those generated with ICMP echo messages. Reasonably accurate estimates of the internal TCP RTT estimate can then be used to infer RTO algorithm implementation. The SYN-ACK RTT estimation technique is used in the current study,

Other studies have indirectly examined the role of clock granularity. A thorough review of the effects of clock

granularity and interrupt processing latency on operating systems is given in [ETF02]. The discussion applies equally well to kernel-mode protocol processing. In [Pax98], Paxson developed algorithms for measuring packet transit times given timestamps of departure and arrival times of unsynchronized hosts and with potentially different clock granularities. While this research is not concerned with retransmission behavior, the techniques employed to estimate clock granularity are applied in the current study.

Similarly, [VCH02] presents a technique to fingerprint a platform based on TCP protocol behavior. The authors use the different arrival times and total number of retransmitted SYN-ACK packets to create a profile of a remote system based on the assumption that the RTT will not vary dramatically during the sample.

This study is primarily concerned with developing a profile of the most “optimistic” TCP retransmission behavior and typical network conditions. There have been several other studies which likewise attempt to indirectly profile the behavior of TCP implementations. [PF01] presents the techniques developed in a tool called `TBIT` (TCP Behavior Identification Tool) which is then applied to a sample of Web servers to test conformance with different TCP standards and recommendations and to identify the flavor of TCP based on responses. This tool was in turn built upon the `Sting` tool and the work done in [Sav99] that developed techniques to independently investigate the loss rates along the forward and reverse paths between hosts. The implementation presented here builds upon both tools.

3 TCP Retransmission Behavior

3.1 Clock Granularity

Time-based calculations are limited by the clock granularity or resolution, the smallest amount of time in each clock update. For example, Windows NT interrupts on a periodic basis, typically 10-15ms depending on the hardware platform such that the timer services available to kernel-mode drivers cannot be more accurate than this. Some implementations may also use a coarser system timer on the order of 100-500ms to manage protocol timers, further constraining the minimum possible RTO. The clock granularity is the minimum variance term in computing the RTO.

Most modern hardware platforms also provide a high-precision timestamp through the processor that is independent of the system clock. The Intel Pentium and compatible processors provide a cycle count that can be used to compute nanosecond accuracies. This does not however affect the granularity of the kernel timer services, only the precision of the timestamps.

3.2 Retransmission Timeout Calculation

According to RFC 1122, The Requirements for Internet Hosts [Bra89], TCP must implement Jacobson’s algorithm for computing the retransmission timeout using a smoothed RTT and incorporated the measured RTT variance. Each successive timeout must be accompanied by an exponential backoff of the RTO for the same segment. New connections should initialize the RTT to 0 seconds and the RTO to 3 seconds using a value for the variance that results in 3 seconds. Finally, the lower bound should be “measured in fractions of a second (to accommodate high speed LANs)” and the upper bound set to 240 seconds. RFC 2988 makes the recommendations for calculating the RTO in RFC 1122 mandatory and further recommends a lower bound on the RTO of 1 second. As will be shown later, if implementations implemented this lower bound, then concerns about the SFT would be unnecessary since the worst case would be a fixed factor of the number of retransmits. Fortunately for this study, no implementations examined clamp the RTO at the minimum recommended value.

Constants	
$RTO_{min} = 1$	Minimum recommended RTO.
$RTO_{initial} = 3$	Recommended initial RTO.
$\alpha = \frac{1}{8} = 0.125$	Weight applied to new RTT samples.
$\beta = \frac{1}{4} = 0.25$	Weight applied to new RTT variance calculations.
$k = 4$	The scaling factor applied to the RTTVAR.
G	The clock granularity used to for the RTO.
Initialization	
$SRTT = RTT$	
$RTTVAR = RTT / 2$	
$RTO = SRTT + \max(G, kRTTVAR)$	
Operation	
$RTTVAR = (1 - \beta) * RTTVAR + \beta * SRTT - RTT $	
$SRTT = (1 - \alpha) * SRTT + \alpha RTT$	
$RTO = SRTT + \max(G, kRTTVAR)$	

Table 3-1: Algorithm for Computing TCP RTO

The constants and algorithms used to compute the initial and operational RTO are shown in Table 3-1. The operational RTO is used as soon as an RTT sample is obtained, either by reception of an ACK for a data segment or by reflection of a TCP timestamp.

