Improving Round-Trip Time Estimates in Reliable Transport Protocols

PHIL KARN
Bell Communications Research, Incorporated
and
CRAIG PARTRIDGE
Harvard University / Bolt Beranek and Newman, Inc.

As a reliable, end-to-end transport protocol, the Transmission Control Protocol (TCP) uses positive acknowledgements and retransmission to guarantee delivery. TCP implementations are expected to measure and adapt to changing round-trip delay so that their retransmission behavior balances user throughput and network efficiency. However, TCP suffers from a problem we call *retransmission ambiguity*: when an acknowledgement arrives for a datagram that has been retransmitted, there is no indication of which transmission is being acknowledged. As a result, an implementation may be unable to determine if the round-trip time it measures is for an original transmission or a retransmission of a datagram. Many existing TCP implementations do not handle this problem correctly. Furthermore, the problem of retransmission ambiguity is also a characteristic of other major transport protocols, including OSI TP4 and DECnet NSP

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

Categories and Subject Descriptors: C.2.0 **[Computer Communications Networks]:** General—*open System Interconnection reference model (OSI)*; C.2.1 **[Computer Communications Networks]:** Network Architecture and Design—*packet networks, store and forward networks*; D.4.4 **[Operating Systems]:** Communications Management—*message sending, network communication*

General Terms: Algorithms, Performance, Reliability

Additional Key Words and Phrases: Round-trip times, transport protocols

1. INTRODUCTION

A round-trip time is the interval between the sending of a datagram and the receipt of its acknowledgment. It implicitly measures both the network

This paper is a substantially revised version of a paper that appeared in the *Proceedings of ACM SIGCOMM '87*, pp. 2–7.
Authors' addresses: P. Karn, Bell Communications Research, 435 South St., Morristown, NJ 07960; C. Partridge, Bolt Beranek and Newman, Inc., 10 Moolton St., Cambridge, MA 02138.

propagation delay, including time spent in gateway queues, and any time spent at the receiver and sender processing the datagram and acknowledgment. In wide area networks, the propagation delay is the most significant contributor to the round-trip time, but even on local area networks, recent advances in reducing protocol processing times [3] have made the propagation delay an important part of the total round-trip time.

Dynamically estimating the round-trip time is a key function in many reliable transport protocols [4, 5, 6, 15, 20, 21, 22]. Round-trip time estimates are used to ensure that data is reliably delivered. If a datagram 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 [1, 18, 19, 20] 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 Federal Research Internet, an IP internetwork that contains over 375,000 hosts and several thousand constituent networks [8]. Using IP gateways, the Internet interconnects a wide variety of networks, with very different speeds, propagation delays, service intervals, and mean times between failure. As a result of this diversity, round-trip times over the Internet are often quite variable. Some networks on the Internet also have relatively high error rates and must discard a moderate number of datagrams that have been corrupted in transit.

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 Internet [16, 17]. Like network size, congestion is known to cause highly variable round-trip times. Congestion also causes datagram loss, as gateways discard datagrams when their queues overflow.

Finally, research has shown that the common approaches to estimating round-trip times for the Transmission Control Protocol (TCP) are inaccurate if datagrams are lost or round-trip times are highly variable [11, 23]. This discovery is distressing because it suggests that the mechanism that reliable transport protocols depend upon to handle loss and variable round-trip times, namely the estimation of round-trip times, may not work well. The problem of round-trip estimation also has important consequences for other reliable protocols, especially OSI TP4 [6] and DECnet NSP [4], which use round-trip time estimation schemes similar to that of TCP. (Actually, the TP4 specification uses a more primitive estimation scheme which is known to be inadequate for use over the OSI connection-less network service (CLNS) [7]. A proposal to adopt a round-trip time estimation algorithm like the one discussed below is currently working its way through the OSI approval process [13].)

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, 23]. The authors, however, have taken a different

approach which tries to improve the data used to compute round-trip estimates. In this paper we present an analysis of this work.

