Improving Round-Trip Time Estimates in Reliable Transport Protocols

Phil Karn
Bell Communications Research, Incorporated

Craig Partridge
Harvard University/BBN Laboratories Incorporated

Abstract

As a reliable, end-to-end transport protocol, the ARPA Transmission Control Protocol (TCP) uses positive acknowledgements and retransmission to guarantee delivery. TCP implementations are expected to measure and adapt to changing network propagation delays so that its retransmission behavior balances user throughput and network efficiency. However, TCP suffers from a problem we call *retransmission ambiguity*: when an acknowledgment arrives for a segment that has been retransmitted, there is no indication which transmission is being acknowledged. Many existing TCP implementations do not handle this problem correctly.

This paper reviews the various approaches to retransmission and presents a novel and effective approach to the retransmission ambiguity problem.

1. Introduction

Dynamically estimating the round-trip time, the interval between the sending of a packet and the receipt of its acknowledgement, is a key function in many reliable transport protocols [5,15,22]. Such estimates are used to ensure that data is reliably delivered. If a packet remains unacknowledged for too long, it is assumed to have been lost and is retransmitted. Estimated round-trip times are used to determine when these retransmissions will occur.

Three developments in IP networking [19,20,21] have led to increased interest in the problems of estimating round-trip times.

First, there has been an explosive growth in the size and complexity of IP *internetworks*, built by interconnecting existing subnetworks. The best known example is the ARPA Internet. (The ARPANET is just one component subnetwork in the ARPA Internet.) The ARPA Internet has highly variable round-trip times. Because its paths are very complex, it also tends to lose more packets.

Second, there has been a large increase in traffic on some of the major IP networks. Higher traffic loads have led to serious network congestion on some parts of the ARPA Internet [16,18]. Like network size, congestion is known to cause highly variable round-trip times and higher packet loss rates.

Finally, recent research has shown that the standard approaches to estimating round-trip times for the Transmission Control Protocol (TCP) are inaccurate if packets are lost or round-trip times are highly variable [9,24]. This discovery is distressing because it suggests that the mechanism reliable protocols depend upon to handle loss and variable round-trip times, namely the estimation of round-trip times, may not work well.

Concern about the accuracy of estimated round-trip times has led to some interesting research into reliability mechanisms which are less dependent on round-trip estimates [2,24]. The authors, however, take a different approach that tries to improve the data used to compute round-trip time estimates. In this paper we present an analysis of this work.

2. The TCP Algorithm

TCP implementations attempt to predict future round-trip times by sampling the behavior of packets sent over a connection and averaging those samples into a ''smoothed'' round-trip time estimate, $SRTT$.

When a packet is sent over a TCP connection, the sender times how long it takes for it to be acknowledged, producing a sequence, S, of round-trip time samples: $s_1, s_2, s_3 ....$

With each new sample, s_i, the new SRTT is computed from the formula:

$$SRTT_{i+1} = (\alpha \times SRTT_i) + (1-\alpha) \times s_i$$

where $SRTT_i$ is the current estimate of the round-trip time, $SRTT_{i+1}$ is the new computed value, and α is a constant between 0 and 1 that controls how rapidly the SRTT adapts to change.

The retransmission time-out (RTO_i), the amount of time the sender will wait for a given packet to be acknowledged, is computed from $SRTT_i$. The formula is:

$$RTO_i = \beta \times SRTT_i$$

where β is a constant, greater than 1, chosen such that there is an acceptably small probability that the round-trip time for the packet will exceed RTO_i.

2.1. General Observations

There are several things to observe about the algorithm. First, it can be viewed as an attempt to approximate the next value from a function R, where $R(i)$ is the actual round-trip time for packet i. Given the sequence of measured round-trip times,

$$S = s_1, s_2, s_3, ..., s_{i-1}$$

which correspond to the values of R:

$$s_1 = R(1), s_2 = R(2), ..., s_{i-1} = R(i-1)$$

we hope that the RTO computed from those values will be a good upper bound on $R(i)$, the round-trip time for the next packet. Notice that if the measured round-trip times, S, are inaccurate then the RTO is probably incorrect; this problem is examined in the next section.