3.3 Delayed Acknowledgements

Normally an ACK is sent for every other TCP segment received on a connection. To reduce the number of packets sent, TCP uses delayed acknowledgements. If there is unacknowledged data, an ACK is automatically sent when this timer expires. RFC 1122 recommends the use of delayed acknowledgements, but mandates a

delay of less than 500ms of used. A delay of 200ms is more common. In BSD-derived implementations this delay is uniformly distributed between 0ms and 200ms, while in Windows NT this delay is distributed uniformly between 100ms and 200ms [CSA00]. The delayed acknowledgement timer is a potentially important factor in determining the worst-case survivable failure time because it is hypothesized that TCP implementations should always have an RTO above the maximum possible delayed acknowledgement timer or else risk retransmitting packets that have already been received.

4 Maximum Survivable Failure Time

TCP starts a retransmission timer when each outbound segment is handed down to IP. If no acknowledgment has been received for the data in a given segment before the timer expires, then the segment is retransmitted, up to the maximum number of data transmissions allowed, typically 3-7.

The retransmission timeout is initialized to three seconds when a TCP connection is established; however it is adjusted dynamically to match the characteristics of the connection using an exponentially weighted moving average of the roundtrip times. Retransmits use a binary exponential backoff, doubling after each retransmission of a segment. As a consequence, connections over high-delay links will take longer to time out than those exhibiting low delay.

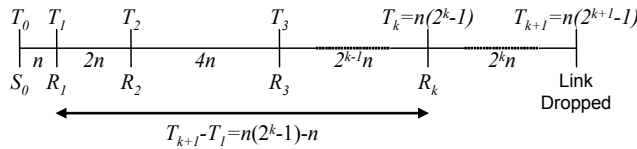


Figure 4-1: TCP Maximum Survivable Failure Time

In the above figure, n is the current retransmission timeout for a connection at the time of a failure and k is the number of retransmit retries. The amount of time, T , that elapses between the time a packet is transmitted and the k th retransmit is the given by Equation 4-1.

$$T_k = n \sum_{i=0}^{k-1} 2^i = n \left(\frac{2^{(k-1)+1} - 1}{2 - 1} \right) = n(2^k - 1) \quad \text{Equation 4-1}$$

The maximum survivable failure time (SFT) is the amount of time between the first and final retransmission of a segment, R_1 and R_k . This is the window of time during which packets can be lost without the connection being terminated and is given by the difference between T_{k+1} and T_1 . Any duration larger than this could eclipse an acknowledgement between the initial send, S_0 , and the first retransmit, R_1 , or all retransmits up to and including R_k , causing the link to be dropped because the data is never acknowledged

from the sender's perspective. The maximum SFT is therefore given by Equation 4-2.

$$T_{k+1} - T_1 = n(2^k - 1) - n \quad \text{Equation 4-2}$$

Many implementations enforce a maximum timeout of between 60 and 240 seconds (RFC 1122 requires a maximum be at least 60 seconds if used). Once the current timeout exceeds the limit, the maximum will be used for any segment that needs to be retransmitted. There is no corresponding lower bound on the retransmission timeout.

Despite this, it is possible to infer a possible lower limit. The RTO should be larger than the worst-case delayed acknowledgement time, 500ms, or packets would be retransmitted prematurely. However, a delayed acknowledgement timer of 200ms is more typical. With a retransmission count of 5, this results in a worst-case maximum SFT given in Equation 4-3:

$$200(2^5 - 1) - 200 = 6000ms \quad \text{Equation 4-3}$$

This 6 seconds is the maximum duration between the last unacknowledged transmission prior to a failure and the final retransmission attempt before the link is terminated. If a 500ms heartbeat timer is used instead, the minimum possible delayed acknowledgement timer (and hence limit on the retransmission timeout) would be 500ms. This allows a more liberal 15 second window. These are clearly worst-case timeouts since a host that uses delayed acknowledgements will usually have the timer factored into the RTO calculations. If, for example, the first segment was delayed by 200ms, then the initial RTO would be greater than or equal to 600ms as shown in Equation 4-4:

$$\begin{aligned} SRTT &\geq 200ms \\ RTTVAR &\geq 200ms / 2 \\ RTO &\geq 200ms + 4(100ms) \end{aligned} \quad \text{Equation 4-4}$$

This yields an initial maximum SFT of 18 seconds for a retransmit limit of 5. Table 4-1 shows the maximum SFT for several other combinations of retransmission times and counts. These values assume negligible roundtrip time variation and fine-grained clocks.

