Passive Estimation of TCP Round-Trip Times

Hao Jiang
Computer and Information Sciences
University of Delaware
hjiang@cis.udel.edu

Constantinos Dovrolis
Computer and Information Sciences
University of Delaware
dovrolis@cis.udel.edu

ABSTRACT

We propose and evaluate a passive measurement methodology that estimates the distribution of Round-Trip Times (RTTs) for the TCP connections that flow through a network link. Such an RTT distribution is important in buffer provisioning, configuration of active queue management, and detection of congestion unresponsive traffic. The proposed methodology is based on two techniques. The first technique is applicable to TCP caller-to-callee flows, and it is based on the 3-way handshake messages. The second technique is applicable to callee-to-caller flows, when the callee transfers a number of MSS segments to the caller, and it is based on the slow-start phase of TCP. The complete estimation algorithm reports an RTT for 55-85% of the TCP workload, in terms of bytes, in the traces that we examined. Verification experiments show that about 90% of the passive measurements are within 10% or 5ms, whichever is larger, of the RTT that *ping* would measure. Also, measurements on several NLANR traces show that the two estimation techniques agree within 25ms for 70-80% of the processed TCP connections. We also apply the estimation methodology on a number of NLANR traces and examine the variability of the measured RTT distributions in both short and long timescales.

1. INTRODUCTION

Passive monitors are increasingly used by network operators and researchers [6, 8]. This technology, which originated with Jacobson's *tcpdump* in the late eighties [9], allows recording of all network traffic that flows through a link. With passive monitors we can observe the entire workload of a link, as opposed to active measurement tools that can just take samples of the end-to-end behavior in a path. In addition, passive monitoring does not introduce traffic overhead, and thus it does not interact with what is being measured.

Prepared through collaborative participation in the Communications and Networks Consortium sponsored by the U. S. Army Research Laboratory under the Collaborative Technology Alliance Program, Cooperative Agreement DAAD19-01-2-0011. The U. S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

Our objective in this paper is to estimate the Round-Trip Times (RTTs) of the TCP connections that go through a network link, using passive measurements at that link. In other words, we start with a traffic trace from a link, and then attempt to measure the RTT of every TCP connection by only investigating the connection's *unidirectional flow* recorded in that trace. If a TCP connection between hosts X and Y was actively opened by X, i.e., X sent the first SYN message, we define that X is the *caller* and Y is the *callee*. A unidirectional TCP flow in a trace can be from the caller to the callee, or from the callee to the caller. Note that a trace does not always include both flows of a TCP connection, even when the passive monitor records traffic in both directions of the link; the reason is that the routes between the caller and the callee may be asymmetric. Our first measurement technique, called *SYN-ACK (SA) estimation*, is applicable to all TCP flows from the caller to the callee. Our second measurement technique, called *Slow-Start (SS) estimation*, is applicable to some TCP flows from the callee to the caller, namely those that transfer at least five consecutive segments, the first four of which are Maximum Segment Size (MSS) packets.

The complete estimation algorithm produces an RTT value for 50-60% of the TCP connections, and for 55-85% of the TCP bytes, in the traces that we examined. Most of the connections that we cannot measure an RTT for are callee-to-caller flows that do not satisfy the previous MSS-related requirement. Additionally, we prefer to ignore connections for which we have some evidence that the RTT estimate may be wrong, reporting an estimation failure. This means that we give a higher priority to accuracy than 'estimation yield'. Our experiments show that about 90% of the passive measurements are within 10% or 5ms, whichever is larger, of the RTT value that *ping* would measure. Measurements on several NLANR-MOAT traces [18] show that the SA and SS estimation techniques agree within 25ms for 70-80% of the processed TCP connections. We have also applied these estimation techniques on a number of NLANR traces in order to examine the variability of the measured RTTs in both short timescales (seconds, minutes) and long timescales (hours, days, months).

Some preliminary RTT measurements using passive measurements are reported in [12, 23], based on variations of the SYN-ACK (SA) estimation technique. *tcptrace* measures several RTT values for each connection from the delay between non-retransmitted data packets and their corresponding ACKs. This technique works only when the trace is collected at the TCP sender [19]. Balakrishnan et. al used TCP traces at a WWW server to reproduce the evolution of several TCP state variables, including the connection's RTT [4]. Allman measured several RTT values during the lifetime of each HTTP connection at a WWW server, using passive measurements at that server [1]. Measuring the RTTs of a connection at end-hosts is easier, as we have access to both data segments and their ACKs. Generally, our measurements cannot be as accurate as those in [1], because the SA and SS techniques operate on unidirectional TCP flows in the 'middle of the network', where the impact of the network jitter and losses is more significant. Also, the estimation techniques that we propose are simple, making it likely that routers could estimate the RTT of a new TCP connection *in real-time*. More sophisticated algorithms, such as the algorithm in §3.4 of [2], may be able to infer the evolution of the congestion window from unidirectional measurements, at the cost of higher computational complexity and per-flow state, however.

The RTT that a connection experiences varies, due to queueing delay variations or routing changes. Our estimation techniques attempt to provide one measurement per connection, which is *the RTT that the connection experienced during connection establishment or during its first slow-start round-trip*. Study of estimating retransmission timer (RTO) in [2] shows that taking only one RTT sample for the RTO estimator works fairly well. [1] reports additional measurements on the RTT variations that a connection can experience during its lifetime.

The knowledge of connection RTTs at a network link has several applications. First, the amount of required buffering at a link depends on the RTTs of the TCP connections that go through that link [24, 15]. An estimate of the TCP RTTs is also useful in the configuration of active queue management modules [13], and in the identification of congestion unresponsive traffic [7, 11]. The proposed RTT estimation techniques can provide input to these network provisioning and traffic management modules.

The paper is structured as follows. Section 2 gives the basic idea for the two RTT estimation techniques, SA and SS. Section 3 describes the implementation of these techniques, and explains some additional estimation heuristics and correctness checks. Section 4 outlines the verification approaches that we followed and shows typical accuracy results. Section 5 presents RTT distributions from different monitored links, focusing on the variations of the RTT distributions over both short and large timescales.

2. BASIC TECHNIQUES

We focus on RTT estimation techniques that operate on a trace of all TCP traffic that flows in a network link during a time period. Such a trace can be collected with a passive monitor that is installed at the corresponding link [6]. The trace must include the IP and TCP header fields and an accurate timestamp for each packet. For each monitored TCP connection the trace includes the flow from the caller to the callee, or the flow from the callee to the caller. The trace may not record both these flows, either because the routes between the two hosts are asymmetric, or because the link (or the passive monitor) is unidirectional[1].

We use two different RTT estimation techniques, depending on the type of TCP flow that the trace records. When we see a flow from the caller to the callee, the connection's trace starts with the SYN packet that initiates the three-way handshake, followed by the the first ACK packet that closes that handshake [22]. In that case, the RTT is estimated based on the *SA (SYN-ACK) estimation* technique. The second estimation technique, that we refer to as *SS (Slow-Start) estimation*, is applied to callee-to-caller flows that transfer at least five consecutive segments, the first four of which are MSS packets. Usually, but not always, such flows are from a server (callee) to a client (caller) (e.g., HTTP reply flows). It is important to note that we cannot measure the RTT of all callee-to-caller flows.

