Using Self-Clocking for Passive Estimation of TCP Round-Trip Times

by

Bryan Veal

(Under the direction of Kang Li)

Abstract

Passive methods of measurement allow the examination of network properties without altering the flow of data, which could change the network properties being measured. Passive methods also allow a choice between online and offline measurement using data traces. We propose a method to passively estimate round-trip times, an important network performance metric, for TCP data flows. The method works by observing repeating patterns of segment clusters caused by TCP's self-clocking mechanism. Estimates need not be taken at the end-points; they can be taken anywhere along the route between the sender and receiver. Also, since only one direction of flow is necessary, estimates can be taken when network routes are asymmetric. Furthermore, estimates can be taken throughout the lifetime of the transfer, allowing observation of changes in round-trip times. We evaluate this method using traces from the Internet and also a testbed with emulated network conditions.

Index words: Networks, Internet, TCP, Passive measurement, Round-trip time, Self-clocking, Asymmetric routes

Using Self-Clocking for Passive Estimation of TCP Round-Trip Times

by

Bryan Veal

B.S., The University of Georgia, 2003

A Thesis Submitted to the Graduate Faculty
of The University of Georgia in Partial Fulfillment
of the
Requirements for the Degree

Master of Science

Athens, Georgia

2005

Using Self-Clocking for Passive Estimation of TCP Round-Trip Times

by

Bryan Veal

Approved:

Major Professor: Kang Li

Committee: David K. Lowenthal
Eileen Kraemer
Hongwei Wu

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
August 2005

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

TCP is the dominant form of traffic on the Internet. The World Wide Web, email, FTP, peer-to-peer downloads, and many other applications use TCP to communicate. Round-trip time (RTT) is an important metric in determining the behavior of a TCP flow. In terms of TCP, RTT is the amount of time between when a data segment is sent and when acknowledgment of that data arrives at back at the sender. With knowledge of the RTT, it is possible to measure the congestion window size and retransmission timeout of a connection, as well as the available bandwidth on its path [8]. This information can help determine factors that limit data flow rates and cause congestion [15]. When known at a network link along a path, RTT can also aid efficient queue management and buffer provisioning. Additionally, RTT can be used to improve node distribution in peer-to-peer and overlay networks [12].

We are interested in determining RTTs *passively*. Passive estimation allows analysis of both current and historical data. It only requires that traces of the flows be captured, which can be done without disturbing the network flow. This avoids changing the nature of the flows being measured, and it also avoids violating any network standards that prohibit changing data in transit.

We introduce a method to passively measure RTTs at any point on the network path between the endpoints of a TCP session. Our method utilizes the existing *self-clocking* mechanism inherent to TCP. During a bulk data transfer, self-clocking causes the pattern of packet arrivals to cycle once per RTT. This means that the arrival times of packets during one RTT

time interval are similar to those in the next. We use a pattern detecting algorithm, *auto-correlation*, to determine the period of the self-clocking pattern, which becomes our RTT estimate.

This method makes several contributions compared to prior techniques.

- It only requires that packets in *one* direction, as opposed to both directions in other approaches, pass through the measurement point. This is important because asymmetric routes are common on the Internet.

- It returns RTT samples over the lifetime of the connection, rather than solely relying on the predictable patterns of the three-way handshake and slow-start phases that begin a TCP connection.

- It detects problems that make RTT estimation less reliable, such as packet arrival patterns caused by saturation of network bottlenecks and noise that obscures the self-clocking pattern caused by RTT variation. It filters these estimates when necessary.

To present our new method, we first describe existing methods to passively estimate RTT in Chapter 2. In Chapter 3, we describe how to turn self-clocking patterns into RTT estimates, and we describe some challenges posed by this method, such as loss, variation, and link saturation. Chapter 4 proposes solutions to these challenges, and in Chapter 5 we evaluate our solutions with experiments using both a testbed with emulated network conditions as well as traces taken from the Internet. Finally, we present our conclusions in Chapter 6.

Chapter 2

Related Work

Other methods to passively estimate RTT exist. One works with asymmetric routes, but only during the initial phase of the connection. The rest can make estimates throughout the TCP session, but they require symmetric routes as well as other conditions. In this chapter we give a brief overview of these methods.

2.1 SYN-ACK and Slow-Start Methods

One method of passive RTT estimation, like our self-clocking method, works with asymmetric routes, but it does not work throughout the lifetime of the session [10]. It has two variants: one uses uses segment association during TCP's initial three-way handshake (known as the *SYN-ACK method*), and the other associates segments during the slow-start phase (known as the *slow-start method*) by taking advantage of the fact that the number of data segments arriving each RTT can be easily predicted. However, during the congestion avoidance phase, it is hard to predict the RTT based on counting segments. Our method continues to generate estimates during the congestion avoidance phase and thus the entire data transfer.

Figure 2.1 illustrates how the SYN-ACK method works. When the measurement point is listening to segments coming from the client, it can measure the difference between arrival times of the SYN and ACK segments. However, if only the segments from the server are available, it can measure the time difference between the SYN/ACK segment and the next data segment (or ACK segment, depending on the protocol).

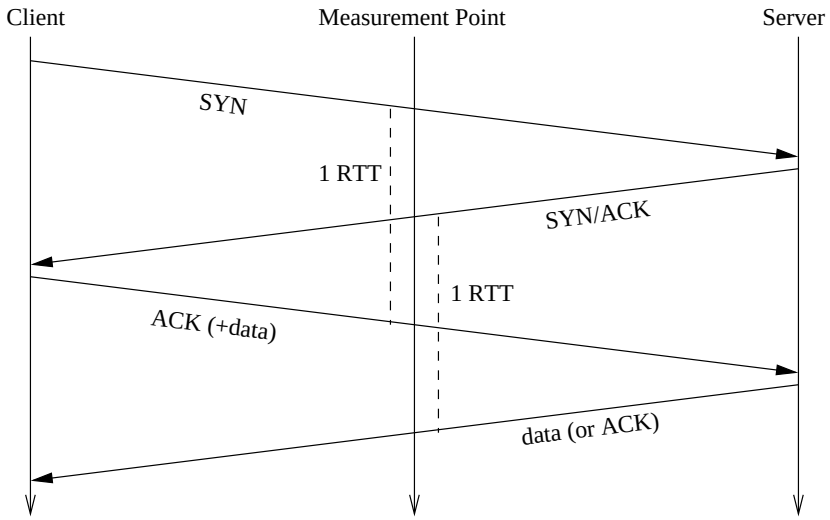

Figure 2.1: RTT estimation using the SYN-ACK method.

2.2 ACK-Data Correspondence Method

There is a method that does generate RTT estimates throughout the lifetime of a session, including the congestion avoidance phase [11]. However, unlike our method, this method requires symmetric routes. It works by creating a three-way association between a data packet, its ACK, and the the next data packet triggered by the arrival of the ACK at the sender. While the first association can be done with sequence numbers, associating the ACK with the next data segment is more difficult since sequence numbers do not increase for ACKs. The method then uses maximum-likelihood estimation to choose from possible sequences.

2.3 TCP Timestamp-Based Method

A simpler method works by leveraging the TCP Timestamps Option [7] to make the data-ACK-data association [13]. Unlike sequence numbers, TCP timestamps increase over time for ACKs. The next data packet carries the timestamp echo of the ACK that triggered it,

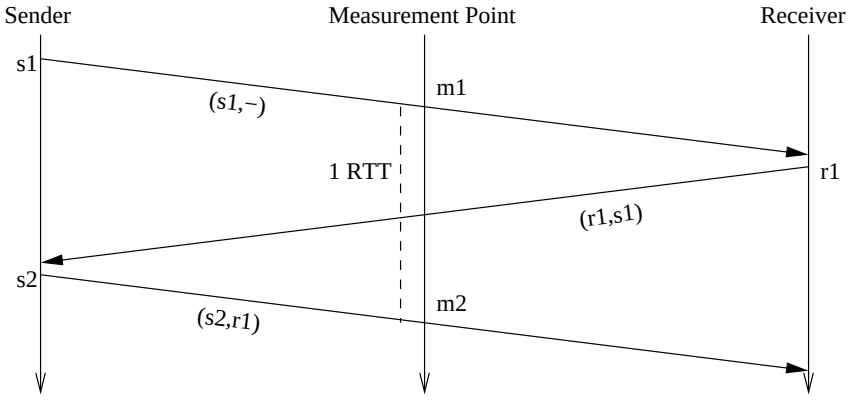

Figure 2.2: RTT estimation using the timestamp-based method.

completing the association. Like the previous method, this only works when the route is symmetric. In our evaluation, we will use this method to compare to ours when estimates from the sender's TCP implementation are unavailable.

Figure 2.2 illustrates this timestamp-based method. The sender transmits a segment at time *s1*. It arrives at the interior measurement point at time *m1*. The receiver responds with an ACK at time *r1* and echoes the sender's timestamp, *s1*. The measurement point recognizes *s1* in both segments and makes an association. Upon receiving the ACK, the sender transmits more data at time *s2* and echoes the receiver's timestamp, *r1*. The measurement point receives this segment at time *m2*. It recognizes *r1* in both segments and forms an association. Having associated all three segments, the measurement point estimates the RTT to be $m2 - m1$.

2.4 TCP State Method

Another method passively measures RTT by mimicking changes in the sender's TCP state, particularly its congestion window size [9]. The measurement point must accurately predict the type of congestion control used: Tahoe [5], Reno [6], or NewReno [4]. The accuracy of the estimate is affected by packet loss, TCP's window scaling option, and buggy TCP

implementations. Like the previous two methods, this method only works for symmetric routes.