SFT (secs)	Retransmission Count				
	3	4	5	6	7
RTO _{min}					
10ms	0.06	0.14	0.3	0.62	1.26
100ms	0.6	1.4	3	6.2	12.6
200ms	1.2	2.8	6	12.4	25.2
500ms	3	7	15	31	63
1s	6	14	30	62	126
3s	18	42	90	186	378

Table 4-1: SFT for Various RTO and Retransmit Values

As can be seen, the maximum SFT for connections with under a second retransmission times is surprisingly short. In a typical LAN, this is more than just a theoretical concern. TCP connections for Windows 2000 and XP easily attain sub-second RTOs and timeout in around 6 seconds with the default retransmit limit of 5. This renders most high-availability network, switching and routing fault detection mechanisms ineffective for maintaining active TCP connections.

5 Basic Techniques

In order to investigate the maximum SFT, several pieces of information must be obtained. This includes:

RTO_{min} The minimum retransmission timeout. If not explicitly defined, this will be determined by the clock granularity. Implementations may also use a coarser timer, which may further determine the lower bound. In newer implementations that adhere to RFC 2988, the lower bound should be set to 1 second.

RTO_{max} The maximum retransmission timeout. Implementations that provide a maximum timeout will clamp all computed retransmit timeouts at this value. This is not strictly necessary to determine the maximum SFT, but it is an interesting characteristic to know about an implementation.

G The clock granularity. This may be the resolution of the clock or some coarser heartbeat timer used by the protocol stack.

RTT The roundtrip time estimate.

R The maximum number of retransmits allowed.

With this information there are several means of estimating the SFT given the maximum number of retransmits. These are discussed in Section 5.7.

5.1 Estimating the Clock Granularity

Following the technique outlined in [Pax98], the clock resolution is estimated by inspecting the differences between successive packet timestamps and accounting for monotonicity increases. This produces an upper bound on the clock resolution because trace data may not include packets separated by only a single tick. The minimum practical value for *RTO_{min}* is $2G$ because any value less than this would result in a retransmission timer of anywhere between 0 and G . In practice, *RTO_{min}* may be bounded by the larger granularity of protocol timers and not the kernel scheduling timer. This approach does not take into account that many kernel drivers have access to a high-precision timestamp through the processor that is independent of the system clock. The use of such nanosecond

granularity timestamps must be determined by inspection.

In order to obtain back-to-back packets, a TCP connection with the timestamp option set in the SYN is created and an HTTP request for a large static document is sent. If the target supports timestamps, then the option will be detected in data packets and reflected back to the target in acknowledgements. The timestamps between packets contained in each burst during slow start are compared and the smallest difference taken as the estimate for clock granularity.

5.2 Estimating the RTT

The RTO is based on the recently observed exponentially weighted moving average of the RTT and the variance. TCP connection setup provides a good opportunity to estimate the RTT because it typically involves only limited kernel processing. The RTT is estimated from the time interval between the SYN and SYN-ACK as shown in Figure 5-1. This process is repeated several times and the lowest observed RTT used as the minimum.

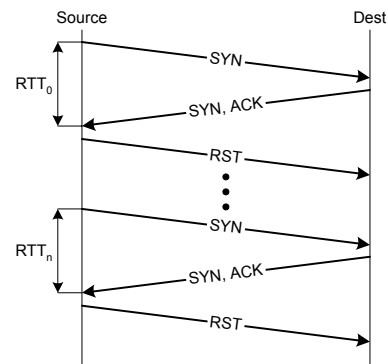


Figure 5-1: Estimating Minimum RTT with SYN-ACK

5.3 Determining the Retransmit Limit

The retransmit limit is determined by establishing a TCP connection with a well-known service such as HTTP, sending a request and not acknowledging the response. This exchange is shown in Figure 5-2. The number of times before a connection reset occurs is the maximum retransmit count for a given implementation. In the event that a reset is not sent, the source will keep track of the maximum retransmit time and timeout when no packet is received beyond this time.