All traces used in this paper are publicly available at the NLANR-MOAT site [18]. The traces (and the corresponding NLANR files) are named as *Site Abbreviation - Timestamp*. For example, the trace OSU-990050251 was taken at 17:57:31 EST on 5/16/2001 at the Ohio State University (OSU) access link[2]. The capture times of all traces are reported in Eastern Standard Time (EST).

2.1 SYN-ACK (SA) estimation

When we see a flow from a TCP caller to a callee, the RTT is estimated from the packets exchanged during the three-way handshake. Specifically, in that case the trace starts with one (or more in case of losses) SYN packet, followed by an ACK from the caller to the callee. Note that the trace may not include the SYNACK packet from the callee to the caller, because that packet is sent in the reverse-direction flow. The basic idea in SA estimation is that *the RTT can be estimated from the time interval between the last-SYN and the first-ACK that the caller sends to the callee*. This time period is shown in Figure 1.

The SA technique provides an accurate RTT estimate when the following three conditions are met. First, the transmission of the SYNACK packet from the callee, and of the first-ACK packet from the caller, is not delayed. Second, the SYNACK packet is not lost while in transit, and the first-ACK is not lost before it reaches the monitor. Third, the time spacing between the last-SYN and first-ACK packets at the caller and at the monitor is roughly the same, i.e., the delay jitter that is introduced in the network between these two packets is not significant. As will be shown in §4, even though these three conditions are not always met, the SA technique works quite accurately for the majority of TCP

[1]It is also possible that the trace does not include all packets between the two hosts because of route changes or multipath forwarding. In such cases, our techniques may fail to estimate the connection's RTT, depending on which packets are recorded in the trace.

[2]More information about the location and status of each monitored link can be found at http://pma.nlanr.net/PMA/Sites.

connections in the traces that we have experimented with.

As an illustration of the SA technique, Figure 2 shows the first two packets for six connections of the same HTTP session from a certain client to a WWW server. For each connection, there is a line for every recorded packet. The first line gives the packet number, timestamp of arrival at the monitor (in seconds), source and destination IP addresses (encrypted), source and destination port numbers, packet size (bytes), and some TCP flags. The following lines give the time spacing ('delta') of that packet relative to the previous packet, as well as the packet size and some TCP flags. The RTT estimate that results with the SA technique is given last. Note that the six RTT estimates in this HTTP session are quite close (with $1ms$ of each other). This was expected, as these six connections are established at about the same time between the same pair of hosts.

2.2 Slow-Start (SS) estimation

The Slow-Start (SS) technique considers TCP flows that transfer data from the callee to the caller. Among all such flows in the trace, *the Slow-Start technique can be applied when the recorded flow starts with at least five consecutive segments, the first four of which are MSS packets.* The MSS value can be estimated from the connection's trace, comparing the flow's largest segment size with one of the 'well-known' MSS values, such as 1460 bytes, 1452 bytes, or 536 bytes (see [14] for more such values). Thus, the SS estimation technique is not applicable for TCP flows that transfer only small segments, or segments with unusual MSS values.

After the three-way handshake is completed, the behavior of a TCP data transfer is governed by the slow start algorithm [10, 3]. During slow start, the TCP sender increments the congestion window variable (**cwnd**) by one MSS for each received ACK that acknowledges new data. The value of the Initial Window (IW) is usually up to two MSSs [10]. Experimental TCP extension allows that a TCP may use a larger initial congestion window up to four MSSs [3]. When $ICW = 2MSS$, and if the sender has enough data, it will send two MSS packets to the receiver. Note that these two first MSS packets are sent back-to-back, in a burst. We refer to these MSS packets as the flow's *first MSS burst*, or simply *first burst*. When the ACK for these two packets is received[3], the sender increases **cwnd** to three, and sends up to three additional MSS packets back-to-back. We refer

[3]With Delayed-ACKs, there will be only one ACK for the first two packets.

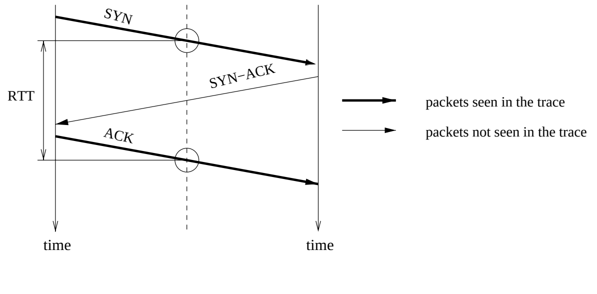


Figure 1: SYN-ACK (SA) RTT estimation.

```
- Connection-1
pack-1:T=0.522292 SA=10878977 DA=131073 SP=2418 DP=80 LEN=48 SYN
pack-2:Δ=68.90 ms                                     LEN=40 ACK
=> RTT=68.90 ms
- Connection-2
pack-1:T=0.523889 SA=10878977 DA=131073 SP=2419 DP=80 LEN=48 SYN
pack-2:Δ=69.46 ms                                     LEN=40 ACK
=> RTT: 69.46 ms
- Connection-3
pack-1:T=0.527233 SA=10878977 DA=131073 SP=2420 DP=80 LEN=48 SYN
pack-2:Δ=68.74 ms                                     LEN=40 ACK
=> RTT: 68.74 ms
- Connection-4
pack-1:T=0.531239 SA=10878977 DA=131073 SP=2421 DP=80 LEN=48 SYN
pack-2:Δ=70.47 ms                                     LEN=40 ACK
=> RTT: 70.47 ms
- Connection-5
pack-1:T=0.664952 SA=10878977 DA=131073 SP=2422 DP=80 LEN=48 SYN
pack-2:Δ=70.14 ms                                     LEN=40 ACK
=> RTT: 70.14 ms
- Connection-6
pack-1:T=0.668825 SA=10878977 DA=131073 SP=2423 DP=80 LEN=48 SYN
pack-2:Δ=69.23 ms                                     LEN=40 ACK
=> RTT: 69.23 ms
```

Figure 2: Sample trace of SA RTT estimation for an HTTP session with six connections.

to these packets as the flow's *second burst*. *The basic idea in SS estimation is that the time spacing between the first and second bursts is roughly equal to the connection's RTT.* This is shown in Figure 3-a. The first burst just after the SYNACK packet should consist of at least two MSS packets, and it may include more (up to 4 MSS packets) if the connection has a larger ICW. The SS technique is accurate when the time spacing between the two bursts, as measured at the monitor (RTT_m), is approximately equal to the time spacing between the two bursts at the sender (RTT_s). Delay jitter in the network can of course distort this spacing, increasing or decreasing the RTT estimate.