One should also observe that the values of the constants α and β have important effects on the behavior of the algorithm.

The value of α controls how rapidly the SRTT adjusts to changing round-trip times. Mills [11] has measured network round-trip times and recommends that there be two values for α, depending on the relative values of the sample, s_i, and $SRTT_i$. Mills observed that round-trip times are roughly Poisson distributed, but with brief periods of high delay. During these periods, he found that the standard way of computing SRTT and RTO often did not adapt swiftly enough, and the TCP sender would unnecessarily retransmit packets because the RTO was set too low. As a result, he suggested a non-linear filter where α is smaller when $SRTT_i < s_i$, allowing the SRTT to adapt more swiftly to sudden increases in network delay.

Choosing a value for β is harder because it has important and conflicting effects on individual user throughput and overall network efficiency [15]. To achieve optimal throughput β should be only a little greater than 1. This keeps the RTO very close to the SRTT and ensures that packet loss will be quickly detected. Detecting lost packets quickly is important for good throughput, since the end-to-end flow control mechanisms in reliable protocols like TCP will cause the sender to stop transmitting new packets if a packet remains unacknowledged for much longer than the round-trip time.

Unfortunately, what is good for throughput is disastrous for efficient network utilization. If the RTO is nearly equal to the SRTT (i.e., if β is near unity) then a large number of packets will be retransmitted unnecessarily because the sender times out too soon. For example, consider the situation where $RTO = SRTT$, (i.e, β=1), and the SRTT is an accurate median of the round-trip times. In this case, roughly half of all packets will be timed out and retransmitted because their acknowledgement took too long, burdening the network with unnecessary retransmissions. To minimize retransmissions, β

should be chosen such that the RTO will be a high upper limit on the round-trip times. The TCP specification [21] recommends a value of $\beta=2$ as a reasonable balance. Another possibility suggested by Van Jacobson [7,8] is to vary β based on the observed variance in measured round-trip times, although this is outside the immediate scope of this paper.

2.2. Back-off

Whenever a timeout occurs, virtually every TCP implementation increases the RTO by some factor before retransmitting the unacknowledged data. Should the new, larger RTO expire yet again before the retransmission is acknowledged, it is increased still further. This technique is known as *back-off*. (Back-off is performed independently of SRTT calculation, since without an acknowledgment there is no new timing information to be fed into the calculation). A variety of algorithms are used since the TCP specification does not prescribe one. Some (e.g., Berkeley UNIX[1]) step through a table of arbitrary back-off factors for each successive retransmission; others simply double the RTO (i.e., perform binary exponential back-off) for each consecutive attempt. Whatever the algorithm, TCP back-off is essential in keeping the network stable when sudden overloads cause packets to be dropped. When the overload condition disappears, packet loss stops and the TCPs reduce their RTO to their normal SRTT-based values.

3. Sampling Round-Trip Times

A key assumption of the TCP algorithm is that the sequence of round-trip samples is an accurate measurement of the true network round-trip times (i.e., that $s_1=R(1), s_2=R(2)$, etc.). It has recently been shown that the two standard sampling methods, measuring from the first transmission and measuring from the most recent transmission, give inaccurate results [9,24].

This inaccuracy is caused by packet retransmission. The information carried in the packet headers of TCP and most other reliable protocols does not indicate if an acknowledgement is in response to the original transmission of a packet or a retransmission. As a result, a round-trip time measurement for a retransmitted packet is ambiguous. We will call this problem *retransmission ambiguity*.