5.4 Determining the Initial RTO

The initial retransmission timer does not affect the SFT of a steady state connection, but is an interesting characteristic of the implementation. It can be obtained following the same procedure as the retransmit limit above and not acknowledging the first data segment. This must be done with TCP timestamps disabled to

prevent the target from obtaining an RTT sample. This is shown in Figure 5-2.

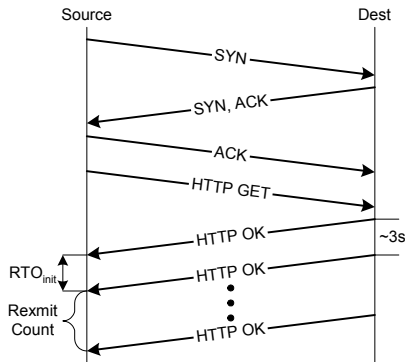


Figure 5-2: Estimating $RTO_{initial}$ and Retransmit Count

5.5 Determining the Maximum RTO

TCP is required to wait for a given interval before retransmitting an unacknowledged segment. TCP implementations are allowed to limit the maximum RTO as long as the limit is at least 60 seconds. The algorithm for determining the minimum RTO is derived from the known security considerations described in RFC 2988 “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.” By slowly increasing the delay between acknowledgements and monitoring retransmissions, it possible to determine whether the implementation has a practical upper bound on the RTO. This is illustrated in Figure 5-3. For example, a delay increment of 200ms could be added to every acknowledgement during a large TCP file transfer up to an apparent RTT of 60 seconds. If there is a maximum RTO, then a retransmission will be detected when the backed-off RTO plateaus at some value.

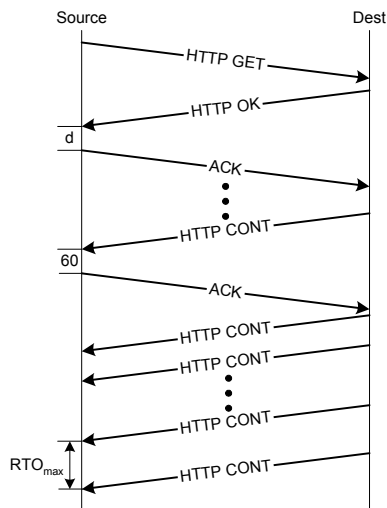


Figure 5-3: Estimating RTO_{max} with Delayed Acks

5.6 Determining the Minimum RTO

TCP is required to provide a non-zero RTO of at least the granularity of the system clock or timer. RFC 2988 recommends a minimum value of one second, but this is rarely implemented.

As with the maximum RTO, the algorithm for determining the minimum RTO is derived from the security considerations described in RFC 2988: “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.”

First a lengthy transaction with a window size of one is profiled and the retransmission time at the end taken as the normal RTO, RTO_{norm} . The same transaction is repeated, but this time by pre-sending a series of acknowledgments based on expected segments. This compresses a host’s RTO to an extremely small value that is less than the actual RTT. An acknowledgement is then dropped and the retransmission time observed. This approach is shown in Figure 5-4.

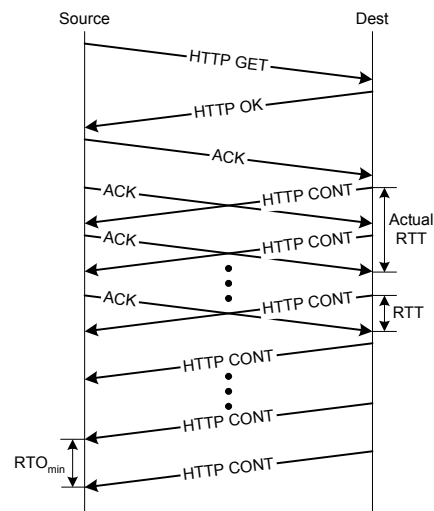


Figure 5-4: Estimating RTO_{min} with Pre-Acks

A second means of compressing the RTO with some implementations that support timestamps is to modify the reflected timestamp by an amount approximately equal to the roundtrip time.

