Round-Trip Time Inference Via Passive Monitoring

Ryan Lance
Department of Mathematics
University of Maryland
College Park, Maryland
rjl@math.umd.edu

Ian Frommer
Applied Mathematics and
Scientific Computation
University of Maryland
orbit@math.umd.edu

ABSTRACT

The round-trip time and congestion window are the most important rate-controlling variables in TCP. We present a novel method for estimating these variables from passive traffic measurements. The method uses four different techniques to infer the minimum round-trip time based the pacing of a limited number of packets. We then estimate the sequence of congestion windows and round-trip times for the whole flow. We validate our algorithms with the ns2 network simulator.

1. INTRODUCTION

Much of the traffic on the Internet consists of bulk TCP flows, in which the sender sends a long sequence of data packets to the receiver. The sender transmits a certain number of packets (its congestion window) and then waits until it receives acknowledgments (ACKs) from the receiver before sending more packets. The time between when the sender sends a data packet and when it receives the corresponding ACK is the current round-trip time (RTT) of the flow. The sender adjusts its window according to the TCP protocol, increasing it by one roughly every RTT, and dividing it by two each time a packet is dropped.

While the sender knows its congestion window and RTT, we are concerned with estimating these quantities from packet arrival times at a particular monitoring point. Since the monitor can be anywhere between the sender and receiver, the time between a data packet and its ACK does not determine the RTT. We instead infer the RTT from the pacing of packets. Packets can then be grouped into *flights*, which correspond to the congestion window. The main variable we use to infer the RTT is the *interpacket time*, which we define as the time from the head of one data packet to the head of the next data packet in a flow. Interpacket times vary by many orders of magnitude due to the dynamics of TCP and the asymmetric nature of networks. Therefore, we mainly work with the logarithm of the interpacket time.

There are other approaches to estimating the RTT, the simplest of which is the time from the SYN to the SYN/ACK during the set-up of a TCP connection. Another simple estimate is the time between the first two flights during slow start [2]. One cannot use these techniques if the bulk TCP flow is already in progress at the start of the trace. An alternate approach is used by Zhang et al. [7], who assume the number of packets per flight should follow certain patterns. They start with a pool of exponentially spaced candidate RTT values, which they use to partition the packets into groups. The best candidate RTT is the one that results in a sequence of groups that best fits congestion window behavior. Jaiswal et al. [1] use a finite state machine that "replicates" the TCP state, which is based on observed ACKs and depends on TCP flavor.

Our algorithm (the frequency algorithm) requires only one direction of a flow, assumes very little about the underlying traffic, and has the flexibility to deal with any flavor of TCP and time-varying RTTs. The basic idea of our algorithm is that the self-clocking nature of TCP causes the sequence of interpacket times to be approximately periodic, with a period roughly equal to the RTT. We apply our algorithm to data sets from the Passive Measurement and Analysis archive at NLANR [5]. These 90 second traces are taken from OC3, OC12, and OC48 links at gateways between university networks and the Abilene Backbone Network. The monitors passively record all packet headers on a link along with a timestamp that has microsecond precision. The bulk TCP flows, on which we focus, have a median of 5000 packets in the 90 second traces. We also validate our algorithm with the *ns2* network simulator, and find that the RTT and congestion window estimates have relative errors that are in the range of 0-5%.

2. FREQUENCY ALGORITHM

The RTT consists of a *fixed delay*, namely the transmission time without network congestion, plus a *queuing delay*, consisting of the time a packet and its ACK spend in queues. The first step of the frequency algorithm is to approximate the *minimum* RTT of a flow, which should roughly approximate the fixed delay. The four components of the algorithm are the sliding window estimate, the interpacket time autocorrelation function, the data-to-ACK-to-data time, and the Lomb periodogram. Estimates from the four components are combined in a way that identifies the smallest plausible RTT. We then approximate the congestion window, by grouping packets into flights based on the minimum RTT

estimate. The time between flights is an estimate of the current RTT.

To generate the initial estimate of the RTT we use only the first 256 interpacket times (257 packets), because the Lomb periodogram is based on the FFT and it is most efficient on sequences that are powers of two in length. For most flows, that is, those that do not use window scaling, 257 packets will comprise at least five complete flights, whereas 128 packets only guarantees two complete flights. This is enough data to obtain a good initial estimate of the RTT, but not so much that it becomes computationally expensive.