3.1. Measuring From the First Transmission

Many TCP implementations measure round-trip times from the first transmission of a packet. Whenever an acknowledgement is received, the round-trip time is computed from the first time the packet was sent, regardless of how many times the acknowledged packet has been retransmitted,

Sampling from the first transmission may cause the SRTT to grow without bound when there is loss on the network. When there is loss, the TCP sender must retransmit lost packets. If we look at the sequence of samples, S_F, we discover that it contains samples of two types. If γ_i is a boolean function which is 0 if the acknowledgement for packet i is acknowledging a retransmission, S_F can be expressed as:

$$S_F = \begin{cases} s_i & \text{if } \gamma_i \neq 0 \\ \overline{s_i} & \text{if } \gamma_i = 0 \end{cases}$$

If we look at the values of $\overline{s_i}$, those samples which are derived from the acknowledgements of retransmissions, we find that they are a function of the true round-trip time, $R(i)$, the SRTT, and the particular retransmission of packet i, r_i, where $r_i>0$, which is being acknowledged:

$$\overline{s_i} = R(i)+r_i \times RTO_i = R(i)+r_i \times \beta \times SRTT_i$$

$\overline{s_i}$ will be used to compute the new smoothed round-trip time, $SRTT_{i+1}$. Plugging $\overline{s_i}$ into the SRTT function gives:

$$SRTT_{i+1} = (\alpha \times SRTT_i)+(1-\alpha)\times(R(i)+r_i \times \beta \times SRTT_i)$$

[1] UNIX is a trademark of AT&T Bell Laboratories.

$$= (\alpha \times SRTT_i) + (1-\alpha) \times R(i) + (\beta r_i - \alpha\beta r_i) \times SRTT_i$$

Since $0<\alpha<1$ the factor $(\beta r_i - \alpha\beta r_i)$ is greater than zero and distorts the function, causing it to inflate the value of the SRTT. Inflated round-trip time estimates may not be a problem if the original reason for the high loss rate was network congestion, because congestion tends to increase round-trip times anyway. It is also acceptable if the loss rate is very low since the accumulated error is so small that it will probably have no noticeable effect on the SRTT. However, if the path is lossy (e.g., a noisy packet radio channel operating without link level acknowledgements), the SRTT grows and throughput unnecessarily decreases to low levels.

3.2. Measuring From the Most Recent Transmission

Another popular method measures round-trip time from the most recent transmission of a packet. The implicit assumption in this method is that the RTO is accurate; if a packet has to be retransmitted then previous transmissions have almost certainly been lost.

Unfortunately, this assumption is often false. If the RTO is smaller than the true round-trip time, acknowledgements for previous transmissions may arrive after a retransmission. If τ_i is a boolean function which returns 0 if the acknowledgement is for a previous transmission, the sequence of sampled values, S_R is:

$$S_R = \begin{cases} s_i & \text{if } \tau_i \neq 0 \\ \overline{s_i} & \text{if } \tau_i = 0 \end{cases}$$

At first glance this doesn't look too bad. $\overline{S_i}$ is a value between 0 and the RTO, which might be expected to distort the SRTT a bit, but doesn't have the growth term caused by measuring from the first transmission.

But the picture is not quite so rosy. Recall that the RTO is intended as an estimate of the maximum possible round-trip time. If an acknowledgement arrives after the RTO has expired, it is highly likely to come very shortly afterwards. In other words, instead of being randomly distributed between 0 and the RTO, $\overline{s_i}$ is likely to be very close to 0 (recall that the sample timer was reset when the RTO had elapsed). This will cause the SRTT to decline, reducing the RTO, and increasing the likelihood that a packet will be acknowledged just after the RTO has expired. The SRTT stabilizes at an unreasonably low estimate. Unnecessary data retransmissions occur constantly, useful throughput drops sharply, and network bandwidth is wasted.

Observe that the problem of a declining SRTT could be avoided if the RTO were set extremely high, so high that no packet could survive that long unacknowledged. Recall however, that for high throughput, the RTO cannot be much larger than the SRTT. An algorithm which requires an extremely high RTO will give unacceptable performance across a lossy path.

3.3. Ignoring Round-Trip Times for Packets That Have Been Retransmitted

Some implementations simply ignore round-trip time samples tainted by retransmission.