In some TCP implementations, the ICW is only one MSS packet [1]. In that case, the sender first sends one MSS packet, and after the ACK for that packet arrives, it sends up to two MSS packets in a burst. This case is shown in Figure 3-b. The time spacing between the first packet and the burst of the second/third packets may be larger than the connection's RTT, because of the *Delayed-ACK* algorithm [5]. According to that algorithm, the receiver acknowledges immediately only every second packet, and so the ACK for the first packet will not be generated until a certain TCP timer expires. It is because of this issue that the SS technique requires that the first burst should consist of at least two MSS packets.

An example of the SS technique is shown in Figure 4 for two callee-to-caller flows of the same HTTP session. The trace shows the first 7 packets of each flow. The third and fourth packets constitute the first burst in the flow. After a 'silence period', some additional data packets are sent from the Web server to the client back-to-back (second burst). The RTT is estimated from that silence period, which is about the same

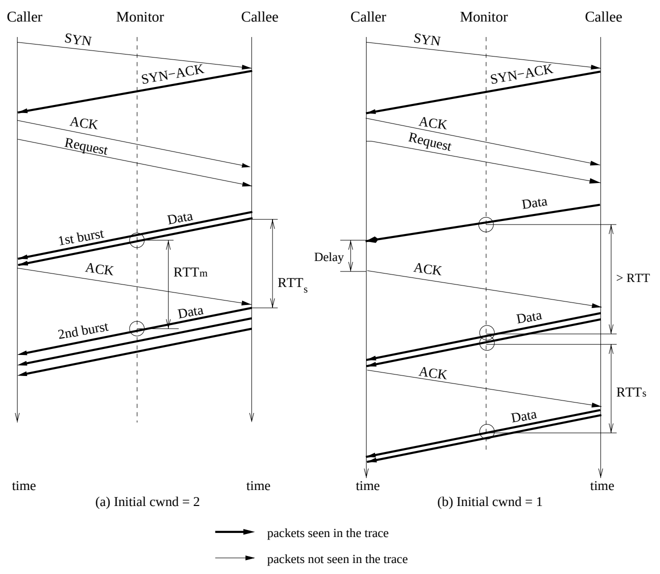


Figure 3: Slow-Start (SS) RTT estimation.

```
- Connection-1
pack-1:T=0.78318 SA=31073 DA=337729 SP=80 DP=1026 LEN=44 SYN ACK
pack-2:Δ=119.94 ms                                 LEN=40 ACK
pack-3:Δ= 8.11 ms                                LEN=1500 PSH ACK
pack-4:Δ= 0.10 ms                                LEN=1500 PSH ACK
pack-5:Δ=115.83 ms                               LEN=1500 PSH ACK
pack-6:Δ= 0.12 ms                                LEN=1500 PSH ACK
pack-7:Δ= 0.12 ms                                LEN=1500 PSH ACK
=> RTT: 115.83 ms
- Connection-2
pack-1:T=2.39328 SA=31073 DA=337729 SP=80 DP=1032 LEN=44 SYN ACK
pack-2:Δ=160.98 ms                                 LEN=40 ACK
pack-3:Δ= 3.05 ms                                LEN=1500 PSH ACK
pack-4:Δ= 0.12 ms                                LEN=1500 PSH ACK
pack-5:Δ=117.87 ms                               LEN=1500 PSH ACK
pack-6:Δ= 0.13 ms                                LEN=1500 PSH ACK
pack-7:Δ= 0.08 ms                                LEN= 504 ACK FIN
=> RTT: 117.87 ms
```

Figure 4: Sample trace of SS estimation for an HTTP session with two connections.

for both connections ($116 - 118$ ms).

3. FURTHER IMPROVEMENTS

The previous section described the basic idea in the SS and SA techniques. An accurate RTT estimation algorithm, however, needs to consider several 'harder cases', such as connections that experience losses, retransmissions, or queueing. In this section, we present some additional estimation heuristics and correctness tests that we developed to deal with, or at least detect, such cases, and give pseudocode for the two estimation techniques.

3.1 SA estimation

3.1.1 Losses in the SA technique

Consider a caller-to-callee flow during the three-way handshake. If the caller's ACK is lost before it reaches the monitor, the spacing between the last SYN and the first ACK packets may include a retransmission timeout. In that case, the SA-based RTT (RTT_{SA}) will be overestimated. According to RFC 2988, the initial retransmission timeout (RTO) should be set to 3 seconds [21]. Even though certain systems use an initial RTO that is as low as 1.7 seconds or as high as 6 seconds, most TCP implementations follow the 3 seconds requirement [16]. When our SA estimate is larger than 3 seconds, we presume that it includes an initial RTO and discard it. This means that the RTTs that we can measure with the SA technique are limited to 3 seconds. We expect that only a few TCP connections have an RTT that is more than 3 seconds in today's Internet.

3.1.2 Delayed first-ACK at the caller

The SA technique assumes that the caller will reply with the first ACK immediately after receiving the callee's SYNACK. This is not always the case, however. Certain operating systems, such as AIX and some versions of Linux, provide socket options for delaying the first ACK. The advantage of this option is that the first ACK can be piggybacked with the caller's first request/data packet [17]. Even though this socket option saves one ACK packet for the client, it can cause significant RTT overestimation errors in the SA technique. As an illustration of this effect, Figure 5 shows the first two packets (SYN and first ACK) for five connections of the same HTTP session. The RTT estimates that the SA technique gives vary from about 250ms to 390ms. It is likely that these large variations are caused because the client delays the transmission of the first ACK. We do not have a general test to detect this problem. However, in certain cases we can identify whether the SA estimate is wrong using some simple tests that we describe next.

3.1.3 The HTTP-request sanity check

In an HTTP connection, the caller (client) sends an HTTP request to the callee (server) after the connection is established. After the client receives the first one or two data packets of the HTTP reply, it sends an ACK packet to the server. As shown in Figure 6, the time interval X between the HTTP request and the subsequent ACK sent by the client is larger than the connection's RTT, because X includes processing delays at the server, and possibly a Delayed-ACK timeout at the client. Consequently, when the monitored flow is an HTTP request we can use X as an upper bound on the estimated RTT. If the SA estimate is $RTT_{SA} > X$, it is discarded and no estimate is given for that connection.

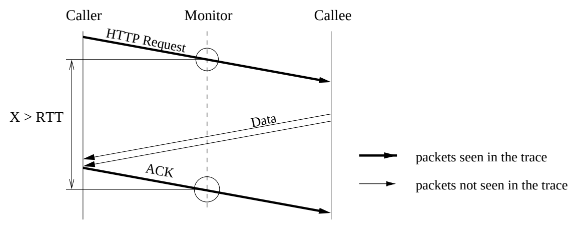


Figure 6: The HTTP-request sanity check.