2.1 Sliding Window Upper Bound

The purpose of the sliding window estimate is to obtain an upper bound on the fixed delay. Let M be the maximum receive window. We let $M = 65535$ bytes if ACKs are not seen by the monitor. Let t_i be the timestamp of the ith data packet, and let b_i be the number of data bytes in the payload of that packet. For each i, let n_i be the smallest integer such that

$$\sum_{k=i}^{n_i} b_k > M.$$

As i increases, packets i to n_i form a sliding window of at least M bytes. Since the sender is required to send no more than M bytes per RTT, packets i and n_i must necessarily be in different flights, and hence they are separated by at least one RTT. Let i^* be the largest integer such that $n_{i^*} < 257$, and for each $i \leq i^*$ let

$$u_i = t_{n_i} - t_i \tag{1}$$

be an RTT upper bound estimate.

If the sender is transmitting less than M bytes per RTT, then u_i might grossly overestimate the RTT. On the other hand, when the congestion window is equal to M, it is possible for u_i to underestimate the RTT. This happens when packets at the end of one flight have some queuing delay, but none of the packets in next flight do. Even though underestimating the RTT in such a way is unlikely, we intend for u_i to be a strict upper bound on the fixed delay. We would like to obtain the least upper bound for the RTT from the values u_i. Therefore, instead of taking $\min u_i$ as the least upper bound, we use the more conservative estimate of the fifth percentile of $\{u_i\}$. Since i^* will be about 200 in most cases, choosing the fifth percentile equates to ignoring the lowest ten values. More formally, let $\{\hat{u}_i\}$ be the values $\{u_i\}$ sorted in increasing order. Our estimate of the least upper bound is defined as

$$u = \hat{u}_{\lfloor i^*/20 \rfloor}. \tag{2}$$

2.2 Autocorrelation Function

One technique to estimate the period of a nearly periodic time series is the autocorrelation function. Let $x_j, j = 1 \ldots n$ be a normalized time series with mean zero and standard deviation one. The unbiased autocorrelation function is defined as

$$A(k) = \frac{1}{n-k} \sum_{j=1}^{n-k} x_j x_{j+k} \tag{3}$$

for lags $k = 0 \ldots n-1$. If a time series has an approximate period of T, then one would expect $A(k)$ to have its largest local maximum for $k > 0$ at $k = T$.

We expect the interpacket times to be approximately periodic because of the self-clocking nature of TCP. Unfortunately, the interpacket times are unevenly spaced in time. One could ignore this fact and treat the them as if they were an evenly spaced time series. However, the sequence of interpacket times will be misaligned if the flow is in the linear increase phase of congestion avoidance or a drop occurs.

We handle these issues by linearly interpolating the logarithm of the interpacket times on an evenly spaced grid of times. We choose the step size of the grid, s, to be 1 ms. This is a trade-off between accuracy and computational workload. If we made the step size smaller, then we would more closely approximate interpacket times less than 1 ms, but the grid would be larger and computing $A(k)$ would take longer. Figure 1 shows a sample sequence of interpacket times, their linear interpolation, and $A(k)$ for the interpolated sequence.