Chapter 3

Using Self-Clocking to Infer RTT

Our algorithm observes self-clocking patterns in TCP flows and generates RTT estimates. We first describe the causes of self clocking and the nature of its patterns, and then we describe how to transform this pattern into an RTT estimate. Afterward, we present a number of situations that may interfere with these patterns such as link saturation, variation in delay, packet loss, and harmonic patterns at multiples of the true RTT.

3.1 TCP Self-Clocking

During a data transfer between TCP hosts, data segments are enqueued at the sender side and sent immediately whenever there is room on the network as allowed by the congestion window. The sender considers more room to be available when an acknowledgment segment (ACK) arrives, and thus new data is released. Similarly, when data segments arrive at the receiver, ACKs are immediately sent back.

Since there is little delay in responding to incoming data, the spacing between arrival times of data is largely preserved in the responding ACKs. Likewise, the time between ACKs is preserved in the following data segments from the sender. This assumes the sender always has data available to send, which is normally true for a file download. When these ACKs arrive at the sender one RTT later, more data is sent with a pattern similar to the previous pattern.

Figure 3.1 shows the number of data segments arriving at the measurement point every millisecond for a 0.5s interval for a TCP data flow with a 60ms RTT. A pattern of segment

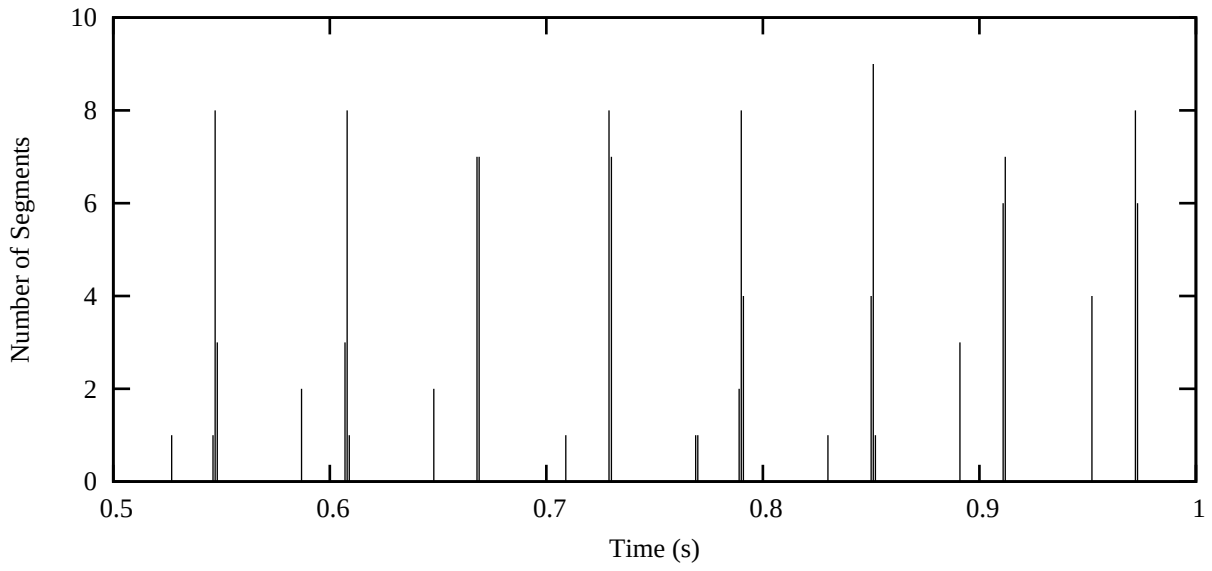

Figure 3.1: Number of segments arriving per millisecond for a 0.5s interval.

bursts repeats with nearly the same size and spacing once per RTT. While the pattern changes over time, it varies little from one RTT to the next.

3.1.1 RTT Estimation

Discrete autocorrelation is a mathematical formula that provides a means to compare a pattern in a data set to a previous pattern in the same data set. If we offset the data with itself at successively different amounts, computing autocorrelation reveals which offsets have stronger similarities than others. The amount of offset is called the *lag*. Figure 3.2 shows a plot of the autocorrelation strengths for the data in Figure 3.1. It shows a strong autocorrelation at the lag of 60ms, which corresponds to the RTT.

Our algorithm uses autocorrelation to detect patterns in segment arrival times and thus makes RTT estimates. The algorithm repeats the RTT estimation once per *measurement interval*, T, which is supplied as a parameter. During this interval, the number of packets that arrive at each time t is stored in an array, P. We call the unit used for t the *time*

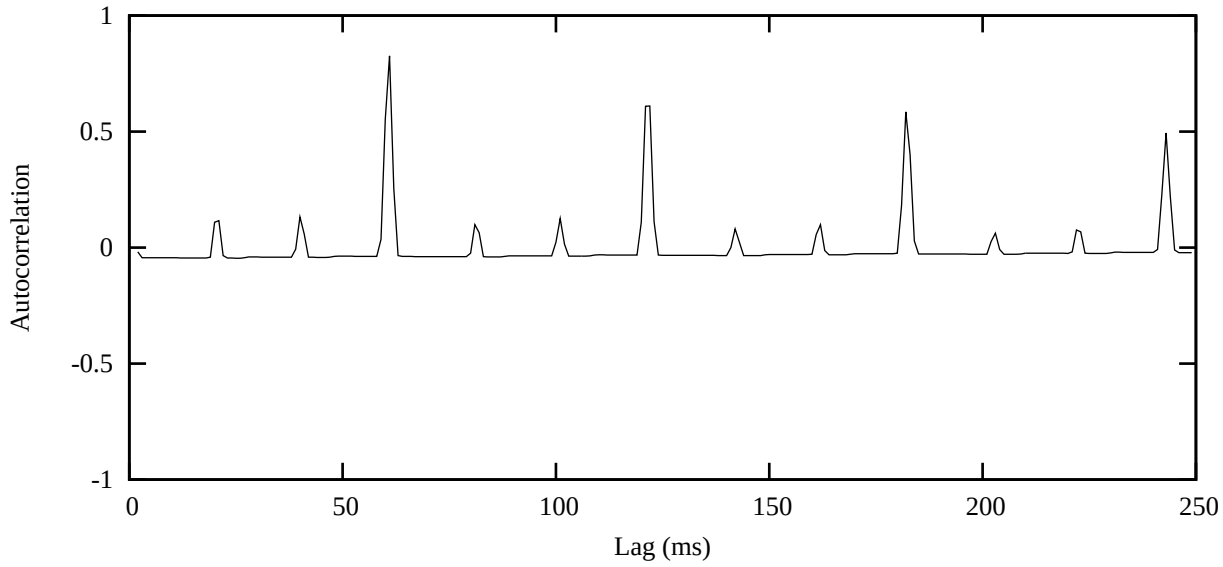

Figure 3.2: Autocorrelation plot for the data in Figure 3.1.

granularity. If a 1ms time granularity is used, P would contain the number of segments arriving each millisecond. Once the segment counts are complete, the discrete autocorrelation A_l is computed over P for each lag l from 1 to $T/2$ using the following formula:

$$A_l = \frac{\sum_{i=1}^{T-l}(P_t - \bar{P})(P_{t+l} - \bar{P})}{\sum_{i=1}^{T}(P_i - \bar{P})^2}$$

where $\bar{P}$ is the mean value of the elements of P.

The RTT estimate is computed as the lag with the highest autocorrelation level. This process is repeated to produce multiple estimates throughout the session. The number of estimates depends upon the size of the measurement interval and the overall duration of the session.

3.2 Challenges

3.2.1 RTT Range

The size of the measurement interval places an upper bound on the RTT estimate. The autocorrelation computation needs at least two full RTTs of data to make a complete comparison. Thus, a measurement interval of duration T can produce estimates up to $T/2$. This presents a tradeoff; a larger measurement interval allows estimation of larger RTTs, but increases the number of segments required to generate an RTT estimate.

Similarly, there is a lower limit on the RTT estimate at twice the time granularity. For instance, if the number of segments arriving every millisecond is used, no RTTs less than 2ms can be detected. A finer granularity would allow smaller RTT estimates, but it would increase the computation time.

3.2.2 Saturation

One problem that can disrupt a self-clocking pattern arises when a network link becomes saturated. That is, the arrival rate at that link is large enough that there are always packets enqueued, and the slow link cannot dequeue them fast enough to empty its buffer. This has the effect of removing the gaps from the self-clocking pattern while producing a new pattern where packets leave the link evenly spaced at a rate determined by the bandwidth of the bottleneck.

While no RTT estimate can be derived from this pattern, we have developed a filter to detect and remove estimates that exhibit signs of saturation. This filter is described in Section 4.1.

3.2.3 Delay Variation

On networks with large amounts of competing traffic, congestion, or other interference, variations in the amount of delay from one segment to the next could potentially disrupt the

self-clocking pattern. If this noise is great enough, the self-clocking pattern could be harder to detect. If the algorithm simply picks the lag with the highest autocorrelation level, it will likely make the wrong choice.

We have determined that autocorrelation levels themselves provide a means to determine whether a measurement interval is too noisy. Like the case with saturation, we can filter estimates from noisy intervals. This filter is described later in Section 4.2.