5.7 Calculating the Survivable Failure Time

With information provided by the techniques above, there are three possible SFT calculations:

- SFT_{WC} A worst-case SFT based on the minimum RTT and a variance of $2G$.
- SFT_{min} An SFT based on the compressed RTO_{min}
- SFT_{norm} An SFT based on RTO_{norm} .

It is expected that SFT_{min} will yield larger—and more realistic—limit on the worst-case SFT, while SFT_{norm} will be closer to the SFT seen in actual operation.

6 Implementation

In order to investigate the maximum SFT of different platforms and network environments, a tool called `TProbe` was developed that implements the previously discussed techniques. This tool is built upon the `TBIT` utility developed by Padhye and Floyd and described in [PF01]. The `TBIT` utility was in turn built upon the `Sting` utility developed by Savage and described in [Sav99]. In addition to the code to implement the techniques discussed above, the `TBIT` code was modified to support asynchronous captures and kernel-mode timestamps. The original `TBIT` code blocks on packet reception so a polling interface was added to support pre-acking. `TBIT` also used user-mode timestamps for packets; `TProbe` propagates the kernel-mode timestamps from `libpcap`.

`TProbe` examines the behavior of a remote TCP by opening a raw IP socket, setting a filter to capture all packets to and from the target host, creating a firewall rule to prevent packets from the target system from reaching the local kernel, and fabricating TCP and HTTP packets to probe the desired characteristics. `TProbe` is careful to generate only conformant TCP packets to prevent interactions with firewalls. This is particularly important since `TProbe` is designed to inspect a target system without installing additional software and using only standard network services.

`TProbe` uses the `libpcap` system to capture packets to and from the target host. The timestamps can be delayed if the host system is under load. To minimize such occurrences, the tool should be used on an unloaded system. Unexpected, dropped or reordered packets are detected by the tool and reported as errors.

6.1 Timestamps

The use of TCP timestamps varies by implementation, so it was necessary to determine the resolution encoded in the timestamp before checking the granularity. This was done by observing the interpacket times in retransmits of the first data packet.

Capture Time	Delta	Timestamp	Delta
76909.916734	0.000000	2.817842	0.000000
76909.923453	0.006719	2.817842	0.000000
76912.923029	2.999576	2.818142	0.000300
76918.921847	5.998818	2.818742	0.000600
76930.953179	12.031332	2.819942	0.001200
76955.197212	24.244033	2.822342	0.002400
77004.779371	49.582159	2.827142	0.004800
77102.275823	97.496452	2.836742	0.009600

Table 6-1: Timestamp Resolution from Packet Timestamps

Correct implementations should start from a value of 3 seconds (or other well defined value) with exponential back off. By comparing the timestamp difference between successive retransmits and adjusting to the observed interpacket times, the resolution could be inferred. For example, Table 6-1 shows the capture and packet time stamps from a Linux system. The difference of 0.000300 for the 3 second timeout and 0.000600 for the 6 second time indicates the timestamp resolution is not in microseconds but instead in units of 10ms. With this information, a complete trace of maximum window of transactions is analyzed for the smallest increment of this resolution.

6.2 Roundtrip Times

A window size of 1 segment was used to ensure sufficient exchanges to stretch or compress the timeout. In addition to the maximum and minimum RTO, `TProbe` also tested the normal RTO based on the first retransmission after the same series of exchanges. This RTO is higher than what would be seen in practice because the acknowledgements are generated in user-mode which contributes to latency and variance.

7 Methodology

The behavior of three platforms, (Windows XP, Linux 2.4, FreeBSD 4.7), was investigated using the `TProbe` tool. The systems were all connected in a private 100Mbs switched Ethernet LAN with a separate Linux test system. Each host was configured to use a commonly available HTTP server (IIS for XP, Apache for Linux and MiniHTTPD for FreeBSD) and an identical 150K file was made available at the same URL on each server. The file size was chosen to be large enough to perform the estimation techniques without requiring persistent connections. In addition data was also gathered against a SunOS 5.8 system across a wide area network in order to test the effectiveness of the tool.

Several iterations of the tests were run. For all but RTO_{max} , the lowest observed value was used across all runs. Traces of the test transactions were captured using `tcpdump` on both the source and destination systems and the results used to verify the operation of the tool.