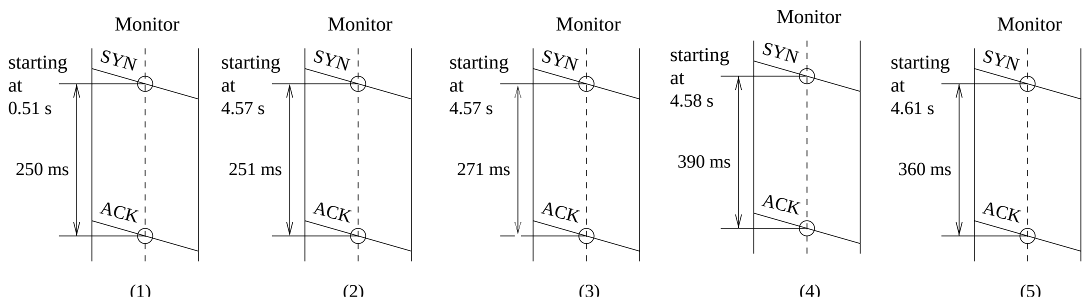


Figure 5: First two packets for five caller-to-callee flows of the same HTTP session.

```
foreach (SA flow)
{
    RTT = P[first_ACK].time - P[last_SYN].time;
    if (flow is HTTP request)
        X = P[Ack_after_request].time - P[request].time;
    if ((RTT <= X) && (RTT < 3sec))
        return RTT;
    else    skip;
}
```

Figure 7: Pseudocode for SA estimation.

3.1.4 *Pseudocode for SA estimation*

Figure 7 gives a sketch of the SA estimation algorithm.

3.2 SS estimation

3.2.1 *The effect of the Initial Window*

In the SS technique, we examine each TCP callee-to-caller flow for which we can estimate its MSS. Additionally, the flow must start with at least five consecutive data packets after the connection establishment, of which the first four must be MSS. If this is the case, let $\{\delta_1, \delta_2, \delta_3, \delta_4\}$ be the four interarrivals between the first five data packets. Suppose that the Initial Window is I segments. If I is 2, 3, or 4 segments, it is easy to see that the RTT estimate is the time spacing between the first and second bursts in TCP's slow-start. That spacing, which is given by δ_I, is expected to be the maximum of the four interarrivals $\{\delta_1, \delta_2, \delta_3, \delta_4\}$ (see Figure 3). When the Initial Window is one packet (I=1), however, δ_1 may include a delayed-ACK timeout, and so it will be quite larger than δ_2 and δ_4; if that is the case, the RTT estimate is δ_3, as can be seen in Figure 3.

3.2.2 *Lost and out-of-order packets*

Packet loss or packet Out-Of-Order (OOO) delivery is not uncommon in the Internet [20]. We have so far assumed that there are no packet losses in the first two bursts of the flow. The presence of losses in the first two bursts can be detected using the TCP sequence numbers in the flow's packets. Some of the loss and out-of-order cases prevent the SS technique from estimating the connection's RTT, because the loss creates a spacing between the first and second bursts that is related to either the Retransmission Timeout (RTO) (see Figure 8-a), or to the Delayed ACK timeout (Figure 8-b). Our algorithm detects packet loss or out-of-order delivery, and discards such connections reporting that it cannot report a reliable RTT measurement for them.

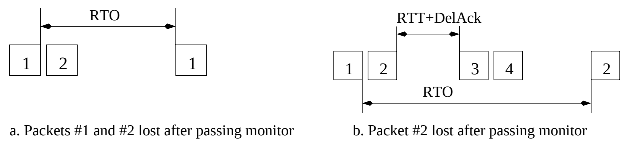

RTO
1
2
1
a. Packets #1 and #2 lost after passing monitor
RTT+DelAck
1
2
3
4
2
RTO
b. Packet #2 lost after passing monitor

Figure 8: Two packet loss scenarios in the first two TCP bursts.

3.2.3 *The SYNACK-ACK sanity check*

In HTTP callee-to-caller flows, we often observe a pure ACK after the SYNACK and before any data packets. That ACK packet acknowledges the HTTP request that the client sent, before the actual data transfer starts. It is easy to see that the time interval X between the SYNACK and that pure ACK is at least one RTT (see Figure 9). We use this time interval X as a 'sanity check' for our SS estimate RTT_{SS}: if RTT_{SS} is appreciably larger than X, the SS estimate is probably wrong, and so it is discarded.

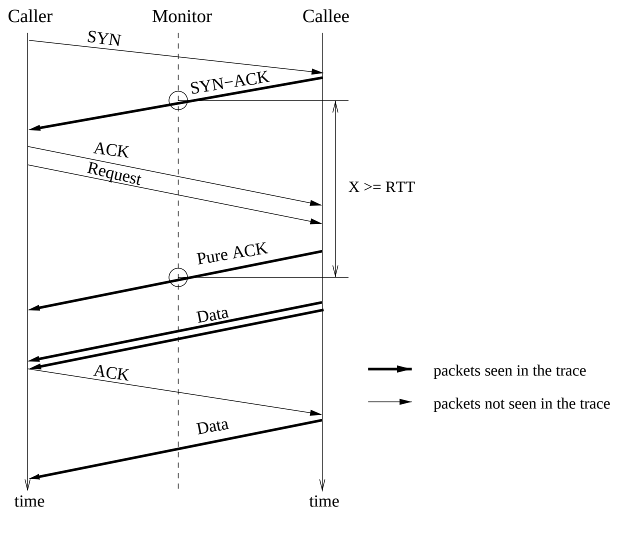


Figure 9: The SYNACK-ACK sanity check.

3.2.4 *Pseudocode for SS estimation*

Figure 10 gives a sketch of the SS estimation algorithm.

```
foreach (SS flow)
{
    P[i] = i'th data packet;      (i $\geq$ 5)
    if (i<5)      skip;          // not enough data packets
    MSS = detect_MSS_value(P[]);
    if (MSS < 0)   skip;          // unknown MSS
    for (i=1; i<5; i++)
        if (P[i] == MSS)    $\delta$[i] = P[i+1].time - P[i].time;
        else      skip;          // not enough MSS packets
    if ($\delta$[1]>>$\delta$[2] && $\delta$[1]>>$\delta$[4])
         RTT = $\delta$[3];              // ICW = 1
    else  RTT = max{$\delta$[2], $\delta$[3], $\delta$[4]};     // 2 $\leq$ ICW $\leq$ 4
    if (pure-ACK after SYNACK) {
         X = P[ACK].time - P[SYNACK].time;
         if (X $\geq$ RTT)    return RTT;
         else            skip;
    }
}
```

Figure 10: Pseudocode for SS estimation.

3.3 Skip cases

There are many TCP connections for which our estimation techniques cannot produce an RTT. These 'skip cases' are listed in Table 1. A 'skip' occurs when our algorithm prefers to discard a connection, rather than producing an estimate which is likely wrong. The various reasons to skip a connection are explained in the first column. The table also gives the amount of TCP traffic that falls in each case as a fraction of connections, and as a fraction of transferred bytes in each dataset.