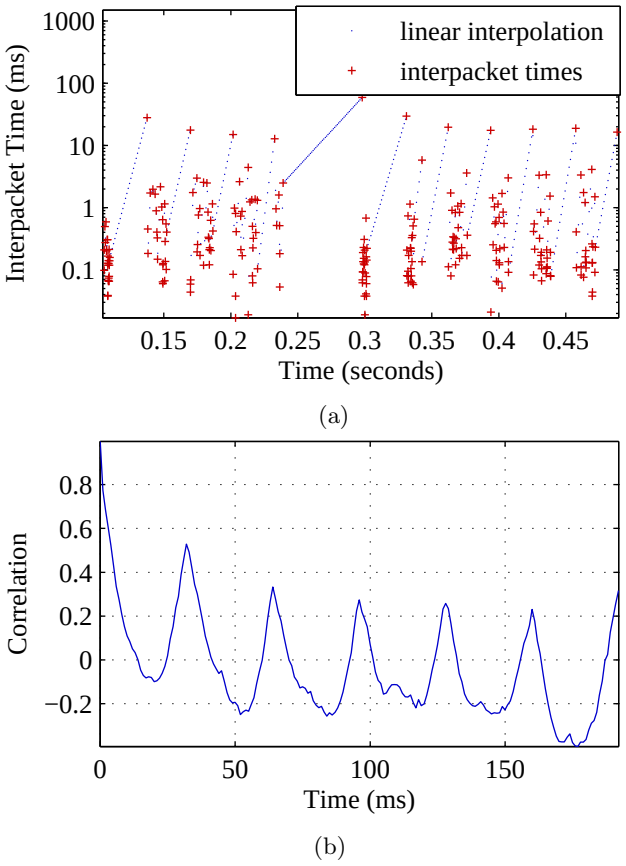

Figure 1: (a) The first 256 interpacket times of an FTP flow. The smaller dots are the linearly interpolated values on a grid with step size 1 ms. (b) The autocorrelation function of the interpolated data. Based on this plot one can estimate the RTT to be about 32 ms.

The grid size for the autocorrelation function influences how it is computed. Depending on the RTT and congestion window, 257 packets generally require between 0.1 and 4 seconds

to transmit. Therefore, the grid size, $g = \lfloor (t_{257} - t_1)/s \rfloor$, can be up to 4000 points. This raises the question of what the best method is to compute the autocorrelation function. Let $\{\tilde{\delta}_j\}_{j=1}^{g}$ be the normalized time series of linearly interpolated interpacket times. The Wiener-Khinchin theorem states that, for a continuous time function, the Fourier transform of the autocorrelation function equals the power spectrum. Thus, we can approximate $A(k)$ by

$$\tilde{A} = F^{-1}\left(|F(\tilde{\delta})|^2\right) \tag{4}$$

where F is the FFT operator and $|F(\tilde{\delta})|^2$ is the power spectrum. Computing $A(k)$ via the definition in Equation 3 requires $O(g^2)$ operations, but using Equation 4 reduces the complexity to $O(g \log_2 g)$. However, we do not need to compute $A(k)$ for all k. Since we have already computed u, the upper bound for the RTT, we only need to calculate $A(k)$ up to $k = \lceil u/s \rceil$. Now using the definition will only require $O(gu/s)$ operations. Suppose that we use both methods to compute the autocorrelation function for a flow with a 40 ms RTT and receive window equal to 12 packets. Using the definition takes about 34000 operations, while Equation 4 requires about 40000 operations. For each flow, one can determine whether it is more efficient to use the definition or Equation 4.

We approximate the RTT by $q = k^* s$, where k^* is the lag at which $A(k)$ attains its greatest local maximum on the interval $(0, \lceil u/s \rceil)$. However, $A(k)$ can fail to have its greatest local maximum at the value that corresponds to the true RTT. This can happen in two basic ways. First, $A(k)$ could be greater at the value that corresponds to twice the RTT; we can encounter this if u greatly overestimates the RTT. Secondly, there can be a spurious local maximum close to $k = 0$, which corresponds to a fairly large interpacket time that is repeated at even intervals.

By filtering $A(k)$ we have a better chance of finding the correct local maximum. To correct for q being too small, we use a simple low-pass filter by applying to $A(k)$ a moving average of the last β values. To correct for q being too large, we multiply $A(k)$ by a linear envelope that equals 1 at $k = 0$, and α at $k = \lceil u/s \rceil$. After originally calculating the unbiased autocorrelation function, multiplying by a linear envelope reintroduces bias, but is does so on the time scale of the RTT rather than the time scale of the grid. In practice we use two filtered versions of $A(k)$, one with $\beta = 4$ and $\alpha = 3/4$, another with $\beta = 8$ and $\alpha = 1/2$, along with the unfiltered $A(k)$. From the three versions of $A(k)$, we obtain three estimates of the RTT, q_j, $j = 1, 2, 3$. If $A(k)$ has no local maximum on the interval $(0, \lceil u/s \rceil)$, then $q_j = 0$.