2. THE TCP ALGORITHM

According to the original TCP specification [20], TCP implementations are expected to predict future round-trip times by sampling the behavior of datagrams sent over a connection and averaging those samples into a "smoothed" round-trip time estimate, $SRTT$. (The specification has since been updated by Braden [1] to require the prediction algorithms described in this paper and by Jacobson [9]).

When a datagram 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: $s_1, s_2, s_3, \ldots$.

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 datagram 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 datagram 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 datagram i. Given the sequence of measured round-trip times,

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

which correspond to the values of R:

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

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 datagram. 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. In 1983, Mills [14] measured network round-trip times and

recommended 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, corresponding to intervals when intermediate gateway queues were overloaded. 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 datagrams because the RTO was set too low. As a result, he suggested a nonlinear 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 datagram loss will be quickly detected. Detecting lost datagrams 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 datagrams if a datagram 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 datagrams will be retransmitted unnecessarily because the sender times out too soon. For example, consider the situation where $RTO = SRTT$, (i.e., $\beta = 1$), and the SRTT is an accurate median of the round-trip times. In this case, roughly half of all datagrams will be timed out and retransmitted because their acknowledgment 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 original TCP specification [20] recommended a value of $\beta = 2$ as a reasonable balance.

More recently, Van Johnson has developed a refinement of the TCP algorithm that dynamically computes the variance instead of using a fixed β [9, 10]. This important refinement has been shown to produce dramatically improved round-trip time estimates and is now in wide-spread use. An examination of Jacobson's work is outside the scope of this paper. A detailed discussion can be found discussed by Jacobson [10].

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, the RTO 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 is used since the TCP specification does not prescribe one. Some (e.g., Berkeley UNIX™) step through a table of

™ UNIX is a trademark of AT&T Bell Laboratories.

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 datagrams to be dropped [10]. When the overload condition disappears, datagram 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.). However, it has been shown that the two obvious sampling methods, measuring from the first transmission and measuring from the most recent transmission, give inaccurate results [11, 23].

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

3.1 Measuring From the First Transmission

Many TCP implementations measure round-trip times from the first transmission of a datagram. Whenever an acknowledgment is received, the round-trip time is computed from the first time the datagram was sent, regardless of how many times the acknowledged datagram 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 datagrams. 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 returns 0 if the acknowledgment for datagram i is acknowledging a retransmission, and nonzero otherwise, 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 acknowledgments of retransmissions, we find that they are a function of the true round-trip time, $R(i)$, the SRTT, and the particular retransmission of datagram 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

$$\begin{aligned} SRTT_{i+1} &= (\alpha \times SRTT_i) + (1 - \alpha) \times (R(i) + r_i \times \beta \times SRTT_i) \\ &= (\alpha \times SRTT_i) + (1 - \alpha) \times R(i) + (\beta r_i - \alpha\beta r_i) \times SRTT_i. \end{aligned}$$

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 datagram radio channel operating without link level acknowledgments), the SRTT grows and throughput unnecessarily decreases to low levels [11, 23].

3.2 Measuring from the Most Recent Transmission

Another popular method measures round-trip time from the most recent transmission of a datagram. The implicit assumption in this method is that the RTO is accurate; if a datagram 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, acknowledgments for previous transmissions may arrive after a retransmission. If τ_i is a boolean function which returns 0 if the acknowledgment is for a previous transmission, and nonzero otherwise, 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 result 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.

Unfortunately, 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 acknowledgment 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 datagram 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 [11, 23].

Observe that the problem of a declining SRTT could be avoided if the RTO were set extremely high, so high that no datagram 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 Datagrams 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 datagrams to be sent via a slower secondary path), and if the new path delay becomes larger than the RTO, then all samples will be discarded. Every datagram will be retransmitted before the acknowledgment comes back. Note that if the RTO is reasonable (i.e., if β is chosen well) then the chance of network round-trip time suddenly exceeding the RTO is small. But the consequences of the round-trip time exceeding the RTO a situation are truly disastrous—the sender is stuck with an unrealistically small RTO that 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

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 datagrams.