Unfortunately, both the percentage of skipped connections and the percentage of skipped bytes is significant in the traces that we have examined (between 40% to 50% for connections, and 15% to 45% for bytes). It is important to note that *practically all skip cases occur in callee-to-caller flows and SS estimation.* The most common reason for ignoring a callee-to-caller flow is that it does not include at least four consecutive MSS packets after the connection establishment. This is often the case with WWW 'mice', with operating systems that do not set the Initial Window to a multiple of MSS, and with applications that perform several small 'writes' to the TCP socket instead of a single large one. We note that even if the SS technique fails to produce an RTT estimate for each callee-to-caller flow, it can still be used to give an RTT distribution, based on a large sample of the TCP connections in the link.

4. VERIFICATION

In this section, we examine the accuracy of the previous RTT estimation techniques. We follow two verification approaches. The first is to compare the SA and SS estimates with active RTT measurements between that connection's end-hosts. The second verification approach is indirect, and it is based on the relation between the SA and SS estimates.

4.1 Direct verification approach

Our objective in this approach is to directly compare the SA and SS passive RTT measurements with active RTT measurements using *ping.* Given that both SA and SS examine the initial phases of a TCP connection (3-way handshake and slow-start, respectively), the *ping* measurements are collected just before the connection establishment.

In more detail, the experimental methodology is as follows. A set of about 120 university WWW servers around the world form our 'measurement targets'. The targets are distributed among US, European, and Asian sites, with about 40 servers in each of the three groups. A publicly available HTTP file, larger than 6KB, is identified in each server. A local machine in our lab measures the RTT to each measurement target using ten *ping* measurements, and then transfers the identified HTTP file from the target using the GNU *Wget* utility. The same machine serves as a network monitor, collecting a trace of all TCP traffic using *tcpdump.* After the transfer is over, the RTT of the HTTP-TCP transfer is estimated from the trace using both the SA and SS estimation algorithms, if possible. Finally, the two RTT estimates are compared to the median of the ten *ping* measurements. The previous procedure was repeated on four different days for each measurement target, to collect a total of about 480 comparison points for SA estimation, and about 350 comparison points for SS estimation. The number of SS estimates is significantly lower, as some of the HTTP transfers did not meet the requirements of the SS estimation algorithm (see Figure 10).

It is important to note that although the trace is collected at one of the two connection end-hosts, the network can still cause delays and losses in both directions of the TCP connection. Consequently, the previous verification approach does not make the passive RTT estimation easier in any way. Second, the RTT in a connection's path is certainly not constant, due to queueing delay variations. For this reason, we use the median of the *ping* measurements as a basis for comparison with the SA and SS estimates. Another approach would be to use the last *ping* measurement before we start the TCP connection; we did not notice significant differences with using the median, however.

Figure 11 shows the SA and SS estimates in comparison with the corresponding *ping* measurements for each of the three geographical areas. Note that since our host is in the East Coast of USA, the RTTs to many US targets are larger than the RTTs to European targets, and the RTTs to Asian targets are generally even larger. *We consider that an SA or SS estimate is accurate, if it is within 5ms or 10%, whichever is larger, of the corresponding median ping measurement.* The dashed lines in Figure 11 bound this accuracy region. *With this error tolerance, and for at least this small set of measurements, we see that the fraction of inaccurate measurements is roughly 5-10% for SA estimates, and only slightly more (10-15%) for SS estimates.*

In absolute terms, the estimation errors generally increase in paths with larger RTTs, as shown in the case of European and Asian targets. This is probably because longer paths tend to consist of more multiplexing devices (routers and switches), introducing increased jitter in the spacing between the packets of a connection. Jitter in the 3-way handshake packets, or in the first two slow-start bursts, can add significant noise in the SA and SS techniques, respectively.

4.2 Indirect verification approach

In order to evaluate the estimation algorithms on the NLANR traces, which are collected from edge and backbone links, we use an indirect verification approach[4]. *The underlying idea in this approach is to compare the SA and SS RTT estimates for the same connection, when the trace includes both flows from the caller to the callee and from the callee to the caller.* This is possible for a significant fraction of connections in the NLANR traces. *Since the SS and SA techniques are independent, it is unlikely that both techniques will give the same and wrong RTT estimate.* Consequently, if the SA and SS techniques give approximately the same RTT for a certain connection, we expect that that RTT is correctly estimated.

Figure 12 shows the distribution of the difference between the SA and SS estimates using several traces at four different links. A difference is measured for each connection that gives both an SA and an SS estimate. The graphs at the left column show the distribution of the absolute difference, i.e., $RTT_{SA} - RTT_{SS}$. The absolute difference is used here because there is no way to know which one of two estimates, RTT_{SA} and RTT_{SS}, is more accurate. The dashed lines mark the $\pm$25ms range.

The lowest 10% to 20% of the difference distributions are due to connections in which the SS estimate is significantly larger than the SA estimate, probably because RTT_{SS} has been overestimated. The highest 5% to 15% of the difference distributions are due to connections in which the SA estimate is significantly higher than the SS estimate, either because RTT_{SA} has been overestimated (because of delays at the transmission of the first ACK), or because RTT_{SS} has been underestimated. Overall, we observe that *the two RTT estimates have an absolute difference that is less than 25ms in about 70%-80% of the processed TCP connections.*

The graphs at the right column show the corresponding scatter plots for the two RTT estimates. The scatter plots show how the absolute difference between the two estimates relates to the absolute magnitude of the estimates. The dashed lines mark again the $\pm$25ms range. Overall, *we do not observe a bias towards larger RTT differences in connections with relatively small or large RTT values for the traces we examined.*

We finally note that a significant difference between the SA and SS estimates is possible in transfers over slow links (e.g., dial-up modems). The serialization time of a packet at such links can dominate the RTT, especially with large packets. Consequently, the SA estimate, which is based on the small (40-50 byte) 3-way handshake packets, can be much smaller that the SS estimate, which is based on MSS packets.

5. RTT DISTRIBUTIONS

5.1 RTT distributions at different locations

In general, the RTT distribution at a link depends on the geographical location of each connection's end-points. Therefore, it is expected that different links can have significantly different RTT distributions. Figure 13 shows the RTT cumulative distribution functions (CDFs) at both interfaces of four duplex links, in terms of connections and bytes. The four links cover a wide range of geographical locations and network types. SDC is an OC3 access link at the San Diego Supercomputer Center that carries commodity Internet traffic. COS is an OC3 access link that connects Colorado State University to the Front Range GigaPoP, and it carries both Internet-2 and commodity traffic. IND is an OC12 link at the Indiana University GigaPOP, and it is part of the Abilene network's backbone. TAU is an OC3 link at Tel Aviv University that connects the Israel GigaPoP to Internet-2 in the US.

The four graphs in Figure 13-a show the RTT distributions *in terms of TCP connections*, i.e., one RTT measurement per connection. A general observation is that *more than 90-95% of the TCP connections have an RTT that is less than 500ms.* Additionally, *in the case of US links, more than 75-90% of connections have an RTT is less than 200ms.* In terms of a lower RTT bound, there is a significant fraction of TCP connections in all traces with an RTT of just a few milliseconds. These are connections within the local geographical area of the monitored link. It is noted that the RTTs at a monitored link cannot be lower than the round-trip propagation delay of that link.