8 Results and Discussion

The results were consistent and repeatable for each of the platforms tested and are reported in Table 8-1. Three possible SFT calculations are computed: the worst-case SFT based on RTT and $2G$, the minimum SFT based on the compressed RTO_{min} , and the normal SFT based on the observed RTO. Of these, SFT_{min} is the value that most closely approximates the worst-case SFT that a connection is likely to see in practice.

	Windows XP	Linux 2.4	FreeBSD 4.7	SunOS 5.8
RTO_{initial}	3	3	3	4
RTO_{min}	70ms	150ms	150ms	286ms
RTO_{norm}	220ms	332ms	1.2s	1.59s
RTO_{max}	120s	120s	64s	60s
R	5	7	12	8
G	10ms	10ms	10ms	10ms
RTT	0.39ms	0.43ms	0.9ms	28ms
RTO_{wc}	20.39ms	20.43ms	20.9ms	48ms
SFT_{wc}	0.61s	2.57s	85.56s	12.19s
SFT_{min}	2.1s	18.9s	614.1s	72.64s
SFT_{norm}	6.6s	41.832s	4912.8s	403.86s

Table 8-1: Results of TProbe and Compute SFTs

8.1 Windows XP

The observed results are confirmed by the Windows 2000 TCP implementation definition in [MB00]. Windows XP uses a TCP timestamp resolution of 100ms, but the compressibility of the RTO clearly indicates a finer-grained clock is actually used for timing purposes. An experiment that was not carried out was to see if the RTO could be compressed below 100ms when timestamp modifications are used instead of pre-acking.

Windows XP does not enable TCP timestamps by default, although it will properly echo timestamps. The tests were conducted with the Tcp1323Opts registry parameter set to 3, which enables both timestamps and the window scale option.

8.2 Linux 2.4.18

The minimum RTO for Linux and FreeBSD both required slightly larger delays between pre-acks which likely contributed to the larger value.

8.3 FreeBSD 4.7

FreeBSD had the largest default retransmit count of the systems investigated with correspondingly higher SFTs.

8.4 SunOS 5.8

It was not possible to determine from TProbe or from the traces gathered whether the results correspond to the target SunOS system or whether they are the result of intervening firewalls or proxies. Information about the remote configuration was not available in time for publication.

The clock granularity could not be obtained from the traces because the only timestamp supplied was in the initial SYN packet despite negotiation of the option. (This further suggests a firewall may be removing this common means of fingerprinting systems.)

SunOS limits the RTO only after it has exceeded the maximum; other systems limit the RTO when they

reach the maximum. For example, the retransmit times from one run in seconds were approximately:

4.7, 9.8, 19.1, 38.2, 119.9, 59.9, 59.9

Other implementations in the study do not let the RTO rise above its maximum.

8.5 Conclusions

The investigations into the TCP survivable failure time yield several discoveries. First, the time that a TCP connection can survive in a LAN is typically very small, on the order of 6-10 seconds. This can be verified using a tool such as netperf between two systems with small roundtrip times between them. Second, it is possible to shape TCP traffic using the techniques developed to prove RTO_{max}. This could be used in situations where a system owns one end of a connection and can schedule down time. It is not, however, practical for switches due to the need to buffer packets during injected delays. Nor is it appropriate during the normal operation of a TCP connection since the longer RTTs will reduce TCP's effective throughput.

The expectation that implementations would limit the RTO to about 200ms, a typical delayed acknowledgement timer, was not borne out by the observations. In fact, it was possible to compress the retransmission timeout to values as small as 70ms with survivable failure times just under a half a second.

The implementations examined do not appear to limit the lowest possible RTO. Since these are often well below possible delayed acknowledgement timers, it would appear that TCP would benefit from eliminating delayed acknowledgement on links with retransmission times less than 200ms.

This suggests two possible TCP modifications that may be of benefit:

1. Do not use delayed acknowledgements when receiving data on a connection whose SRTT is less than 200ms. This will avoid senders retransmitting prematurely.
2. Do not allow the RTO to drop below 200ms to avoid unnecessarily retransmitting data when delayed acknowledgements are used by the receiver.

Future work would study these two options for their effectiveness in reducing premature transmissions and impact on performance.

Source Code Availability

The source code for TProbe and the tcpdump traces for sample test runs are available at:

<http://www.ini.cmu.edu/~clord/Projects/SFT>

The code requires libpcap-0.7.1 which is available at <http://www.tcpdump.org>.