This method works, provided the true round-trip time never grows faster than the algorithm can adapt. If there is a sudden increase in network round-trip time (e.g., when the failure of a primary path causes packets to be sent via a slower secondary path), and if the new path delay becomes larger than the RTO, all samples will be discarded, because every packet will be retransmitted before the acknowledgement comes back. Note that if the RTO is reasonable (i.e., if β is chosen well) then the chance of a dramatic surge is quite small. But the consequences of such a surge are truly disastrous — the sender is stuck with an unrealistically short RTO that it has little chance or no chance of correcting. Once again, there are numerous unnecessary retransmissions, throughput drops sharply and network capacity is wasted.

3.4. Karn's Algorithm

Very recently a new sampling method has been suggested by one of the authors. This method addresses the problems with ignoring round-trip times of retransmitted packets.

The fundamental notion of Karn's algorithm is to use RTO back-off to collect accurate round-trip time measurements uncontaminated by retransmission ambiguity. The rule is as follows:

> When an acknowledgement arrives for a packet that has been sent more than once (i.e., retransmitted at least once), ignore any round-trip measurement based on this packet, thus avoiding the retransmission ambiguity problem. *In addition, the backed-off RTO for this packet is kept for the next packet. Only when it (or a succeeding packet) is acknowledged without an intervening retransmission will the RTO be recalculated from SRTT.*

The last provision ensures that new and accurate round trip measurements will be taken and fed into the SRTT estimate regardless of any sudden increase in round-trip delay. If the increase is large, the RTO may oscillate between the backed-off value necessary to avoid an unnecessary retransmission and the value calculated from SRTT. However, the SRTT will converge to the correct value, and unnecessary retransmission will stop.

How quickly the SRTT converges to the new round-trip time depends on the back-off algorithm and the SRTT smoothing algorithm, but typically this convergence is quite fast. To prevent unnecessary retransmissions, the RTO must be greater than the new round-trip time. To achieve this new RTO value the SRTT must be at least as large as the new round-trip time, s, divided by β. (For simplicity in the proof, we assume that the new value for s does not vary). Reaching the new RTO takes n valid samples, where n is the minimum value for which the following equality in terms of the new RTT, s, and the old SRTT, z, is true:

$$\frac{s}{\beta} \leq (z \times \alpha^n) + \sum_{i=1}^{n} ((s \times (1-\alpha)) \times \alpha^{(i-1)})$$

In the worst case $s - z$ is almost s (i.e., $s \gg z$), so the z term may be ignored. Dividing the remaining terms by s, we find that the upper limit on n is given by the solution to:

$$\frac{1}{\beta} \leq \sum_{i=1}^{n} (1-\alpha) \times \alpha^{(i-1)}$$

Using typical values of $\alpha = 0.875$ and $\beta = 2$, n is only 6. Since the number of required valid samples is small, convergence is usually swift.

A TCP implementation using Karn's algorithm and Mills' nonlinear filter has been in heavy use on perhaps the worst medium ever used to pass IP datagrams: amateur packet radio [10]. Despite packet loss rates often exceeding 50%, SRTT values remain quite stable, changing only in response to true changes in round-trip time. Packets lost due to noise leave the SRTT unaffected.

3.5. Sampling RTTs in Parallel

While Karn's algorithm is currently the best available solution to the sampling problem, it is worth taking a few paragraphs to discuss another class of sampling algorithms which have been developed recently. These algorithms depend on the fact that most transport protocols send more than one packet at a time, and as a result it is possible to take multiple round-trip time samples in parallel.

One such algorithm has recently been developed in an implementation of the Reliable Data Protocol (RDP), which uses the TCP algorithm to estimate round-trip times [4,17,22]. It relies on the fact that networks almost always preserve packet ordering. If two packets are sent close together it is likely that they will reach their destination in the order they were sent, and be acknowledged in the same order. So if an acknowledgement for packet i is received after the acknowledgement for packet j, where $j > i$, it is a strong hint that the acknowledgement is for a retransmission of packet i. We can check the retransmission count, r_i, for packet i, so this observation gives a sampling method.