3.2.4 Loss

Another potential disruption to the self-clocking pattern is packet loss. During a measurement interval, some segment clusters can have missing segments due to loss, while other clusters can have additional segments due to retransmission. This causes a significant change in the pattern from one RTT to the next. In Section 4.3, we explain why loss actually has a limited impact on our ability to estimate RTT.

3.2.5 Harmonic Frequencies

The fact that the self-clocking pattern repeats more than once also presents a challenge. If a pattern is strong at an interval of one RTT, then it is also strong at twice the RTT, three times the RTT, and so on. The autocorrelation plot in Figure 3.2 illustrates this pattern with a peaks at 120, 180, and 240ms. Our autocorrelation function has a built-in correction mechanism to correct for these harmonic patterns, which we describe in Section 4.4.

Chapter 4

Improving RTT Inference

The previous chapter presented a number of challenges to using self-clocking to derive round-trip times. Some of these challenges require us to differentiate whether a measurement interval can produce an accurate estimate at all, and if it cannot, the estimate gets filtered out. In the case of variation, we present a method to discard intervals where the signal is lost in the noise, while we use a heuristic algorithm to check for saturation. For challenges such as data loss and harmonic patterns, we describe how the autocorrelation function helps to overcome them.

4.1 Saturation

As we described previously, saturation can replace the pattern caused by self-clocking with its own pattern. Fortunately, this type of pattern can be distinguished from a self-clocking pattern so that faulty estimates are avoided. We modified our algorithm with a filter to detect these patterns and discard the estimates.

The insight behind this filter is that saturation spreads segments out, so that each burst contains very few segments. Segments are likely to arrive singly when the capture point is after the bottleneck, or in pairs before the bottleneck because of delayed ACKs from the receiver. If the observed pattern were at the actual RTT, the number of segments should be larger during the congestion control phase of the session.

For a given measurement interval, let P be the array of segments arriving per millisecond and let A be the autocorrelation plot where A_l is the autocorrelation for the lag l. Let the

candidate RTT estimate, r, be the lag with the highest autocorrelation level. The filter works as follows:

1. Let $s = \sum_{i=0}^{r-1} P_i$.
2. If $s > 7$, then do not discard.
3. Else, for $m \in \{1, 2\}$, if $\sum_{i=rm}^{r(m+1)-1} P_i < s - 1$ or $> s + 1$, then do not discard.
4. Otherwise, discard this estimate.

First, we count the number of segments arriving from zero up to the first candidate RTT. Instead of accepting only RTT candidates with more than two segments, we only allow more than seven, since our RTT candidate may be at a harmonic frequency of the actual pattern. In our experiments, no more than seven segments occurred per RTT candidate for saturated traces in nearly all cases. If the estimate has not been accepted so far, the filter counts the segments in the next two candidate RTT intervals and ensure that the counts are within one segment of the first interval. This prevents discarding an interval during slow-start where the number of segments per RTT is likely to start small and increase.

We will evaluate this filter in Section 5.5.1 using traces that saturate an emulated bottleneck link.

4.2 Delay Variation

Another potential way to upset the self-clocking pattern is variation in the delay from one segment to the next. This can create a noise effect in the segment arrival times. For a noiseless autocorrelation plot, the peak corresponding to the RTT stands out. Aside from the harmonic multiples of the RTT, the rest of the autocorrelation levels are noticeably smaller than that of the RTT. For a trace with high variation, this peak shrinks and the rest of the autocorrelation levels are scattered, leaving the peak lost in the noise. To simply pick the lag corresponding to the highest peak in this situation is to risk picking some peak that

was actually caused by the noise. To mitigate this risk, we present a new filter to discard untrusted estimates.

For our filter to get a sense of the noise level for a measurement interval, it first calculates the mean, m, of the autocorrelation levels at each lag. In this calculation, the negative values are considered zero. Allowing negative numbers would cause the mean to always be nearly zero. We expect m to increase with respect to the peak lag r when the variation is high. For an autocorrelation plot A and the peak lag r, the filter determines whether A_r is sufficiently greater than m.

To control the level of filtering, we introduce a threshold level, t, as a parameter. If $A_r(1-t) < m$, then the estimate for this interval is discarded. Thus, t ranges from 0%, where no estimates are filtered, to 100%, where all are filtered. The threshold creates a tradeoff. When t is too high, correct estimates may be filtered along with the noise. Otherwise, when t is too low, incorrect estimates may be trusted.

4.3 Loss

While packet loss may affect the self-clocking pattern, we do not expect changes to be significant enough from one RTT to the next to greatly affect the autocorrelation computation. There is a potential impact if loss rates are large enough to cause the connection to stall. In this case, several RTTs may contain no segments at all. However, with the autocorrelation calculation, as few as two RTTs worth of data is enough to make an estimate. As a consequence, larger measurement intervals are less likely to have problems due to stalled flows. In section 5.1, we will demonstrate how loss affects the autocorrelation calculation using measurement intervals from lossy traces.

4.4 Harmonic Frequencies

One other potential difficulty is that a pattern at a multiple of the RTT has a higher autocorrelation level than the RTT itself. We chose an autocorrelation function where the maximum

level automatically decreases as the lag increases. This means that the lag for the RTT should have a higher autocorrelation than the harmonic multiples at larger lags. It is still possible to have a better match at some multiple of the RTT than for the RTT itself, but this should be rare. Since the self-clocking pattern tends to change gradually, patterns at a distance of one RTT are likely to be more similar than patterns at multiple RTTs.

Chapter 5

Experiments

To evaluate our RTT estimation method and our solutions to challenges posed in previous chapters, we provide an implementation, called the *Self-Clocking Passive Estimation Tool*, or *SCpet*. In our implementation, we use a default measurement interval of 500ms for most experiments. This allows RTT estimates up to 250ms, which is sufficient for most TCP flows. We also implemented the variation and saturation filters described in the previous chapter so that we may evaluate them by experiment. Most of our experiments use a filter threshold (Section 4.2) of 90%, which gives a moderate tradeoff between filtering out untrusted estimates while avoiding accidental filtering of correct estimates.

Our evaluation consists of a series of experiments using a testbed network with emulated conditions followed by tests using Internet traces. For the experiments using the testbed, we set up an network composed of four machines: a TCP data sender, the sender's router, the receiver's router, and the receiving TCP host. We set up a static route between these machines so that any packet sent from one end host to the other would pass through all the machines. We used NIST Net [3] on the sender's router to emulate delay, variation, loss, bandwidth restrictions, and saturation. We used the receiver's router as our measurement point and captured traces there with `tcpdump`. To generate traces, we used `ttcp`, which generates a simple TCP bulk data transfer from the sender to the receiver.

For comparison, we used RTT estimates taken directly from the sender's TCP implementation, which are produced by the machine's kernel through the use of the TCP Timestamps Option [7]. We modified `ttcp` to report RTT estimates from the Linux `tcp_info` structure after every `send` call. TCP computes its estimates as a moving average, so short-term changes

have a small effect. Also, our sender's TCP implementation tends to start with a high RTT estimate, so it may take up to a second for the estimate to converge to the actual RTT. Thus, average estimates tend to be slightly higher than the baseline RTT for a particular trace.

5.1 Loss

In Section 4.3 we stated that packet loss should have little impact on the RTT estimate. As shown in Figure 5.1, we experimented with seven traces, using NIST Net to set a base delay of 60ms and random loss rates ranging from 0 to 2%. With the exception of the 2% loss rate, there is no variation in the estimates. In the 2% case, only 5 out of the 177 total estimates were taken at harmonic multiples of the actual RTT. These were caused by the rare cases where random loss causes the number of packets per cluster to align more closely at two or three times the RTT rather than at the RTT itself.

To show the effects of loss more closely, the top of Figure 5.2 shows the number of packets that arrive per millisecond for three evenly spaced, nonconsecutive 500ms measurement intervals taken from the trace with 2% loss and a base RTT of 60ms. The bottom of the figure shows the corresponding autocorrelation plot for each interval. The self-clocking differs for each interval and it even changes gradually within a measurement interval. The rightmost interval shown has a stall that lasts for about four RTTs. One potential consequence of such stalls is that if only a few segments arrive during the measurement interval, the gap between the stalled segments may produce an erroneous RTT estimate. Despite these challenges, the autocorrelation plots produce 61ms RTT estimates for each measurement interval shown.

5.2 Variation

Here, we evaluate SCpet under two types of variation. First, we look at how random variation on a sub-RTT scale affects our ability to make accurate estimates. Next, we study the reaction of SCpet when the RTT changes entirely due to a route change.