References

- [AP99] M. Allman and V. Paxson, "On Estimating End-to-End Network Path Properties," *Proceedings of ACM SIGCOMM*, Sept. 1999.
- [APS99] M. Allman, V. Paxson, and W. Stevens, "TCP Congestion Control," *RFC 2581*, April 1999.
- [Bra89] R. Braden, "Requirements for Internet Hosts - Communication Layers," *RFC 1122*, October 1989.
- [BSK95] H. Balakrishnan, S. Seshan, and R.H. Katz, "Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks," *ACM Wireless Networks*, 1(4), December 1995.
- [CSA00] N. Cardwell, S. Savage, and T. Anderson, "Modeling TCP Latency," *Proceedings of IEEE INFOCOM 2000*, Tel-Aviv, Israel, March 2000.
- [ETF02] Y. Etsion, D. Tsafirir, and D. G. Feitelson, "Effects of Clock Resolution on the Scheduling of Interactive and Soft Real-Time Processes," *Technical Report 2001-14 (Revised)*, School of Computer Science and Engineering, The Hebrew University of Jerusalem, Oct 2002.
- [Jac88] V. Jacobson, "Congestion Avoidance and Control," *Proceedings of ACM SIGCOMM*, September 1988, pp. 314-329.
- [JD02] H. Jiang, C. Dovrolis, "Passive Estimation of TCP Round-Trip Times," *ACM Computer Communications Review*, August 2002.
- [KP87] P. Karn and C. Partridge, "Improving Round-Trip Time Estimates in Reliable Transport Protocols," *ACM SIGCOMM*, August 1987, pp. 2-7.
- [MB00] D. MacDonald and W. Barkley, *Microsoft Windows 2000 TCP/IP Implementation Details White Paper*, 2000.
- [Mog95] J. Mogul, "The Case for Persistent-Connection HTTP," *ACM SIGCOMM*, 1995, pp. 299-313.
- [PA00] V. Paxson and M. Allman, "Computing TCP's Retransmission Timer," *RFC 2988*, November 2000.
- [PAD+99] V. Paxson, Mark Allman, S. Dawson, et al, "Known TCP Implementation Problems," *RFC 2525*, March 1999.
- [Pax97] V. Paxson, "End-to-end Internet Packet dynamics," *Proceedings of ACM SIGCOMM*, September 1997.
- [Pax98] V. Paxson, "On Calibrating Measurements of Packet Transit Times," *Proceedings of SIGMETRICS '98*, 1998.
- [PF01] J. Padhye and S. Floyd, "Identifying the TCP Behavior of Web Server," *ICSI TR-01-002*, February 2001.
- [Pos81] J. Postel. Transmission Control Protocol, *RFC 793*, September 1981.
- [Rus97] M. Russinovich, *Inside Windows NT High Resolution Timers*, <http://www.sysinternals.com/ntw2k/info/timer.shtml>, July 9, 1997.
- [Sav99] S. Savage, "Sting: a TCP-based Network Measurement Tool," *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, October 1999, pp. 71-79.
- [VCH02] F. Veysset, O. Courtay, O. Heen, "New Tool And Technique For Remote Operating System Fingerprinting," <http://www.intranode.com/pdf/techno/ring-full-paper.pdf>, *Intranode Research Paper*, April 2002.
- [ZPS00] Y. Zhang, V. Paxson, and S. Shenker, "The Stationarity of Internet Path Properties: Routing, Loss and Throughput," *ACIRI Technical report*, May 2000.

Author



Christopher is a graduate student in the Information Networking Institute at Carnegie Mellon University.

Before joining the Information Networking Institute in August 2001, Christopher was a software engineer specializing in networked and distributed systems. He contributed to the development of networked storage systems at Cereva, fault-tolerant systems at Marathon, Internet search engines at Alta Vista, and networking software for Windows NT at the former Digital Equipment Corporation. He received his Bachelor of Science in Computer Science from Fitchburg State College in 1988.

Christopher's research interests include survivable and emergent systems. He has joined David Fisher and his research team in the Software Engineering Institute to study unbounded systems and the design of emergent algorithms. His current work involves developing homeostatic models of the immune system and exploring the computational capabilities of such systems.