2.3 Data-to-ACK-to-Data

The sliding window algorithm produces an upper bound, u, on the fixed delay. To obtain a lower bound, we find the time between data packets that occupy the same relative positions in two successive flights of packets. Since this method requires ACKs, it will not work if the monitor does not see ACKs for a particular flow. Also, this method is intended for flows in which the sender is closer to monitor than the receiver. If this assumption does not hold, and the sender is on the remote side of the monitor, then this method may result in an estimate that is spuriously low. Since we intend for this method to produce a lower bound, these spuriously low values are not a fatal flaw.

We now describe the data-to-ACK-to-data method. Suppose we have a flow that meets the requirements laid out above, and suppose the data packets have timestamps t_i and sequence numbers s_i and the ACKs have timestamps t_j^a and acknowledgment numbers a_j. For data packet i find the corresponding ACK, that is, the first ACK such that $t_j^a > t_i$ and $a_j > s_i$. Call the index of this ACK j'. Now find the first data packet that ACK j' liberates, that is, the data packet with index m_i such that $t_{m_i} > t_{j'}^a$ and $s_{m_i} > a_{j'}$. As in the sliding window algorithm, let i^* be the largest integer such that $m_{i^*} \leq 257$. Our estimate of the RTT lower bound is defined as

$$\ell = \min_{i \leq i^*} (t_{m_i} - t_i). \tag{5}$$

Figure 2 illustrates this method with two examples, one where the sender is closer the monitor, and another where the receiver is closer to the monitor.

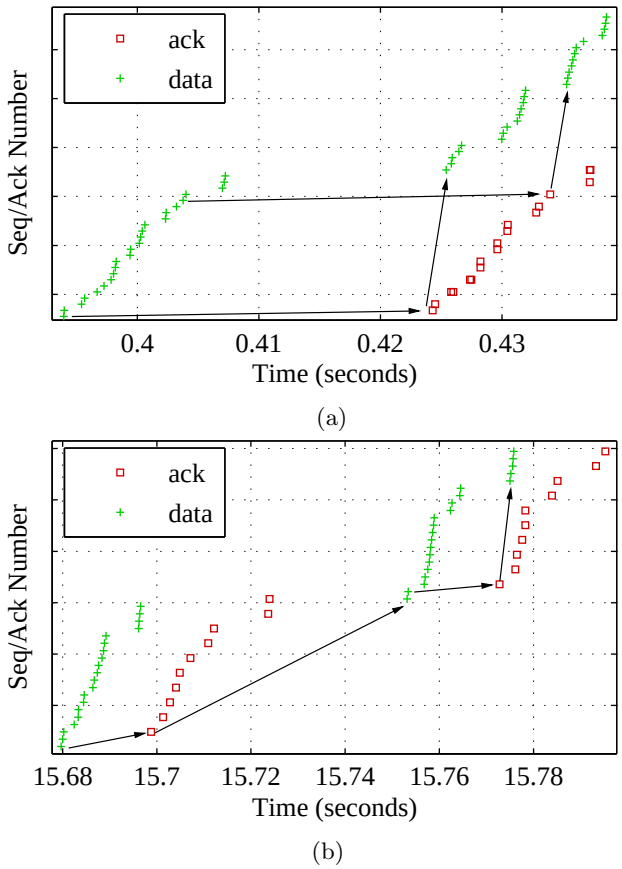

Figure 2: Sequence number plots for two flows from the same trace, note that the vertical axis is not labeled since the actual value of the sequence numbers does not matter. In (a) the monitor is closer to the sender, in (b) it is closer to the receiver. By following each pair of arrows in (a) we find a valid RTT estimate. In (b) the first pair of arrows gives a good estimate, but the result of the second pair is too small.

2.4 Lomb Periodogram

The Lomb periodogram [6] is a method for the spectral analysis of unevenly spaced data, such as interpacket times. We could instead use the power spectrum on linearly interpolated data, but the Lomb periodogram is more naturally suited to analyze data with highly irregular sampling. If there are long stretches without data, as there would be after a timeout or when a sender has no data to transmit, then the power spectrum can exhibit erroneously large power at low frequencies. Katabi et al. [3] noted that bottleneck links impose structure on the interpacket time distribution. The accuracy of the Lomb periodogram might be reduced by this structure, but it is still better than the power spectrum, as is shown in Figure 3.