Note that the backbone link (IND) has a wider RTT distribution that the two US access links (SDC and COS). It is likely that this is because IND, being a core link, carries traffic between a more geographically dispersed population of users. SDC has a slightly wider distribution than COS as well. Since SDC resides at the southwest tip of the US, while COS resides more centrally in the US, it is reasonable that the COS connections to US sites have a smaller RTT range than the SDC connections.

The effect of the geographical location is more prominent in the case of the TAU link. The TAU RTT distribution makes a significant 'step' between about 50ms and 200ms. About 35% of the connections have an RTT that is less than 50ms, while the rest of the connections have an RTT that is larger than 200ms. We expect that the former group is connections within Israel, or between Israel and Europe, while the latter is connections mainly to North America.

Figure 13-b shows the RTT CDFs for the same traces and links, but *in terms of bytes*, i.e., one RTT measurement per transferred byte. These CDF graphs are more sensitive to individual connections, since a huge transfer can cause a significant vertical step in the measured RTT distribution. This is the case, for example, with a connection at interface-1 of the COS link: that connection transferred about 30% of the link's traffic during the trace, and it had an RTT of about 200ms. As was also the case in Figure 13-a, more than 95% of the bytes are transferred from connections with RTT less than 500ms.

Figure 13 shows that the RTT distributions of the two link interfaces are quite similar, when the distributions are plotted in terms of connections. This is the case with these particular links because they usually carry both directions of a TCP connection. So, if we estimate the RTT of a certain connection at interface-1, it is likely (but not always the

[4]Direct verification based on *ping* measurements would not be possible, because the NLANR traces encrypt the source and destination IP addresses.

case) that we will also estimate the RTT of that connection in interface-2. In terms of bytes, however, the RTT distributions of the two interfaces can be significantly different. This is expected, as most TCP connections transfer data only in one direction.

We also investigated the correlation between a connection's RTT and transfer size to examine whether shorter-RTT connections tend to transfer more data. Based on the measurements that we have analyzed so far, it is inconclusive to say that this is the case in general. At the TAU link, however, we note that about 60% of the bytes are transferred by short-RTT connections (RTT less than 50ms), while the fraction of short-RTT connections is only 35%. The presence of Web caches (or similar proxy servers) in Israel may be one underlying reason for this effect. Another reason may be that large transfers usually take place locally in that dataset.

5.2 RTT variations in different timescales

By splitting a trace of duration T into four parts, each having a duration about $T/4$, we can examine the stationarity of the RTT distribution in the timescales of tens of seconds. For an ordinary NLANR trace, T is usually about 90 seconds, and so each trace-quarter is about 22 seconds. Figure 14 shows the four resulting RTT distributions[5] for both interfaces of the SDC link. Visually, the CDFs at the four quarters are almost the same, providing some evidence that the RTT distributions do not change significantly in the timescales of tens of seconds for the traces we examined. Occasionally, there are some measurement periods over which the RTT distribution is quite different, but this usually happens when the corresponding trace-quarter does not include enough measurement points.

In the timescales of hours, we are mostly interested in differences between daytime and nighttime. Figure 15 shows the RTT distributions for both interfaces of the SDC link at three different times on a weekday in 1999[6]. One of the traces (4:29pm) was captured during daytime in the continental US, while the two other traces (1:47am and 7:52am) were captured at late night in the West Coast and early morning in the East Coast respectively. What is more important is that during both 1:47am and 7:52am EST, it is daytime in both Europe and Asia. Consequently, we expect that the link's workload during those two traces carried a larger fraction of traffic from remote continents. This explains why the RTT distributions in the 1:47am and 7:52am traces are larger by 100-300ms for about 50% of the connections than the 4:29pm trace.

In the timescales of days, we are interested in differences between weekdays and weekends. We were initially expecting to measure lower RTTs during the weekend, as a result of lower network usage, and thus lower queueing delays. It turned out, however, that that is not the case, at least in the links that we have experimented with. Figure 16 shows the RTT distributions for both interfaces of the IND link at about the same time on three different days in September 2001. Note that there are no important or consistent differences in these RTT distributions between weekdays and weekends. The reason may be that the queueing delays in several backbone networks (including the Abilene) are quite low compared to the propagation delays.

[5]The following RTT distributions are all in terms of connections.

[6]We remind the reader that all the reported times are in EST, and so the corresponding time in San Diego would be three hours earlier.

In the timescales of months, variations in the RTT distribution can be due to technology changes (e.g., addition of new links or routers), or due to long-term Internet evolution trends (e.g., gradually lower queueing delays). Figure 17 shows the RTT distribution over more than a year (March 2000 to April 2001) at TAU. An RTT distribution is measured for every other month. Prior to September 2000, a large fraction of connections had larger RTTs than 400ms. Sometime around October 2000, there was a major shifting of the RTT distribution towards values that were about 200ms lower. A possible explanation for this change is that a satellite link with round-trip delay of about 400ms was replaced with a transatlantic link with round-trip delay of about 200ms, causing a 200ms RTT reduction in all the Israel-USA traffic. The distribution from 10/14/2000 is somewhere in the middle, indicating that the network may have been in a transition state during that month, using both the satellite and transatlantic links.

5.3 Effect of queueing delays

Overall, we have not observed persistently increased RTTs during certain times or days that we can attribute to queueing delays. As an exception to the rule, however, Figure 18 shows the RTT distributions at SDC on May 15, 2000 (a Monday) at 11:21am and 2:54pm EST. The 2:54pm trace gave an RTT distribution that is shifted by approximately 100ms towards larger values at both interfaces. A possible explanation for this change is that 'the Internet cloud around SDSC' became significantly congested on that morning, inflating the RTT of most connections by about 100ms.

Based on all the traces we examined, we summarize in the perspective of 'middleman' in the network that RTT distributions are different across network, they may vary in the granularity of hourly timescale, and they strongly depend on the traffic conditions of the Internet cloud. So, it is important to be able to passively measure RTTs in the middle of network for those possible applications mentioned in the introduction, instead of simply hardcoding one known RTT.

6. SUMMARY AND FUTURE WORK

We presented and evaluated a passive measurement methodology that attempts to estimate the RTT of a TCP flow during the connection's initial stages. Measurements on several NLANR traces show that the estimation algorithm produces an RTT value for 55-85% of the TCP workload, in terms of bytes. In terms of accuracy, we found that about 90% of the passive measurements are within 10% or 5ms, whichever is larger, of the RTT value that *ping* would measure. Also, experiments with NLANR traces showed that the SA and SS passive RTT estimates agree within 25ms for 70-80% of the processed TCP connections.