Measuring from the first transmission, a sample for packet i, s_i, should be discarded if:

- An acknowledgement for some packet j, $j > i$, has already been received, *and*
- $r_i \neq 0$

The first test can only be applied if packets can be acknowledged when they are received out of order. Such acknowledgements are called *extended* or *selective acknowledgements*. RDP supports extended

acknowledgements. TCP does not, although consideration is being given to adding this facility [1].

The performance of this sampling method is highly dependent on the number of parallel transmissions a protocol implementation will support. Observations suggest that the algorithm keeps accurate round-trip estimates at much higher loss rates than sampling from the first transmission.

Other estimation methods that use multiple samples taken in parallel have been explored in Mills' work on synchronizing network clocks [12,13,14]. Synchronizing clocks involves problems of handling a certain amount of bad data, caused by noisy links or faulty clocks. The techniques used to eliminate such bad values can also be applied to the problem of extracting good round-trip times from a set of several round-trip times collected at roughly the same time.

Parallel sampling methods tend to suffer from two problems. First, they can be adversely affected by the loss of an entire group of packets; all the samples become bad. Second, they often fail when a network is congested. When a network is congested, most protocols attempt to reduce the data they put on the network by limiting themselves to sending one packet at a time. Unfortunately, a network is most likely to drop packets when it is congested. Thus the ability to take parallel samples is lost at precisely the time we would most like to have an accurate sampling method. One could blur the notion of ''parallel'' and simply apply these techniques after every n samples, where n is chosen to equal the number packets that are normally in flight. But when the protocol is in one-packet-at-a-time mode, recomputing the SRTT must be delayed until n samples have been collected, which could be a long time, at minimum it is roughly $n \times SRTT$.

4. The Perfect Sampling Method

It is worth noting for a moment that most of the methods discussed in this section are attempts to achieve the sampling function γ discussed in section 3.2. Recall that γ was a boolean function which returned 0 if the sample was taken from the acknowledgement of a retransmission. The sampling methods want to use only those samples measuring the time between the first transmission of a packet and the acknowledgement of that first transmission, i.e., those samples for which $\gamma \neq 0$. The problem is that most sampling methods are inadequate approximations of γ, and either exclude too many good samples or include too many bad samples.

Karn's algorithm solves this problem by accepting only good samples and using the retransmission back-off strategy to ensure that good samples will eventually be available even if round-trip times increase dramatically.

5. Deficiencies in the TCP Algorithm

So far this paper has focussed on how to improve round-trip estimates by using better sampling methods. Before concluding we would like to touch briefly on some other ways that round-trip estimates can be improved.

One improvement is to sample more frequently. Some protocol implementations sample round-trips only once per sending window, leading to poor estimates because the estimator does not have enough recent samples to detect changes in the round-trip time.

Another possible improvement is to chose a new algorithm for estimating the RTO. The TCP algorithm assumes that a weighted average, the SRTT, adjusted by some estimate of variance, β, is a good approximation of the behavior of the network function, R. Recently, research by Jacobson has shown that R is a more complex function than a simple average can accurately model [6,8]. Encouragingly, however, Jacobson's work also suggests that it may be possible to predict the values generated by R with functions of roughly the same complexity as the functions presented in section 2.

6. Conclusion

Much attention has recently been paid to the question of whether one can accurately sample round-trip times over a transport protocol connection. The authors have shown that round-trip times can be accurately sampled and have presented a simple method that gives good round-trip time samples.

7. Acknowledgements

The authors would like to thank Will Leland and Chase Cotton for their useful comments on this paper.