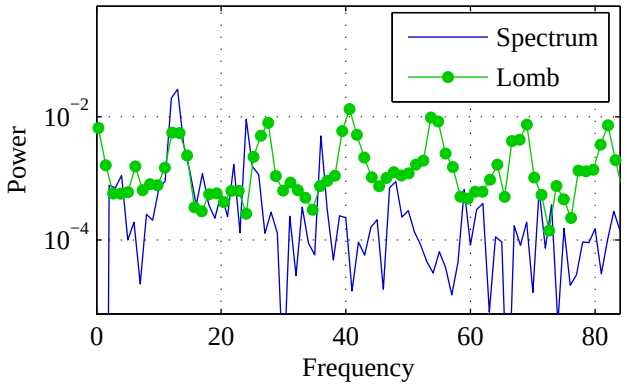

Figure 3: The Lomb periodogram and power spectrum have the same fundamental frequency of about 16, corresponding to a RTT of 65 ms. The periodogram picks up harmonics, but the power spectrum leaks power to lower frequencies and decays rapidly. This can cause the fundamental frequency to be distorted or obscured in flows with RTT greater than 100 ms. This comparison is based the flow in Figure 2(b).

Suppose we have data x_i measured at times t_i, $i = 1 \ldots n$, and suppose the data have mean $\bar{x}$ and standard deviation σ_x. The Lomb periodogram is a measure of spectral power at at angular frequency $\omega = 2\pi f$ defined by

$$L(\omega) = \frac{1}{2\sigma_x^2}\left[\frac{\left(\sum_i (x_i - \bar{x})\cos\omega(t_i - \tau)\right)^2}{\sum_i \cos^2\omega(t_i - \tau)} + \frac{\left(\sum_i (x_i - \bar{x})\sin\omega(t_i - \tau)\right)^2}{\sum_i \sin^2\omega(t_i - \tau)}\right]. \quad (6)$$

Here τ makes $L(\omega)$ invariant under any shift of the measurement times, which is a key property $L(\omega)$ shares with the Fourier transform. The value of τ depends on ω as follows:

$$\tan(2\omega\tau) = \frac{\sum_i \sin 2\omega t_i}{\sum_i \cos 2\omega t_i}. \quad (7)$$

For each fixed ω, Equation 6 is equivalent to solving the least squares problem where the data are fit to the model

$$y(t) = a\cos(\omega t) + b\sin(\omega t),$$

and letting $L(\omega) = a^2 + b^2$. This makes clear the difference between this method and the power spectrum on interpolated data. The FFT weights each time interval equally, but $L(\omega)$ weights each data point equally. The algorithm to compute the Lomb periodogram makes indirect use of the FFT and requires $O(n \log n)$ operations.

Let ω^* be the angular frequency that maximizes $L(\omega)$ on a suitable grid over the interval Ω, where $\Omega = [2\pi/u, 2\pi/\ell]$ if both u and ℓ are greater than 5 ms, and $u > \ell$, otherwise $\Omega = [2\pi/1.0, 2\pi/0.005]$. See [6] for information on how to compute the grid. We then define an estimate of the RTT by

$$\theta = \frac{2\pi}{\omega^*}. \quad (8)$$

2.5 RTT Lower Bound

We have defined a set of six RTT estimates, u, q_1, q_2, q_3, ℓ and θ; call this set S. We now combine the unreliable estimates in S into one reliable RTT estimate, $\tilde{r}$. We say informally that two estimates *agree* if they are within 10% of each other. The meaning of *agree* is formalized by defining a relation $\cong$, such that $a \cong b$ if $0.9 < a/b < 1.1$ or $0.9 < b/a < 1.1$ or $|a - b| < 3$ ms. The basic idea is to find the smallest value in S that agrees with at least one other estimate.