One possibility for future work would be to improve the SS technique so that it is applicable to a larger number of connections, and to analyze more than the first slow-start round-trip. We spent a significant amount of time attempting to do so. Unfortunately, it is quite hard to analyze in a robust manner the slow-start behavior of unidirectional TCP flows in ‘the middle of the network’. This is because there are significant differences among various TCP implementations in terms of their ACKing policy, initial window, and MSS selection. Application-layer issues, such as using persistent HTTP connections, or writing small chunks of data to the TCP socket, can cause additional unpredictability in the evolution of a TCP connection. Finally, the possibility of losses, reordering, and arbitrary delay jitter in the flow of data segments or ACKs makes the analysis even more difficult.

As a next step, we plan to investigate ways in which routers can use this RTT estimation methodology in real-time to compute the required amount of buffering at the link, to configure Active Queue Management modules, or to detect congestion unresponsive flows. Another interesting task would be to investigate the fraction of TCP connections that a router would need to examine in order to get a good approximation of the link’s RTT distribution. Such sampling can reduce the overhead associated with these measurements.

7. ACKNOWLEDGMENTS

The authors are grateful to the three anonymous referees and to the CCR editor B. Krishnamurthy. The two-round review process helped significantly in improving the paper.

This work would not be possible without the traces collected from the NLANR Passive Measurement and Analysis (PMA) project. The NLANR PMA project is supported by the National Science Foundation Cooperative Agreement No. ANI-9807479, and by the National Laboratory for Applied Network Research.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U. S. Government.

8. REFERENCES

[1] M. Allman. A Web Server’s View of the Transport Layer. *Computer Communication Review*, 30(5), Oct. 2000.

[2] M. Allman and V. Paxson. On Estimating End-to-End Network Path Properties. In *Proceedings of ACM SIGCOMM*, Sept. 1999.

[3] M. Allman, V. Paxson, and W. Stevens. *TCP Congestion Control*, Apr. 1999. IETF RFC 2581.

[4] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, M. Stemm, and R. H. Katz. TCP Behavior of a Busy Internet Server: Analysis and Improvements. In *Proceedings of IEEE INFOCOM*, Apr. 1998.

[5] R. Braden. *Requirements for Internet Hosts – Communication Layers*, Oct. 1989. IETF RFC 1122.

[6] J. Cleary, S. Donnelly, I. Graham, A. McGregor, and M. Pearson. Design Principles for Accurate Passive Measurement. In *Proceedings Passive and Active Measurements (PAM) workshop*, Apr. 2000.

[7] S. Floyd and K. Fall. Promoting the Use of End-to-End Congestion Control in the Internet. *IEEE/ACM Transactions on Networking*, 7(4):458–473, Aug. 1999.

[8] C. Fraleigh, C. Diot, B. Lyles, S. Moon, P. Owezarski, D. Papagiannaki, and F. Tobagi. Design and Deployment of a Passive Monitoring Infrastructure. In *Proceedings of Passive and Active Measurements (PAM) Workshop*, 2001.

[9] V. Jacobson. TCPdump, the protocol packet capture and dumper program. ftp://ftp.ee.lbl.gov/tcpdump.tar.Z.

[10] V. Jacobson. Congestion Avoidance and Control. In *Proceedings of ACM SIGCOMM*, pages 314–329, Sept. 1988.

[11] R. Mahajan, S. Floyd, and D. Wetherall. Controlling High Bandwidth Flows at the Congested Routers. In *Proceedings of IEEE ICNP*, Nov. 2001.

[12] H. S. Martin, A. J. McGregor, and J. G. Cleary. Analysis of Internet Delay Times. In *Proceedings of Passive and Active Measurements (PAM) workshop*, 2000.

[13] V. Misra, W. B. Gong, and D. Towsley. Fluid-based Analysis of a Network of AQM Routers Supporting TCP Flows with an Application to RED. In *Proceedings of ACM SIGCOMM*, Sept. 2000.

[14] J. C. Mogul and S.E.Deering. *Path MTU Discovery*, Apr. 1990. RFC 1191.

[15] R. Morris. Scalable TCP Congestion Control. In *Proceedings of IEEE INFOCOM*, Apr. 2000.

[16] R. Mortier, I. Pratt, C. Clark, and S. Crosby. Implicit Admission Control. *IEEE Journal on Selected Areas in Communications*, 18(12):2629–2639, Dec. 2000.

[17] E. M. Nahum, T. Barzilai, and D. Kandlur. Performance Issues in WWW Servers. *IEEE/ACM Transactions in Networking*, 2001. Accepted for publication.

[18] NLANR MOAT. Passive Measurement and Analysis. http://pma.nlanr.net/PMA/, Nov. 2001.

[19] S. Ostermann. tcptrace: TCP dump file analysis tool. http://www.tcptrace.org.

[20] V. Paxson. End-to-end Internet Packet dynamics. In *Proceedings of ACM SIGCOMM*, Sept. 1997.

[21] V. Paxson and M. Allman. *Computing TCP’s Retransmission Timer*, Nov. 2000. IETF RFC 2988.

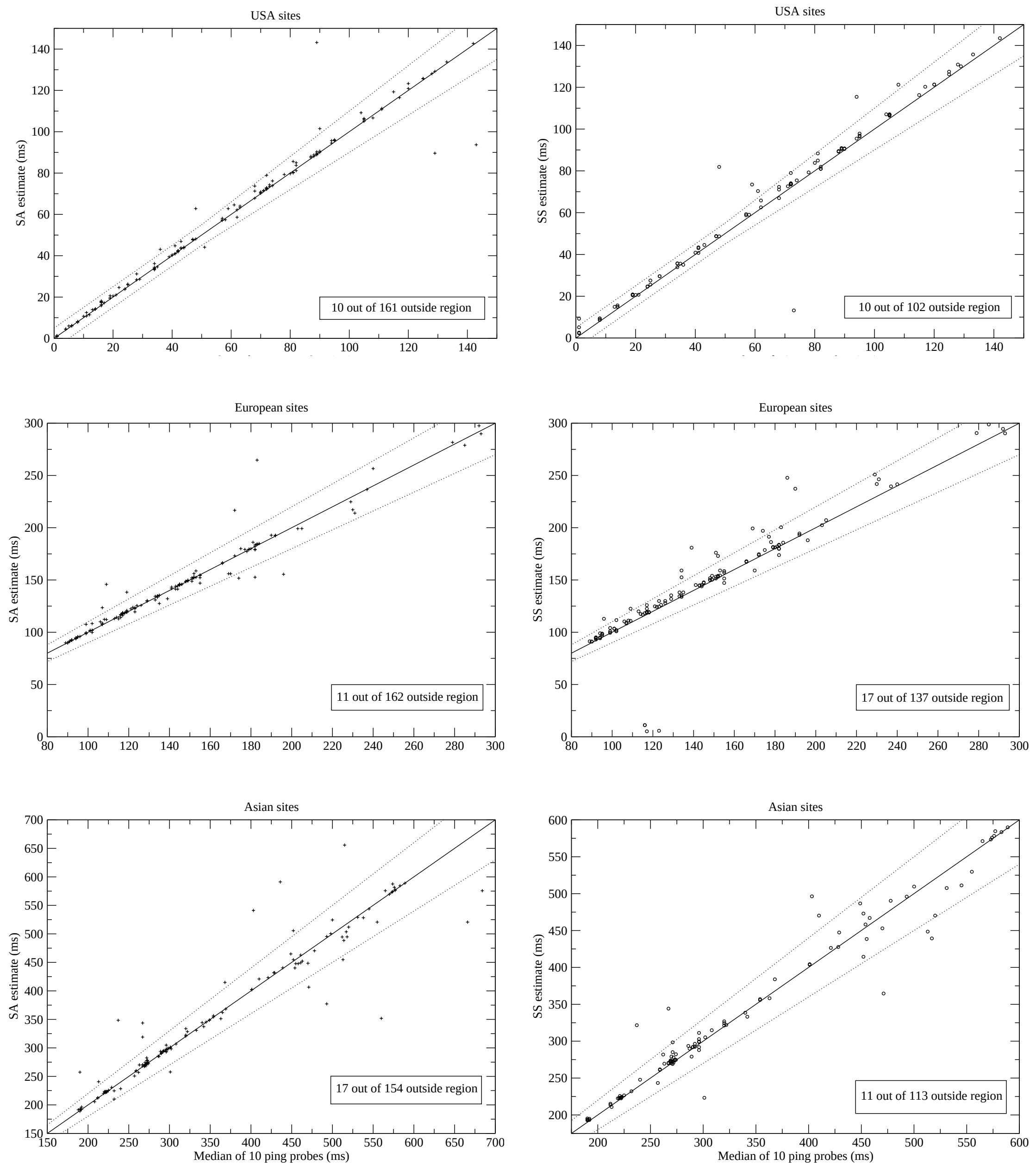
[22] L. L. Peterson and B. S. Davie. *Computer Networks, A Systems Approach*. Morgan Kaufmann, 2000.