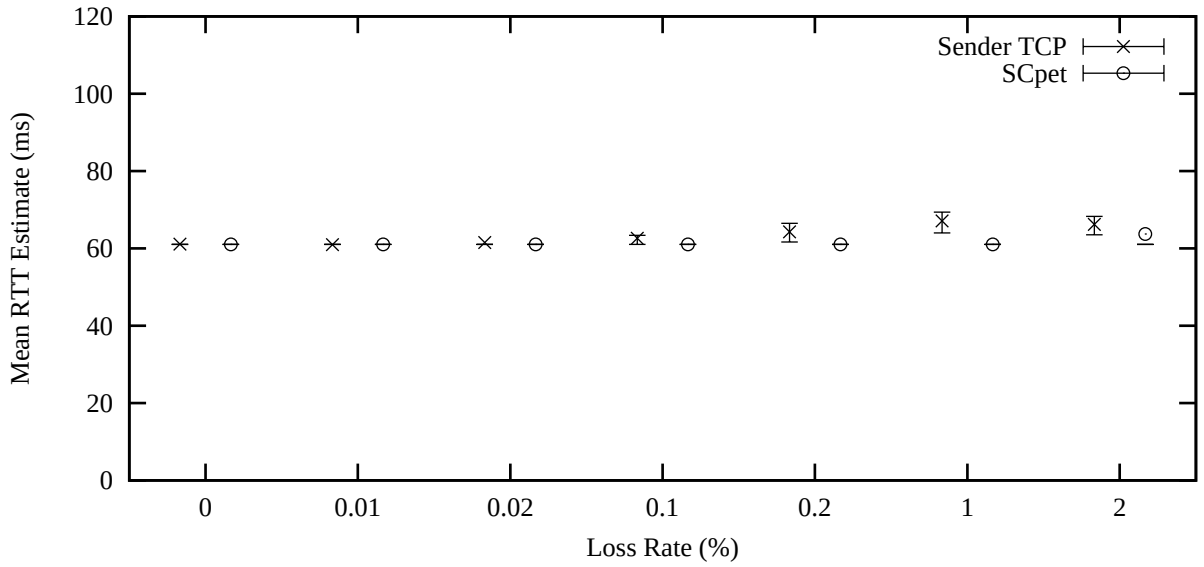

Figure 5.1: A comparison of RTT estimates from SCpet and the sender's TCP implementation with different levels of random packet loss. Each pair of points represents the mean RTT estimates for a trace with a base delay of 60ms and the indicated loss rate. Error bars indicate the first and third quartiles of the RTT estimates.

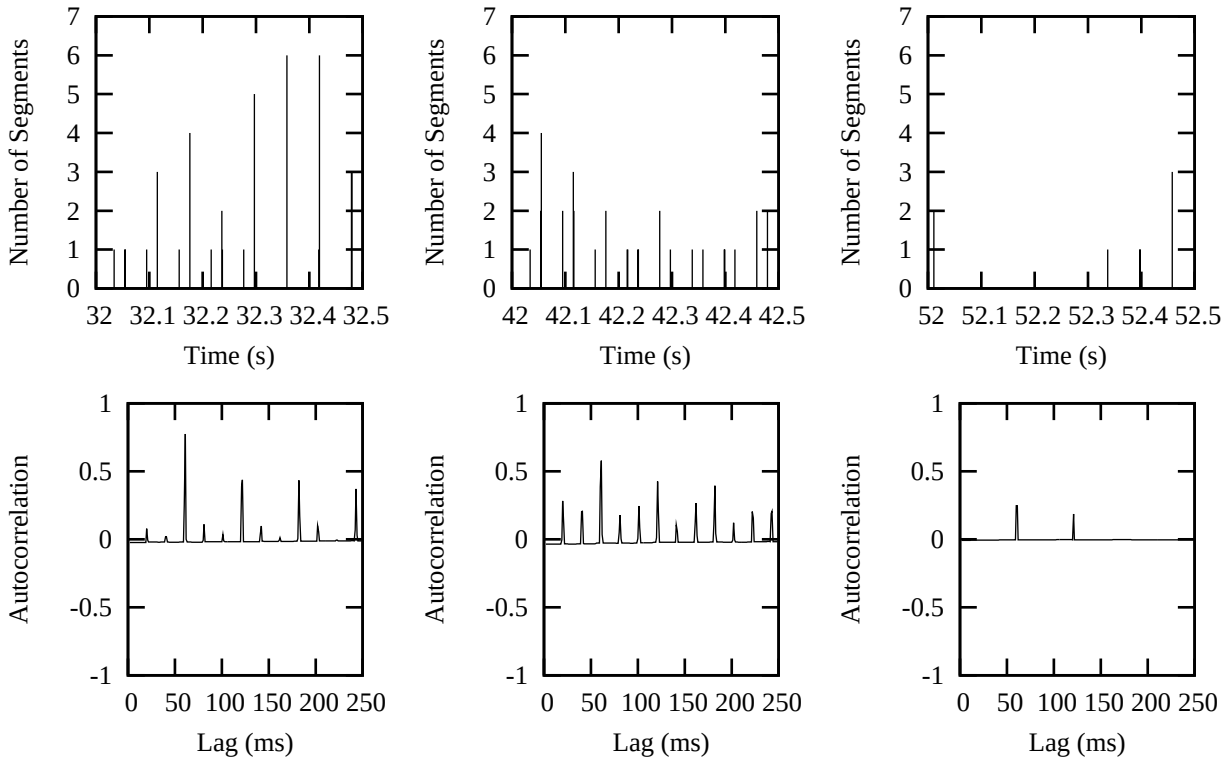

Figure 5.2: RTT estimation for traces with packet loss. Top: Three nonconsecutive measurement intervals of a trace with with a 60ms base delay and a 2% loss rate. Bottom: Autocorrelation plots for each interval showing the peak lags near 60ms.

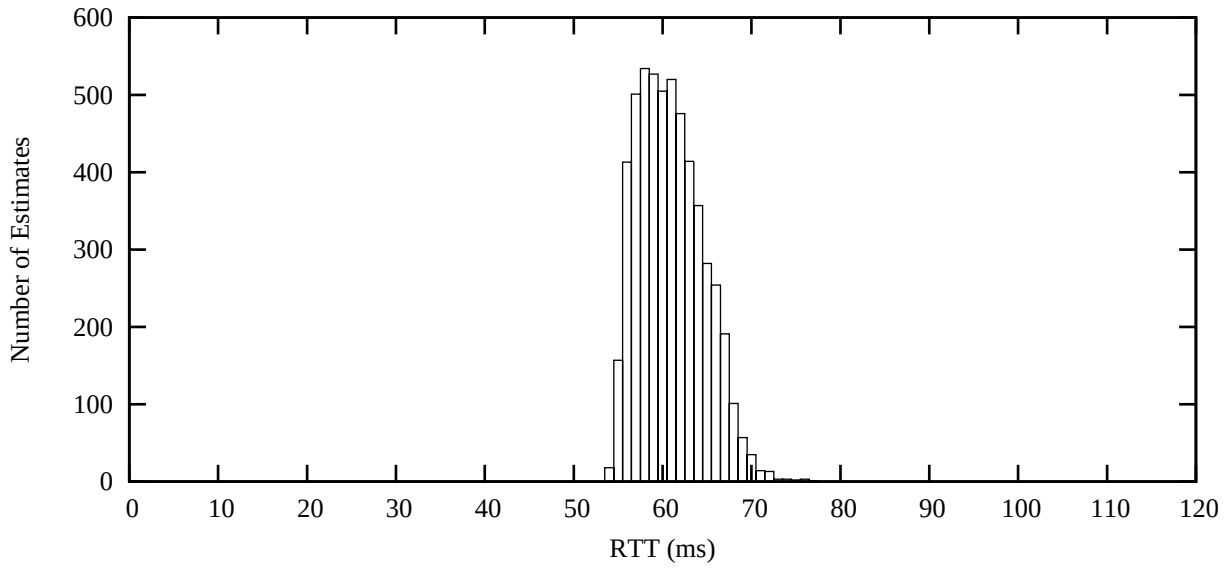

Figure 5.3: The distribution of RTTs for a trace with random variation of the delay. Estimates were calculated as the time differences between data segments and their ACKs on the sender. The base delay is 60ms, and the RTTs have a standard deviation of 5ms and cover a range of 23ms.

5.2.1 Random Variation

In Section 4.2, we described that if enough variation in the delay exists during a measurement interval, it could disrupt the self-clocking pattern so that a trustworthy estimate cannot be made. We presented an algorithm to filter these estimates based on whether the highest autocorrelation level stood significantly above average. We have implemented this algorithm as a part of SCpet.

To study the effects of variation, we used NIST Net to induce random changes to the RTT. With NIST Net, the amount of variation is specified as a standard deviation. Delays are randomly picked from a distribution with the given standard deviation and mean. Figure 5.3 shows the distribution of RTTs for a trace with a base delay of 60ms. While the standard deviation is set to 5ms, the total range of RTTs for the trace is actually much higher.