We set 5 ms as the minimum feasible RTT; any estimate less than this value is discounted. Let $S' = \{s \in S | s > 5\text{ms}\}$. As an intermediate step toward defining $\tilde{r}$, we define r' as:

$$r' = \min\{s \in S' \mid \exists t \in S',\ t \cong s\}. \quad (9)$$

If none of the elements of S' agree, then r' will be undefined. In such a case let $r' = \ell$ if the sender is on the local side of the monitor, otherwise let $r' = u$. Finally, we define $\tilde{r}$, the estimated lower bound of the RTT, as

$$\tilde{r} = 0.95r'. \quad (10)$$

We multiply by 0.95, since r' can potentially equal u, which is an upper bound for the fixed delay, or q_1, q_2, q_3, or θ, which are estimates rather than bounds.

It is possible for the RTT to be well above the fixed delay for the entirety of the first 257 packets. The initial estimate of $\tilde{r}$ will no longer be valid if the congestion later subsides. It is simple to continue evaluation of u and ℓ as new packets arrive. It is also possible to update the autocorrelation function and the Lomb periodogram if one uses their definitions instead of FFT-based approximations.

2.6 Congestion Window Estimation

The basic idea of the congestion window (CW) estimation scheme is as follows: given a past history of CW estimates $w_1 \ldots w_{k-1}$ and the first packet in the kth flight, start with $w_k = 0$ and increment w_k by the number of data bytes in each packet that arrives until one minimum RTT has elapsed, or until w_k is greater than w_{k-1}. We assume CW is initially equal to the receive window and let $w_0 = M$ be a placeholder that allows us to compare w_1 to w_0. This assumption is necessary since most flows are already in progress at the start of the trace, and thus, we normally do not see the slow start phase.

Two timers are needed for this algorithm; one limits the time between packets in the same flight, the other is the

expected time between flights. The first timer, c_1, is set at $0.75\tilde{r}$. When two packets are separated by more than c_1, we assume that a new window has begun. Let $\bar{b}$ be the median number of data bytes per packet, and let $p = 100(1 - \bar{b}/M)$. If the sender is transmitting M bytes per RTT, then we expect flow to have $M/\bar{b}$ packets per RTT. When the bandwidth delay product is much greater than M, we expect $M/\bar{b} - 1$ small interpacket times followed by one large interpacket time. Thus, we define the second timer, c_2, as the pth percentile of the interpacket times. We use c_2 to infer when congestion present.

As before, t_i is the timestamp, s_i is the sequence number, and b_i is the number data bytes of the ith packet. Define the ith interpacket time as $\delta_i = t_i - t_{i-1}$. Suppose we have counted w_k bytes so far in the current flight. Suppose that i is the index of the first packet in current window, and current packet has index $j \geq i$. We increment the flight size, $w_k \rightarrow w_k + b_j$, if condition 1 is satisfied, and either condition 2 or 3 is met:

1. *Receive Window:* $w_k + b_j \leq M$ and $\delta_j < c_1$
2. *Congestion Avoidance:* $w_k < w_{k-1} + \bar{b}$
3. *Slow Start:* $w_k < 2w_{k-1} < M$ and $\delta_i < c_2$

If condition 1 is false, or conditions 2 and 3 both fail, then packet j is deemed to part of the next flight, and we set $w_{k+1} = b_j$. Note that condition 3 uses δ_i not δ_j. The inequality $\delta_i < c_2$ means that flights k and $k-1$ were closer than expected, which is a sign of congestion. Condition 3 includes $\delta_i < c_2$ to ensure it is only satisfied when there is an indication of congestion. The inequality $\delta_j < c_1$ in condition 1 allows the kth flight to terminate when $t_j < t_i + \tilde{r}$, that is, before one minimum RTT has elapsed. This inequality prevents w_k from increasing unduly, since without it condition 3 will often be satisfied outside of slow start. It also allows for realignment of flights to more natural boundaries, but can lead to spuriously small CW or RTT values that can be filtered out in post-processing.

3. VALIDATION

We validate the RTT and congestion window estimation algorithms via simulation using the *ns2* network simulator [4]. Although our simulation is highly simplified, it reflects key features of the university networks from which the trace data originated. We use two network configurations; in configuration S the monitor is positioned closer to the senders, in configuration R it is closer to the receivers. Figure 4 shows the network diagram of configuration S; configuration R is similar, except the positions of the senders and receivers are reversed, as are the cross traffic sources and sinks.