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

The last provision ensures that new 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 example, 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 inequality in terms of the new RTT, s, and the old SRTT, z, is true

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

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 [12]. Despite datagram loss rates often exceeding 50%, SRTT values remain quite stable, changing only in response to true changes in round-trip time. Datagrams lost due to noise leave the SRTT unaffected. Similar results on other networks have been achieved using Karn's algorithm and Jacobson's improved round-trip time estimation algorithm [10].

3.5 Implementing Karn's Algorithm

In practice, Karn's algorithm can be implemented in a handful of instructions. We illustrate the key steps here for a TCP that only has one datagram outstanding at a time. (In practice, TCP implementations have multiple outstanding datagrams at any given time, but showing this parallelism would unnecessarily complicate the illustration of Karn's algorithm).

In any TCP implementation, for each TCP connection, one must track the round-trip time, the smoothed round-trip time, the current round-trip timeout and the number of retransmissions of the current datagram. Call these variables, **rtt, srtt, rto,** and **rxmit,** respectively. We keep these variables in a TCP connection block, **tcb**. To this group of variables, Karn's algorithm adds an additional variable, the value to which the round-trip timeout should be set, **rto_set**. Using these variables, we can illustrate an implementation of Karn's algorithm using the *C* programming language.

When transmitting a TCP datagram, instead of setting the round-trip timeout to a function of **srtt,** we will set it to **rto_set**:

```
/* set retransmission timer and clear retransmission count */
tcb → rto = tcb → rto_set;
tcb → rxmit = 0
```

Now we modify the TCP acknowledgment routine to only update **srtt** and **rto_set,** if there has been no retransmission. This change is the key in Karn's algorithm. Thus the code to update the round-trip time might look as follows:

```
/* see if the acknowledged datagram being timed has been transmitted */
if (tcb → rxmit = = 0) {
  /*
   *no, not retransmitted, so update srtt and rto_set
   *rtt contains the measured round-trip time
   */
  tcb → srtt = new_srtt(tcb → rtt, tcb → srtt);
  tcb → rto_set = rto(tcb → srtt)
}
```

Where the function *new_srtt* updates the estimated round-trip time based on the observed round-trip time, and *rto* computes the new round-trip timeout based on the new estimated round-trip time. (In practice, to save the overhead cost of a procedure call, these two functions are typically done in-line. They are separate functions in this example to clearly distinguish them from the code to implement Karn's algorithm). Finally, if there is a retransmission timeout, we exponentially backoff both **rto** and **rto_set**:

```
/*
 *retransmission timeout: backoff rto, and record new backed off rto in
 *rto_set for possible use by next datagram
 */
tcb → rxmit + + ;
tcb → rto = backoff(tcb → rto, tcb → rxmit);
tcb → rto_set = tcb → rto
```

The function, *backoff*, performs the exponential backoff of the round-trip timeout. Observe that because **rxmit** is no longer set, **rto_set** will not be recomputed when the datagram is acknowledged and the backed-off value of **rto_set** will be used to set the round-trip timeout for the next datagram.

4. THE PERFECT SAMPLING METHOD

It is worth noting for a moment that most of the methods discussed in Section 3 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 acknowledgment of a retransmission. The sampling methods want to use only those samples measuring the time between the first transmission of a datagram and the acknowledgment 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 include too many bad samples or admit the possibility of excluding all samples, good and bad.

Karn's algorithm is a good approximation of the perfect function, γ, because it only accepts good samples and uses the retransmission back-off strategy to ensure that good samples will eventually be available even if round-trip times increase dramatically.

5. CONCLUSION

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

ACKNOWLEDGMENT

The authors would like to thank Will Leland and Chase Cotton as well as our anonymous TOCS reviewers for their useful comments on this paper.

REFERENCES

1. BRADEN, R. (Ed.). Requirements for Internet hosts—Communication layers. In *Internet Requests for Comments*, no. 1122, SRI International, Menlo Park, Calif., Oct. 1989.