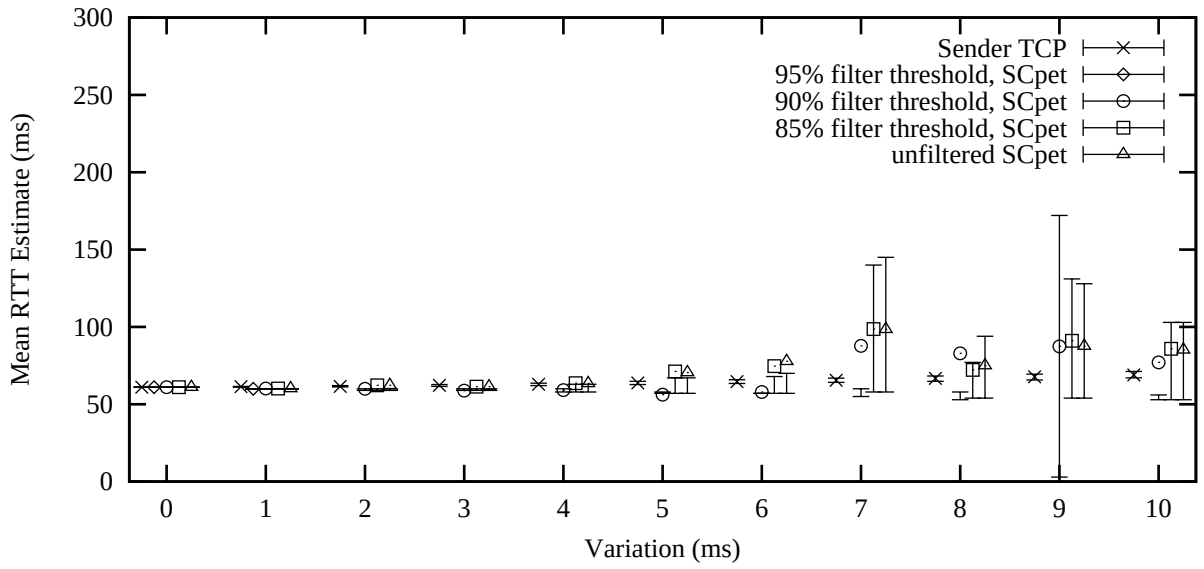

Figure 5.4: A comparison of mean RTT estimates from SCpet and the sender's TCP implementation for traces with random variation. Each point represents the mean RTT estimate for a trace with a 60ms base delay and the indicated variation. The leftmost point represents the estimate from the sender's TCP implementation, the next three points represent SCpet with filtering of untrusted estimates, and the rightmost point represents SCpet without any filtering. Error bars indicate the first and third quartiles of the RTT estimates. Variation levels are shown as the standard deviation of individual packet delays.

For our experiment, we captured traces with a base delay of 60ms and variation having standard deviations ranging from 0 to 10ms. To illustrate the tradeoff presented by the choice of filter threshold, we tested three different levels: a strict 95%, a moderate 90%, and a lax 85%. Figure 5.4 shows the average RTT estimates and their deviations for each trace at each filter level, as well as the the average estimate with no filtering at all. It also shows the average RTT estimate from the sender's TCP implementation as a reference.

We see that for strict filtering, variations of 0 and 1ms produce results near the sender's estimate. However, for larger variations, all estimates are filtered. A lower threshold of 90% shows correct results up to a 6ms deviation, which is an improvement over SCpet with no filtering. However, at deviations of 7ms and higher, estimates pass the filter with error

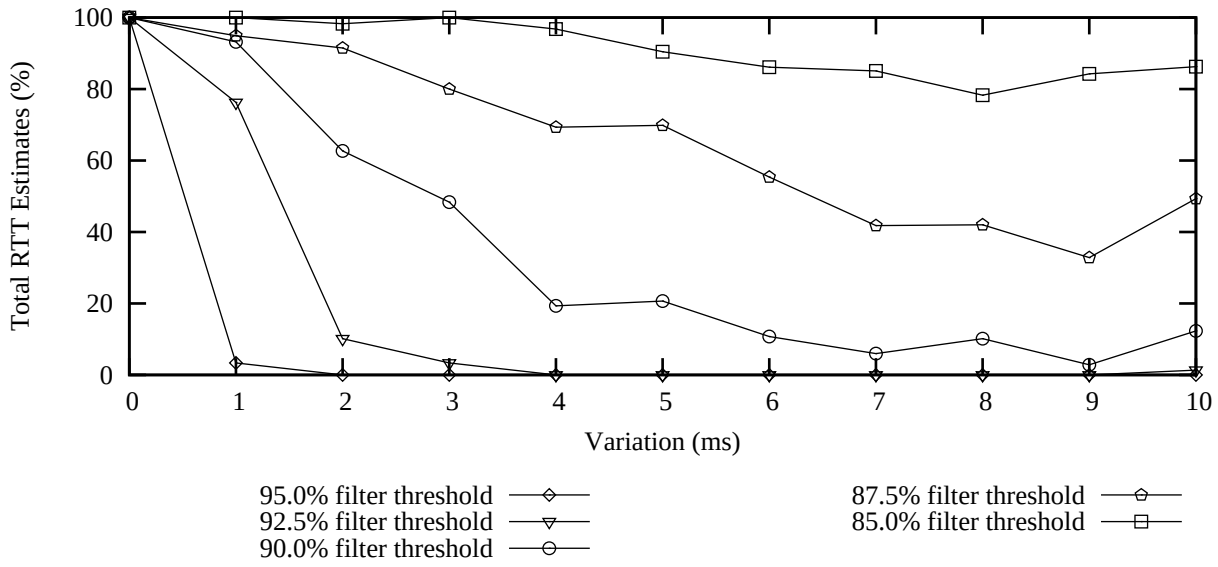

Figure 5.5: The percent of RTT estimates accepted by different thresholds for filtering untrusted estimates.

amounts similar to those of the unfiltered estimates. As Figure 5.5 shows, the 90% filter threshold succeeds at filtering out at least 86% of the untrusted estimates at each variation level of 7ms and above. The error levels of the lax filter threshold of 85% is indistinguishable from the estimates with no filtering. Figure 5.5 reveals that this is because very few estimates are discarded at this threshold.

An ideal result is a filter that, like the 90% filter threshold, produces accurate results until there is too much variation to produce any accurate estimates. With larger amounts of variation where no estimates are reliable, the ideal result would be like the 95% threshold, where all the estimates are filtered for the trace. We speculate that most traces over the Internet may fall into the category that the variation is small enough to filter accurately, while larger variations are uncommon. A study of a representative sample of TCP data flows over the Internet would reveal whether this is the case. We leave this study for future work.

5.2.2 Route Changes

A different kind of variation can occur on a larger scale when the route between sender and receiver changes. This can lead to a sudden jump from one RTT to another. Since the ability to detect changes in RTT over time sets our method apart from previous works, we test SCpet to verify that it can adjust to shifts in RTT.

One concern is that a shift between RTTs where one is a multiple of the other can leave behind a strong pattern of packet clusters that may leave a high autocorrelation level at the old RTT. Thus, we tested both harmonic RTT changes between 30ms and 60ms, and a non-harmonic RTT changes between 40ms and 70ms.

Figure 5.6 shows the RTT estimates made for series of measurement intervals in which the RTT changes. The points for SCpet are shown at the end of the measurement interval in which the estimate was made. For the measurement interval which contains the RTT change, the algorithm may choose either the beginning or ending RTT, depending on which is dominant. The following measurement intervals produce estimates close to those reported by the sender's TCP implementation, as we expect.

Another concern with route changes is that the accuracy of the estimate may depend up on the portion of the measurement interval in which the RTT change occurs. We used the 30ms-to-60ms and 40ms-to-70ms traces discussed previously to evaluate SCpet over a range of offsets of the measurement interval with respect to the RTT change. We took a 500ms measurement interval and shifted it across the route change in increments of 10ms. We started with the RTT change just *after* the measurement interval ends and shifted until the RTT change occurred just *before* the measurement interval begins. For each increment, we took an RTT estimate using SCpet.

For the trace with the 30ms-to-60ms RTT change, the RTT estimates change from 31ms to 61ms at an offset of 110ms. That is, at the point where 390ms of the measurement interval has a 30ms RTT and 110ms of the interval has a 60ms RTT, autocorrelation reveals a dominant pattern for the 60ms interval. The 60ms RTT dominates quickly because the

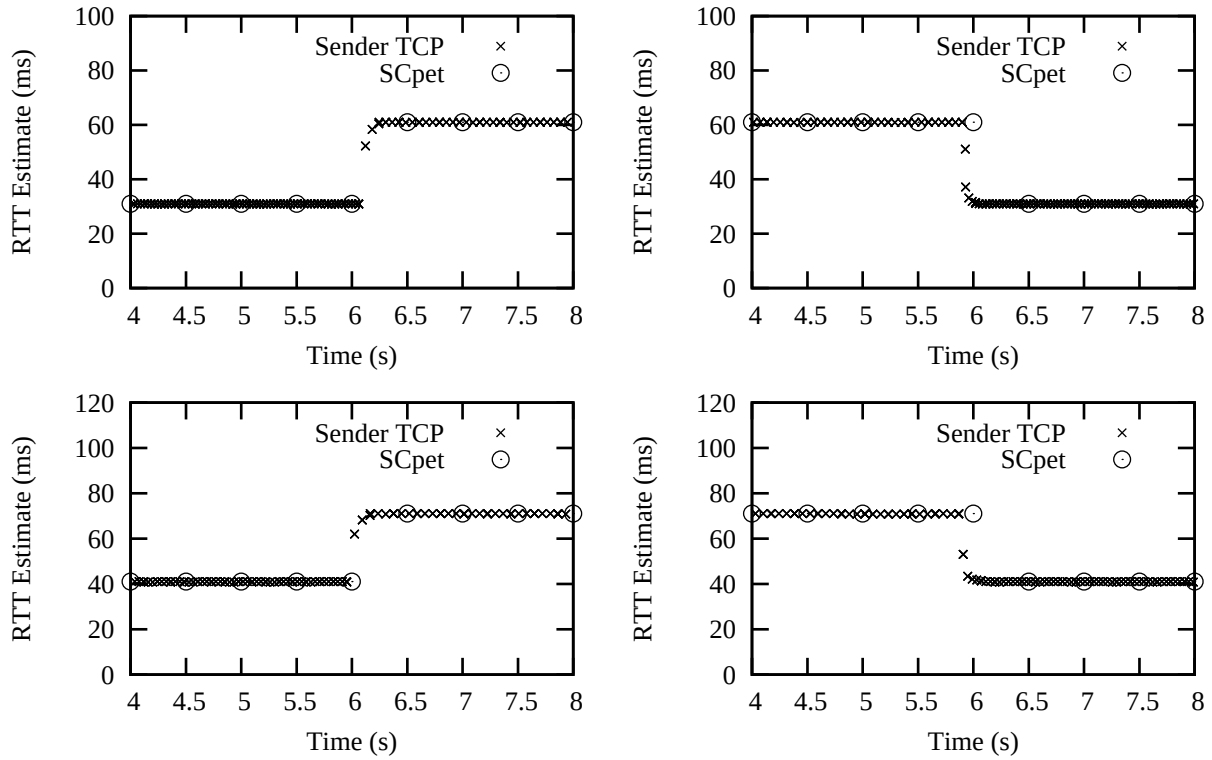

Figure 5.6: A comparison of RTT estimates from SCpet and the sender's TCP implementation during an emulated route change. Top: Harmonic changes between base delays of 30ms and 60ms. Bottom: Changes between base delays of 40ms and 70ms.

harmonic multiple at the 60ms lag already produces a strong autocorrelation for the 30ms RTT.