We studied four bulk FTP flows between sender-receiver pairs (1,12), (2,14), (3,16), and (4,18). We refer to these as flows 1, 2, 3 and 4, respectively. The receive window is 22 packets for flows 1 and 2, and 44 packets for flows 3 and 4. All transfers were 90 seconds long, so that the amount of traffic generated would be similar to that of real flows. Cross traffic is also generated with the intent of being realistic. All TCP flows use TCP Reno and delayed ACKs. All routers use RED with a maximum queue of 45 packets, in the simulations no queue ever reached 40 packets.

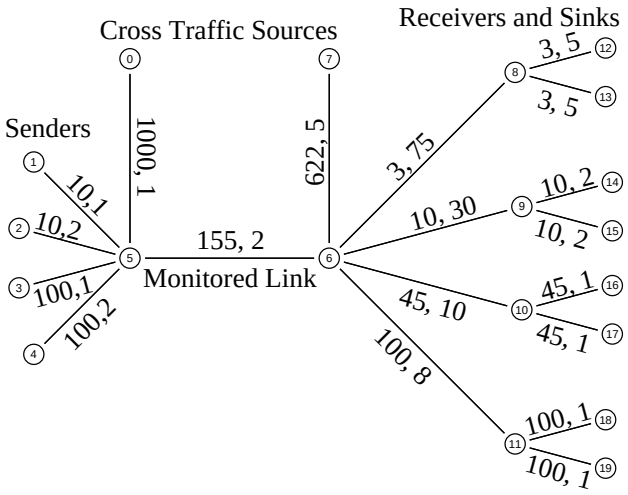

Figure 4: This network diagram illustrates configuration S. The pair of numbers on each link are the link capacity in Mbps, and the one way delay in milliseconds. For example the fixed delay for sender-receiver pair (1,12) is 166 ms.

For each configuration, we perform two simulations with different cross traffic properties. In simulation 1 cross traffic does not travel across the monitored link. For example, in configuration S simulation 1 (S1) cross traffic flows between source host 7 and sink hosts 13, 15, 17, and 19. In simulation 2 cross traffic travels on both the bottleneck links and monitored link, so that drops occur both before and after the monitoring point. Table 1 summarizes the flow orientation for each simulation.

	R1	R2	S1	S2
Senders	12-18	12-18	1-4	1-4
Receivers	1-4	1-4	12-18	12-18
Source(s)	13-19	13-19	7	0
Sink(s)	7	0	13-19	13-19

Table 1: Each column summarizes the flows orientation for the simulation at the top of the column. For example, in simulation R2 flow 3 has host 16 as sender and host 3 as receiver. Note that the host numbering remains fixed in Figure 4, but the role of each host can be either sender or receiver.

The cross traffic is a mix of TCP and constant bit rate UDP flows, as well as exponential on-off (EOO) flows. An EOO flow is a constant bit rate flow that sends bursts of traffic where the burst duration is an exponentially distributed random variable with mean λ_{ON}, and the idle time between bursts is exponentially distributed with mean λ_{OFF}. In all simulations we let $\lambda_{\text{ON}} = \lambda_{\text{OFF}} = 0.5$ seconds. We use simultaneous low bandwidth EOO flows to generate random oscillations in the cross traffic. All packets are 1500 bytes, except for the UDP packets, which are 640 bytes. We chose 640 bytes, since that is a common packet size for Real Time Streaming Protocol. The amount of cross traffic is similar in all four simulations, the only difference is position of the source and destination hosts, as detailed in Table 1. We summarize the cross traffic characteristics in Table 2.

	13		15		17		19	
UDP	0.1	(1)	0.25	(1)	2.0	(1)	5.0	(1)
EOO	0.05	(6)	0.125	(8)	1.0	(12)	2.5	(16)
TCP	22	(3)	22	(4)	44	(6)	44	(8)

Table 2: The numbers at the top of each column denote the cross traffic source (configuration R), or sink (configuration S). Numbers in parentheses are the number of concurrent flows of that type. Decimal numbers in the UDP and EOO rows are the constant rates (in Mbps) for the flow(s). In the TCP row, 22 and 44 denote the size of the receive window.

We test the RTT and congestion window algorithms on the four main flows. We expect the algorithms to perform better on simulations using configuration S, since the perspective of the monitor is closer to that of the sender. The simulator records the true value of the congestion window, RTT, and the timestamps of data packets and ACKs when they leave or enter link $5-6$ at host 5.