2. CLARK, D. D., ZHANG, L., AND LAMBERT, M. NETBLT: A high throughput transport protocol. In *Proceedings of the ACM SIGCOMM '87* (Stowe, Vt., Aug. 1987). ACM, New York, 1987, 353-359.
3. CLARK, D. D., JACOBSON, V., ROMKEY, J., AND SALWEN, H. An Analysis of TCP processing overhead. *IEEE Commun. 27*, 6 (July 1989), 23-29.
4. Digital Equipment Corporation. *DECnet Digital Network Architecture, Phase IV; NSP Functional Specification*, Tech. Rep., Dec. 1983.
5. EDGE, S. W. An adaptive timeout algorithm for retransmission across a packet switching network. In *Proceedings of the ACM SIGCOMM '84*. ACM, New York, 1984, 248-255.
6. International Organization for Standards. Information processing systems—Open systems interconnection. In *Connection Oriented Transport Protocol Specification*. International Standard 8073. ISO, Switzerland, 1986.
7. International Organization for Standards. Information processing systems—Open systems interconnection—*Protocol for Providing the Connectionless-Mode Network Service*. International Standard 8473. ISO, Switzerland, 1988.
8. Internet Engineering Task Force. In *Proceedings*, 1990, Tech. Rep., National Research Initiatives, and personal communication from M. K. Lottor.
9. JACOBSON, V. Interpacket arrival variance and mean. Letter to the TCP-IP mailing list, June 15, 1987.
10. JACOBSON, V. Congestion avoidance and control. In *Proceedings of the ACM SIGCOMM '88* (Stanford, Calif., Aug. 1988), ACM, New York, 1988, 314-329.
11. JAIN, R. Divergence of timeout algorithms for packet retransmissions. In *Proceedings of the Fifth Annual International Phoenix Conference on Computers and Communications* (Scottsdale, Ariz, Mar. 1986), 174-179
12. KARN, P. R., PRICE, H., DIERSING, R. Packet radio in the amateur service *IEEE J. Select Areas Commun. 12*, 4 (May 1985).
13. MANKINS, A. Private communication, July 1989.
14. MILLS, D. Internet delay experiments; RFC889. *Internet Requests for Comments*, no. 889. SRI International, Menlo Park, Calif., Dec. 1983.
15. MORRIS, R. J. T. Fixing timeout intervals for lost packet detection in computer communications networks. *AFIPS Conference Proceedings*. 1979 National Computer Conference. AFIPS Press, Montvale, N.J., 1979, 887-891.
16. NAGLE, J. Congestion control in IP/TCP networks *ACM Comput. Commun. Rev. 14*, 4 (Oct. 1984), 11-17.
17. PERRY, D. G. *Congestion in the ARPANET*. Letter to the TCP-IP mailing list, Oct. 1, 1986.
18. POSTEL, J. (Ed). *ARPANET Working Group Requests for Comments*. No. 791. Internet protocol; RFC791. SRI International, Menlo Park, Calif., Sept. 1981
19. POSTEL, J. (Ed). *Internet Requests for Comments* No. 792. Internet control message protocol; RFC792. SRI International, Menlo Park, Calif., Sept. 1981
20. POSTEL, J. (Ed). *Internet Requests for Comments*. No. 793. Transmission control protocol; RFC793. SRI International, Menlo Park, Calif., Sept. 1981.
21. VELTEN, D., HINDEN, R., SAX, J. *Internet Requests for Comments*. No. 908 Reliable data protocol; RFC908. SRI International, Menlo Park, Calif., July 1984.
22. WATSON, R. W. Timer-based mechanisms in reliable transport protocol connection management. In *Computer Networks*, North-Holland, Amsterdam, 1981, 47-56.
23. ZHANG, L. Why TCP timers don't work well. In *Proceedings ACM SIGCOMM '86* (Aug. 1986), 397-405.

Received July 1988; revised July 1991; accepted July 1991