For the trace with the 40ms-to-60ms RTT change, a 41ms pattern is dominant until the measurement interval reaches an offset of 290ms. At this point, SCpet produces incorrect estimates of 2ms from the 300ms offset to the 330ms offset. For the offsets from 340ms to 500ms, a 71ms pattern dominates. For the region that produces 2ms estimates, since the measurement interval is split across two RTTs, both RTT patterns produce weak autocorrelations. During this portion of the trace, pairs of clusters arrive at the measurement point with a 1ms gap between them. These clusters exist during both the 40ms and 70ms portions of the measurement interval, so the 2ms pattern is not weakened by the fact that the measurement interval is split between two RTTs. Normally, the self-clocking pattern of the RTT would dominate the pattern caused by cluster pairs, but since both RTT patterns are weak, the 2ms pattern dominates.

5.3 Time Granularity

One possible means of compensating for sub-RTT variation is to use a coarser time granularity. For example, we could count the number of segments arriving at the measurement point per 5ms or 10ms, rather than 1ms as we have been using. The insight is that, since variation scatters clusters of segments, a 5ms or 10ms granularity could increase the size of the target time interval so that these segments would fall together into the needed cluster. This cluster would produce a more prominent peak in the autocorrelation plot and allow us to get the RTT, but with a larger error, since the granularity of potential RTT estimates becomes more coarse.

While we have seen this behavior in some cases, our experiments show that coarser time granularities can also produce repeating clusters of segments at the wrong RTT. Figure 5.7 illustrates this problem. The top two graphs show a measurement interval and its autocorrelation plot for a trace with a an RTT having a mean of 60ms and a standard deviation

of 10ms. The autocorrelation plot shows no distinct pattern. Moving down the figure to the 5ms and 10ms time granularities, crowded areas of small clusters converge into single large clusters. These crowded areas are caused by random variation and are not indicative of the self-clocking pattern. In the 10ms case, these crowded areas become clusters at a 160ms interval, which produces a peak autocorrelation at 160ms, though this is not the RTT. Problems such as this one prevent coarser granularities from producing more accurate RTT estimates than finer granularities.

5.4 RTT Range

The choice of measurement interval introduces an important tradeoff. On one hand, a larger interval allows a larger range of possible RTT estimates. The maximum RTT estimate is half the measurement interval since autocorrelation needs at least two full RTTs in a measurement interval to compare one to the other. A larger measurement interval can also increase the accuracy of an estimate because of the increased sample size. On the other hand, decreasing the interval allows for more fine-grained estimates and is thus more responsive to changes in RTT than larger intervals. Small measurement intervals also diminish the likelihood of erroneously choosing harmonic multiples of the RTT since these multiples would lie beyond the interval's range.

We expect that as long as RTTs are within half the measurement interval, SCpet should produce correct estimates. To test this expectation, we ran traces with seven different delays induced by NIST Net, ranging from 3.75ms to 240ms. We added no variation to the network as these traces were captured. The results are shown in Table 5.1.

For delays 30ms and larger, SCpet produces the expected result when measurement intervals are larger than twice the RTT. Smaller delays are getting caught by the variation filter, described in Section 4.2. These estimates are filtered because the mean autocorrelation level is elevated for small RTTs. Since the distances between harmonic multiples are small,

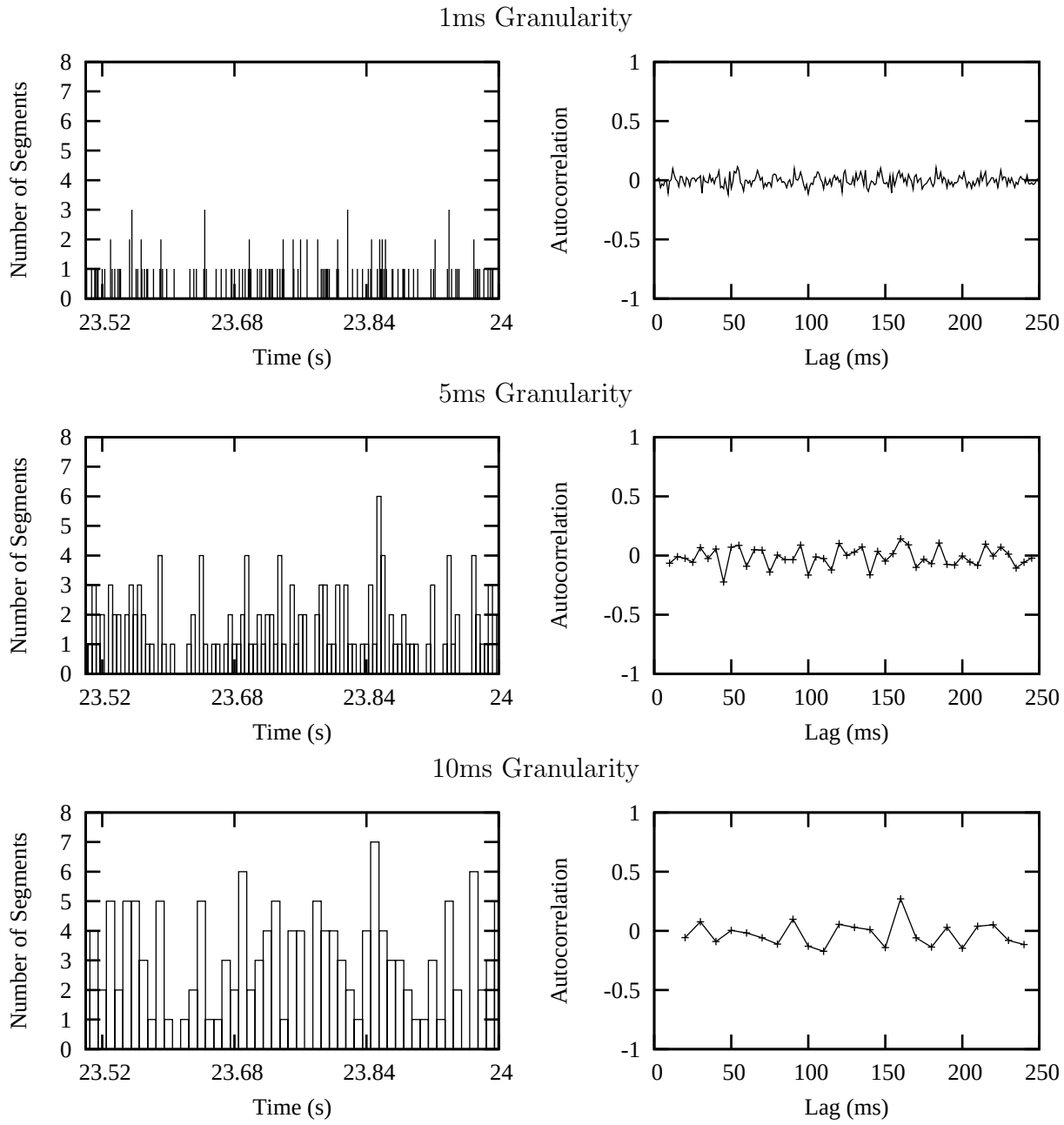

Figure 5.7: A non-RTT autocorrelation pattern created by coarse time granularity. Measurement intervals and autocorrelation plots are shown for the same trace using 1, 5, and 10ms time granularities. The trace has a base delay of 60ms and a variation of delay with a standard deviation of 10ms. As granularity becomes coarse, a peak emerges at a lag of 160ms.

Table 5.1: A comparison of RTT estimates from SCpet and the sender's TCP implementation for traces with different base delays and measurement intervals. The column "Base" shows the base RTT, "TCP" shows the mean estimate from the sender, "Es." shows the mean SCpet estimate, "Smp." shows the number of samples in the trace, and "Unf." shows the percent of samples accepted by the filtering algorithm. A dash ("–") indicates that all estimates were filtered.

		Measurement Interval														
		10ms			50ms			100ms			500ms			1s		
Base	TCP	Es.	Smp.	Unf.	Es.	Smp.	Unf.	Es.	Smp.	Unf.	Es.	Smp.	Unf.	Es.	Smp.	Unf.
3.75	4	–	220	0%	–	44	0%	–	22	0%	–	4	0%	–	2	0%
7.5	8	–	398	0%	–	79	0%	–	39	0%	–	7	0%	–	3	0%
15	16	–	775	0%	16	155	3%	16	77	1%	–	15	0%	–	7	0%
30	31	–	1514	0%	2	303	26%	31	151	100%	31	30	100%	31	15	100%
60	61	–	2991	0%	13	599	23%	31	299	87%	61	59	100%	61	29	100%
120	121	–	5929	0%	2	1186	4%	33	593	26%	121	118	100%	121	59	100%
240	241	–	11589	0%	2	2365	0%	39	1183	40%	240	236	100%	241	118	100%

there are a lot of peak autocorrelation levels and this inflates the mean value. Thus, the variation filter errs toward strict filtering at small RTTs.