The first test is to validate the method of estimating the RTT lower bound. We break each flow into segments of 257 packets and compute $\tilde{r}$ for each segment. The algorithm is successful for a given segment if the estimate is less than the minimum true RTT, but within 15% of that value. Figure 5 shows the estimate and the minimum true round-trip times for one flow. Here the algorithm is successful on 52 out of 53 segments for a success rate of 98%. Four out of sixteen simulated flows had a success rate of 100%. The median success rate was more than 98%. The algorithm performs better on configuration S, where the median success rate is over 99%, whereas the rate is 95% for configuration R.

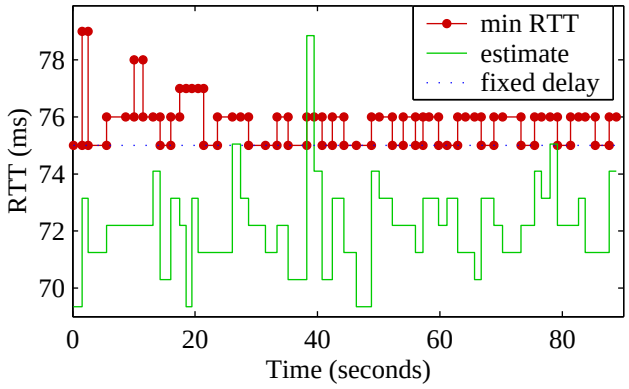

Figure 5: The interval [0,90] is broken into 53 segments of 257 packets each. The lower curve is the RTT estimate, the upper curve is the minimum true RTT on each interval, and the dashed line is the fixed delay. This plot is for flow 2 of simulation R2.

Upon obtaining an estimate of the RTT lower bound we proceed to estimate the sequence of flights using the method outlined in Section 2.6. The sequence of round-trip times are then estimated as the time from the start of one flight to the start of the next flight. The median absolute error for the flight sizes as an estimate of the true CW is between 0.3 and 1.0 packets for all simulated flows. The flight size is usually less than the true CW; this bias is due to delayed ACKs. The RTT estimates have no bias, their median absolute errors are summarized in Table 3.

	R1		R2		S1		S2	
1	4.8	2.5%	9.9	5.1%	1.1	0.56%	2.9	1.5%
2	1.2	1.5%	1.1	1.4%	0.4	0.49%	0.5	0.62%
3	0.5	1.3%	0.4	1.0%	0.4	1.0%	0.4	1.0%
4	0.3	1.1%	0.3	1.1%	0.3	1.1%	0.4	1.5%

Table 3: The first number in each box is the median absolute error (in ms) of the RTT estimate. The second number is the error relative to the true RTT.

4. CONCLUSION

We have presented a novel algorithm for inferring the minimum RTT of a flow from passive measurements. The results of the algorithm agree quite well with the true congestion window and RTT as recorded by the simulator. The main advantage of this algorithm is its flexibility. It assumes nothing about the position of the monitoring point, the flavor of TCP or the nature of cross traffic, and most importantly it still functions even if the ACKs are not observed at the monitor. The six different estimates of the RTT provide robustness should one or more of them fail to be correct. The techniques introduced in this paper will prove to be useful even as new TCP flavors and options are introduced. They may also prove useful in inferring characteristics of congestion.

5. ADDITIONAL AUTHORS

Additional authors: Brian Hunt, Edward Ott and James Yorke (University of Maryland) and Eric Harder (US Department of Defense).

6. REFERENCES

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

[2] H. Jiang and C. Dovrolis. Passive estimation of TCP round-trip times. *Computer Communications Review*, July 2002.

[3] D. Katabi, I. Bazzi, and X. Yang. A passive approach for detecting shared bottlenecks. In *Proc. of the IEEE International Conference on Computer Communications and Networks*. IEEE, 2001.

[4] ns2 Network Simulator. http://www.isi.edu/nsnam/ns, 2000.

[5] Passive Measurement and Analysis archive. National Laboratory for Applied Network Research. http://pma.nlanr.net, 2005.

[6] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C*. Cambridge University Press, 1992.

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