Bibliography

1. Braden, Robert T., Selective Acknowledgments in TCP. Draft *ARPANET Working Group Requests for Comments*.
2. Clark, David D., Lambert, Mark L., and Zhang, Lixia. NETBLT: A Bulk Data Transfer Protocol; RFC998. In *ARPANET Working Group Requests for Comments*, no. 998. SRI International, Menlo Park, Calif., March 1987.
3. Edge, Stephen William. An Adaptive Timeout Algorithm for Retransmission Across a Packet Switching Network. In *Proceedings of SIGCOMM '83*, Association for Computing Machinery, 1983.
4. Hinden, Robert M. and Partridge, Craig. Version 2 of the Reliable Data Protocol. Draft *ARPANET Working Group Requests for Comments*.
5. International Organization for Standards, Information processing systems — Open Systems Interconnection — *Connection oriented transport protocol specification.* International Standard, no. 8073. ISO, Switzerland. 1986.
6. Jacobson, Van. Presentation to the Internet End-To-End Services Task Force. April 16, 1987.
7. Jacobson, Van. *Interpacket Arrival Variance and Mean.* Letter to the TCP-IP mailing list, 15 June 1987.
8. Jacobson, Van. *Retransmit Timers: Theory and Practice*, working title of draft paper.
9. Jain, Raj, Divergence of Timeout Algorithms for Packet Retransmissions. In *Proceedings Fifth Annual International Phoenix Conference on Computers and Communications*, Scottsdale, AZ, March 26-28, 1986.
10. Karn, P. R., Price, H., Diersing, R. Packet Radio in the Amateur Service. In *IEEE Journal on Selected Areas in Communications*, May 1985.
11. Mills, David. Internet Delay Experiments; RFC889. In *ARPANET Working Group Requests for Comments*, no. 889. SRI International, Menlo Park, Calif., Dec. 1983.
12. Mills, David. Algorithms for Synchronizing Network Clocks; RFC956. In *ARPANET Working Group Requests for Comments*, no. 956. SRI International, Menlo Park, Calif., Sep. 1985.
13. Mills, David. Experiments in Network Clock Synchronization; RFC957. In *ARPANET Working Group Requests for Comments*, no. 957. SRI International, Menlo Park, Calif., Sep. 1985.
14. Mills, David. Network Time Protocol; RFC958. In *ARPANET Working Group Requests for Comments*, no. 958. SRI International, Menlo Park, Calif., Sep. 1985.
15. Morris, Robert J.T. Fixing timeout intervals for lost packet detection in computer communication networks. In *AFIPS Conference Proceedings*, 1979 National Computer Conference. AFIPS Press, Montvale, Jew Jersey.
16. Nagle, John. Congestion Control in IP/TCP Networks; RFC896. In *ARPANET Working Group Requests for Comments*, no. 896. SRI International, Menlo Park, Calif., Jan. 1984.
17. Partridge, Craig. Implementing the Reliable Data Protocol (RDP). In *Proceedings of the 1987 Summer USENIX Conference*, Phoenix, Arizona.
18. Perry, Dennis G. *Congestion in the ARPANET.* Letter to the TCP-IP Mailing List, October 1, 1986.
19. Postel, J., ed. Internet Protocol; RFC791. In *ARPANET Working Group Requests for Comments*, no. 791. SRI International, Menlo Park, Calif., Sep. 1981.
20. Postel, J., ed. Internet Control Message Protocol; RFC792. In *ARPANET Working Group Requests for Comments*, no. 792. SRI International, Menlo Park, Calif., Sep. 1981.

21. Postel, Jon, ed. Transmission Control Protocol; RFC793. In *ARPANET Working Group Requests for Comments*, no. 793. SRI International, Menlo Park, Calif., Sep. 1981.

22. Velten, David, Hinden, Robert and Sax, Jack. Reliable Data Protocol; RFC908. In *ARPANET Working Group Requests for Comments*, no. 908. SRI International, Menlo Park, Calif., July 1984.

23. Watson, Richard W. *Timer-Based Mechanisms in Reliable Transport Protocol Connection Management.* Computer Networks 1981, North-Holland Publishing Company.

24. Zhang, Lixia. Why TCP Timers Don't Work Well. In *Proceedings of SIGCOMM '86*, Association for Computing Machinery.