For RTTs greater than half the measurement interval, sometimes the variation filter removes these estimates, but often the wrong RTT is reported. When multiple segment clusters per measurement interval exist, sub-patterns show up in the autocorrelation plot. When the dominant pattern from the RTT is missing because it is out of range, these sub patterns are all that remains. They can be easily mistaken for the RTT. Thus, it important to choose an measurement interval sufficiently large to handle the range of expected RTTs.

5.5 Bandwidth

Unlike our experiments so far, real networks contain bottleneck links that restrict bandwidth. Thus, this section studies the effects of bandwidth restrictions on SCpet's RTT estimates. First we take a look at saturated networks where the self-clocking pattern is replaced. Next, we study how the self-clocking pattern holds up under a range of common bandwidths.

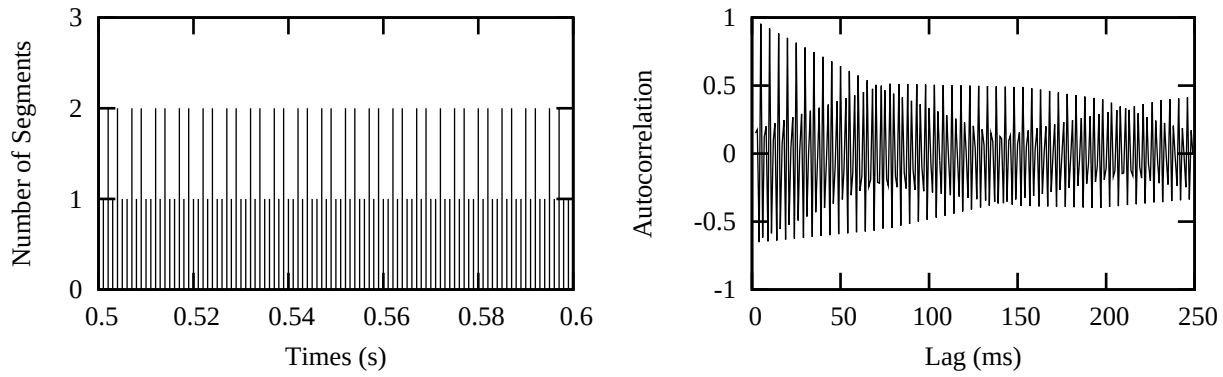

Figure 5.8: A pattern caused by saturation of a bottleneck link instead of self-clocking. Left: Partial measurement interval for a trace taken over a saturated 16Mb/s network link with an RTT of 18ms. Right: Autocorrelation plot for this measurement interval.

5.5.1 Saturation

In Section 4.1 we presented an algorithm to filter estimates from measurement intervals where saturation has replaced the clusters of the self-clocking pattern with a steady stream of segments. To test this filter, we created a saturated bottleneck using NIST Net by restricting bandwidth while allowing a virtually unlimited queue length. We set NIST Net's base delay to 0 to allow saturation to easily occur. The actual RTT is regulated by the time each segment spends in the queue.

We ran traces using bandwidth restrictions of 4Mb/s, 8Mb/s, and 16Mb/s. Figure 5.8 shows a typical measurement interval and its autocorrelation plot. As expected, saturation spreads out the segment arrival times so that the self-clocking clusters no longer exist. In the autocorrelation plot, the 5ms lag has produces the peak autocorrelation. However, there are seven segments per 5ms interval, which is what the filter expects.

For these traces, our variation filter discards the estimates before the saturation filter runs. This happens for the same reason that the small RTTs were filtered in Section 5.4: the small spacing between peak autocorrelations causes the mean to increase. However, the filter

threshold may be changed, or a smaller bandwidth could cause the spacing between peaks to increase.

Thus, we ran SCpet without variation filtering traces, and our tests produced the desired result in nearly all cases. Patterns caused by saturation instead of self-clocking are detected, and estimates for such measurement intervals are filtered out. The only exceptions were during the slow-start phase, where there were more than seven segments during the first RTT candidate, and a single estimate in the 4Mb/s trace due to a stall lasting a few milliseconds.

5.5.2 Bandwidth Range

While a saturated bottleneck link can change the self-clocking pattern, bottlenecks are not always saturated. Thus, we tested SCpet with a range of bandwidth restrictions. As before, we used NIST Net to set the bandwidth. We chose settings near common bandwidths of modems, home broadband, wireless links, and LANs. For each bandwidth setting, we set the queue length to the delay-bandwidth product to prevent saturation from inflating the base delay of 60ms.

Table 5.2 shows the results of our tests. In the 10 and 100Mb/s cases, SCpet produces the expected result. For the 0.5Mb/s bandwidth, the saturation filter discards large numbers of estimates. Due to the fact that the queue length is set to the delay-bandwidth product, the congestion window is forced to stay within three segments. The small number of segments per RTT cause many measurement intervals to appear saturated. If the queue length were larger, this trace would have been saturated similarly to the traces in the previous section. Among the estimates that do pass through the filter, many produce patterns at harmonic intervals of two or three times the actual RTT. This happens because the small queue length causes segments to be dropped frequently, which causes clusters to vary in size from 0 to 3 segments from one RTT to the next. When cluster sizes happen to align more closely at multiples of the RTT, these lags can produce higher autocorrelation levels than the RTT itself.

Table 5.2: A comparison of RTT estimates from SCpet and the sender's TCP implementation for traces with links links having different bandwidths. Estimates and standard deviations are shown for both estimation methods. The percent of estimates accepted by filtering is shown for the SCpet method.

	SCpet			Sender TCP	
Bandwidth (Mb/s)	RTT (ms)	St. Dev.	Percent Unfiltered	RTT (ms)	St. Dev.
0.05	208	32	45.5%	188	32
0.5	83	42	49.6%	61	0.08
10.0	61	0	100%	61	0.40
100.0	61	0	100%	61	0.18

For the 0.05Mb/s trace, the delay-bandwidth product is smaller than one segment. Setting the queue length to one segment causes saturation to inflate the bandwidth, which results in the large estimate from the server. Despite this problem, SCpet produces a similar average estimate to the server.

5.6 Competing Traffic

Previously, we have experimented with traces with network conditions emulated by NIST Net. Here, we add more realistic conditions by adding competing flows over a bandwidth restricted link. We expect that congested network conditions may cause problems with the self-clocking pattern, but we also expect that they will resemble the loss and variation conditions we have already tested.

We set up NIST Net with a 10Mb/s link and a 60ms delay. We used Surge [2, 1] on the receiver host to emulate Web users which request files from the Apache Web server running on the sender host. The competing traffic is generated from servicing the Web requests.

Fifty Web users emulated by Surge is enough to generate a throughput of 5.85Mb/s over a link with no bottleneck. We ran tests ranging from 0 to 250 users over our 10Mb/s link.

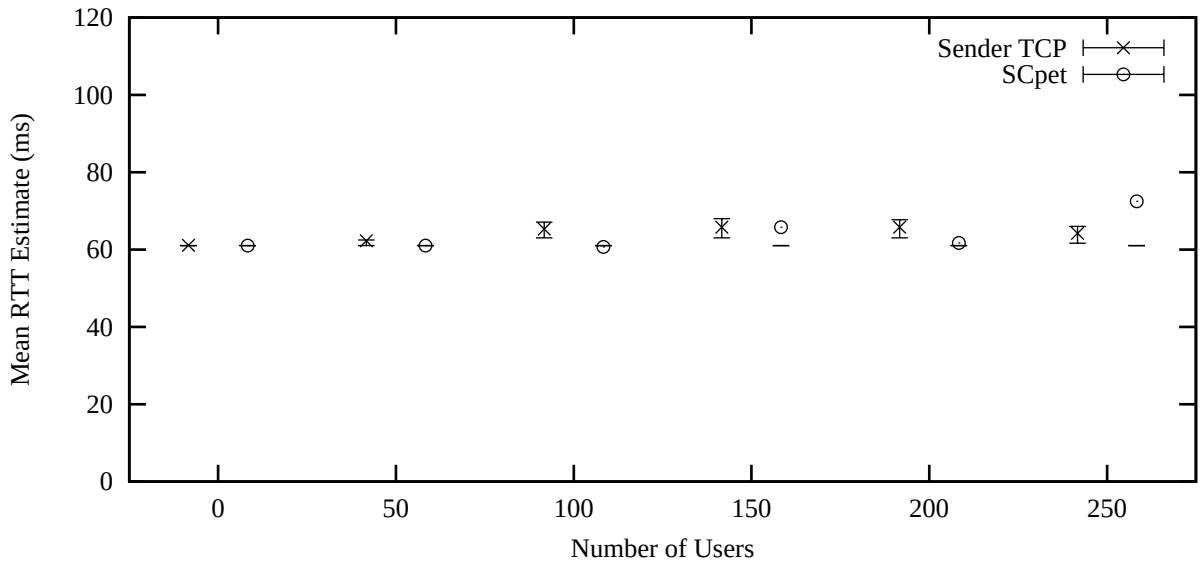

Figure 5.9: A comparison of RTT estimates from SCpet and the sender's TCP implementation for traces with competing traffic. Traces were generated by Surge for different numbers of emulated Web site users. Points represent the mean RTT estimates for each trace with a base delay of 60ms. Error bars indicate the first and third quartiles of the estimates.

Increasing the number of users beyond 250 caused the Apache service to stall, preventing it from increasing the amount of competing traffic.

Figure 5.9 shows the results of our experiment. For up to 100 users, SCpet produces accurate results. The high variations for 150 to 250 users are from picking harmonic multiples of the actual RTT. Similarly to Section 5.1, losses caused by heavy congestion can alter the sizes of segment clusters, causing multiples of the RTT to have stronger patterns, but there are only a few erroneous estimates. Only 4% of the estimates for the 150-user trace are at multiples of the RTT, while 13% of the estimates in the heavily congested 250-user case are at RTT multiples.

Thus, our experiments show that heavy competing traffic has a limited impact on SCpet's ability to estimate the RTT. For moderate levels of competing traffic, our algorithm produces reliable results.

Table 5.3: Internet FTP sites used to collect traces.

Site Name	Download URL
DTI	ftp://ftp.dti.ad.jp/pub/Linux/debian/ls-lR
FreeBSD	ftp://ftp10.us.freebsd.org/pub/FreeBSD/ls-lR.gz
Freenet	ftp://ftp.freenet.de/pub/ftp.debian.org/debian/ls-lR
JRiver	ftp://ftp.jriver.com/pub/downloads/tcp-plus-demo.exe
Pacific	ftp://mirror.pacific.net.au/debian/ls-lR
Stanford	ftp://ftp.cs.stanford.edu/pub/gcc.tar.gz
Sun	ftp://docs-pdf.sun.com/817-0924/817-0924.pdf
UIUC	ftp://ftp.cs.uiuc.edu/pub/research-groups/perts/binary.tar.Z
Washington	ftp://ftp.cs.washington.edu/pub/Maitrd.tar.Z

5.7 Internet Traces

Our previous experiments used a testbed network with emulated network conditions. To evaluate SCpet under realistic conditions, we captured traces from FTP downloads from nine Internet sites. For each site, three traces were taken at peak traffic times (2PM-3PM EST) and three were taken at off-peak times (2PM-3PM EST). These sites were chosen based on an earlier study of RTT behavior [14], which intended to choose sites covering a large range of RTTs. We also added three international sites: DTI in Japan, Freenet in Germany, and Pacific in Australia. Table 5.3 provides a list of the FTP sites and files we downloaded.

For our experiments, we used the timestamp-based passive RTT estimation method for reference, described in Chapter 2, since RTT estimates from the sender's TCP implementation were unavailable. While we used a 500ms measurement interval when possible, the FreeBSD peak traces and all the Pacific traces had RTTs that were too large. We used 1500ms and 1000ms measurement intervals for these traces, respectively.

For most traces, SCpet produces similar results to the timestamp-based method, but a few notable exceptions exist. The FreeBSD traces taken at peak times show signs of saturation. In fact, 81% of these estimates were caught by the saturation filter. Due to problems with

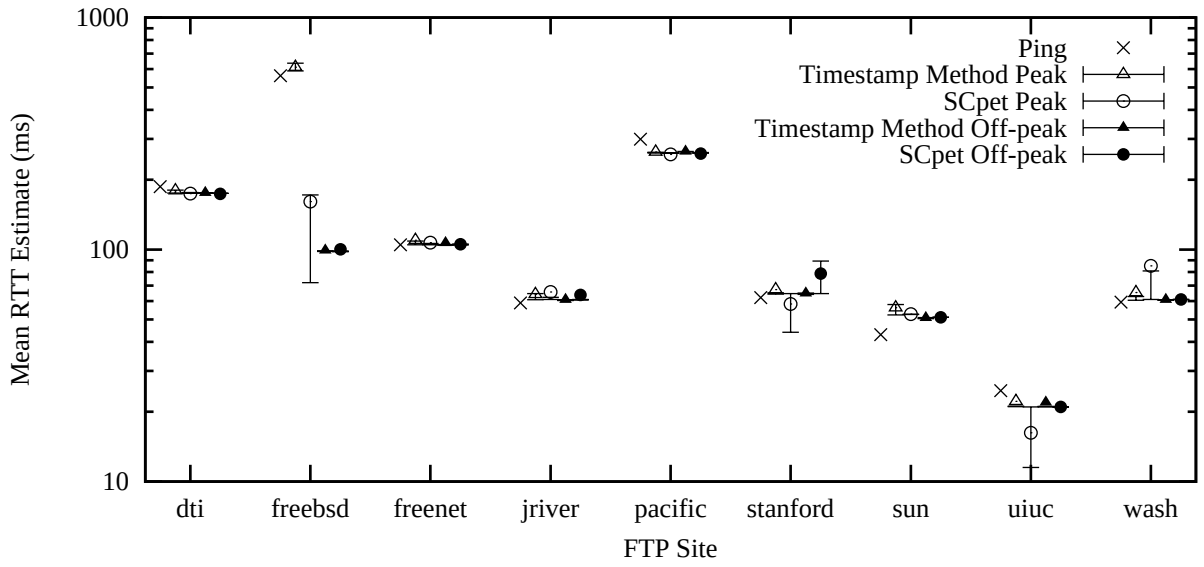

Figure 5.10: A comparison of RTT estimates SCpet and the timestamp-based method for FTP downloads from nine remote sites. Points represent the average estimate over three traces at peak traffic times and three at off-peak times. Error bars indicate the first and third quartiles of the estimates. Ping times were measured after the traces were captured.

variation, the remaining estimates slipped through. The low estimate for the peak UIUC traces occurred due to high autocorrelation levels at 2ms for some measurement intervals. This stems from an unusual situation where clusters are spread out at at small intervals similarly to the case with saturation. Unlike saturation, these clusters contain large numbers of segments. For the peak time Washington traces, there was a frequent tendency to choose harmonic multiples of the RTT. This stems from frequent changes in cluster sizes, which is similar to the problem with heavy loss rates in Section 5.1.

With a few limitations, SCpet performs well overall for this set of Internet traces. This suggests that self-clocking patterns are a viable means to passively estimate RTT for Internet downloads.

Chapter 6

Conclusions

We have presented a new method of passively estimating round-trip times using TCP self-clocking. Unlike previous methods, ours can make estimates throughout the lifetime of a session, and it can do this even when only one direction of network traffic can be monitored. We have shown that our estimates are robust under moderate levels of loss, variation, and congestion, and we have produced effective means of filtering patterns created by saturation. While our method usually produces correct results during adverse network conditions, there are a few limitations. With heavy loss, we can erroneously choose harmonic multiples of the RTT in a few cases. Also, we have difficulty filtering erroneous estimates due to random variation without also filtering accurate estimates as well. Nevertheless, self-clocking produces RTT estimates in many situations where previous methods cannot. Thus, tools like SCpet can be useful in the study of TCP behavior on the Internet.

Bibliography

[1] Paul Barford, Azer Bestavros, Adam Bradley, and Mark Crovella. Changes in Web client access patterns: Characteristics and caching implications. *World Wide Web*, 2(1), 1999.

[2] Paul Barford and Mark Crovella. Generating representative Web workloads for network and server performance evaluation. In *Measurement and Modeling of Computer Systems*, pages 151–160, 1998.

[3] Mark Carson and Darrin Santay. NIST Net: a Linux-based network emulation tool. *ACM Computer Communication Review*, 33(3):111–126, Jul. 2003.

[4] Kevin Fall and Sally Floyd. Simulation-based comparisons of Tahoe, Reno and SACK TCP. *Computer Communication Review*, 26(3):5–21, Jul. 1996.

[5] V. Jacobson. Congestion avoidance and control. volume 18, pages 314–329, Aug. 1988.

[6] V. Jacobson. Modified TCP congestion avoidance algorithm. end2end-interest mailing list, Apr. 1990.

[7] V. Jacobson, R. Braden, and D. Borman. TCP extensions for high performance. IETF RFC 1323, 1993.

[8] M. Jain and C. Dovrolis. End-to-end available bandwidth: measurement methodology, dynamics, and relation with TCP throughput. In *SIGCOMM*. ACM, Aug. 2002.

[9] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley. Inferring TCP connection characteristics through passive measurements. In *INFOCOM*. IEEE, Feb. 2004.

[10] J. Jiang and C. Dovrolis. Passive estimation of TCP round-trip times. *ACM Computer Communication Review*, 32(3), Jul. 2002.

[11] Guohan Lu and Xing Li. On the correspondency between TCP acknowledgment packet and data packet. In *Internet Measurement Conference*. ACM, 2003.

[12] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Topologically-aware overlay construction and server selection. In *INFOCOM*. IEEE, Jun. 2002.

[13] Bryan Veal, Kang Li, and David K. Lowenthal. New methods for passive estimation of TCP round-trip times. In *Sixth Passive and Active Measurement Workshop*, March 2005.

[14] Haijin Yan, Rupa Krishnan, Scott A. Watterson, and David K. Lowenthal. Client-centered energy and delay analysis for TCP downloads. In *12th IEEE International Workshop on Quality of Service (IWQoS)*, June 2004.

[15] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker. On the characteristics and origins of Internet flow rates. In *SIGCOMM*. ACM, Aug. 2002.