[23] A. Veres. end2end-interest mailing list. http://www.icir.org/floyd/other/Veres.April2001, Apr. 2001.

[24] C. Villamizar and C.Song. High Performance TCP in ANSNET. *ACM Computer Communication Review*, Oct. 1994.

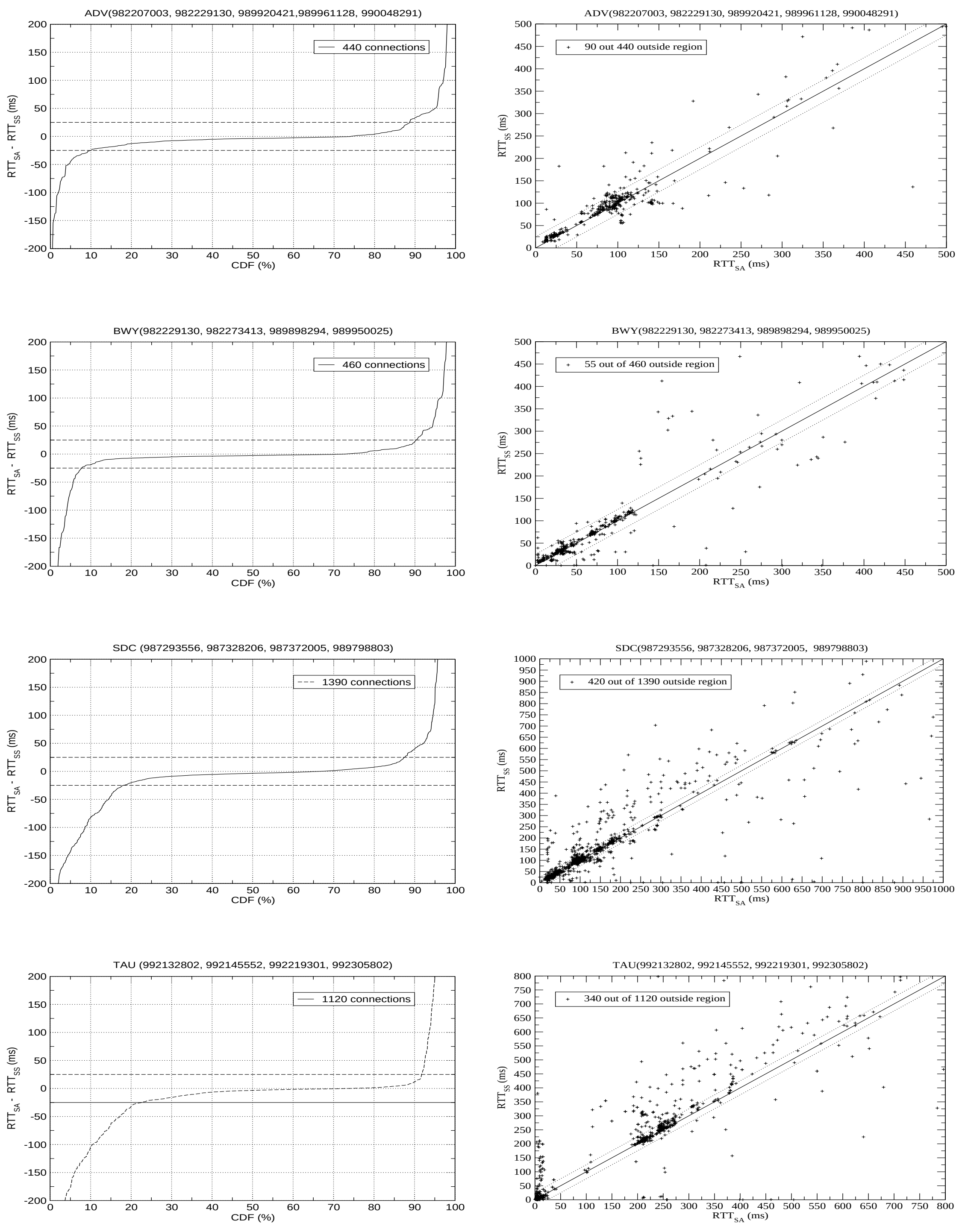
Skip case	ADV-990048291		SDC-989798803		BWY-989898294		TAU-992132802	
	% conns	% bytes	% conns	% bytes	% conns	% bytes	% conns	% bytes
SA sanity checks	0.0	0.0	0.3	0.1	0.0	0.0	0.1	0.0
SS sanity checks	3.1	20.0	1.3	6.3	0.3	0.7	0.6	5.2
Packet Loss/OOO delivery	0.4	0.8	0.8	2.5	1.0	12.2	0.3	0.7
All other SS skip cases	37.5	21.1	41.7	20.6	47.4	5.2	47.1	32.3
Total	41.0	41.9	44.1	29.5	48.7	18.1	48.1	38.2

Table 1: Skip cases and the corresponding fractions of TCP connections and TCP bytes.



(a) SA estimates (b) SS estimates

Figure 11: Comparison of the SA and SS estimates with the median of ten *ping* measurements just before the TCP connection establishment.



(a) Difference distribution (b) Scatter diagrams

Figure 12: Difference distributions between the SA and SS RTT estimates, as well as the corresponding scatter plots, for four links.

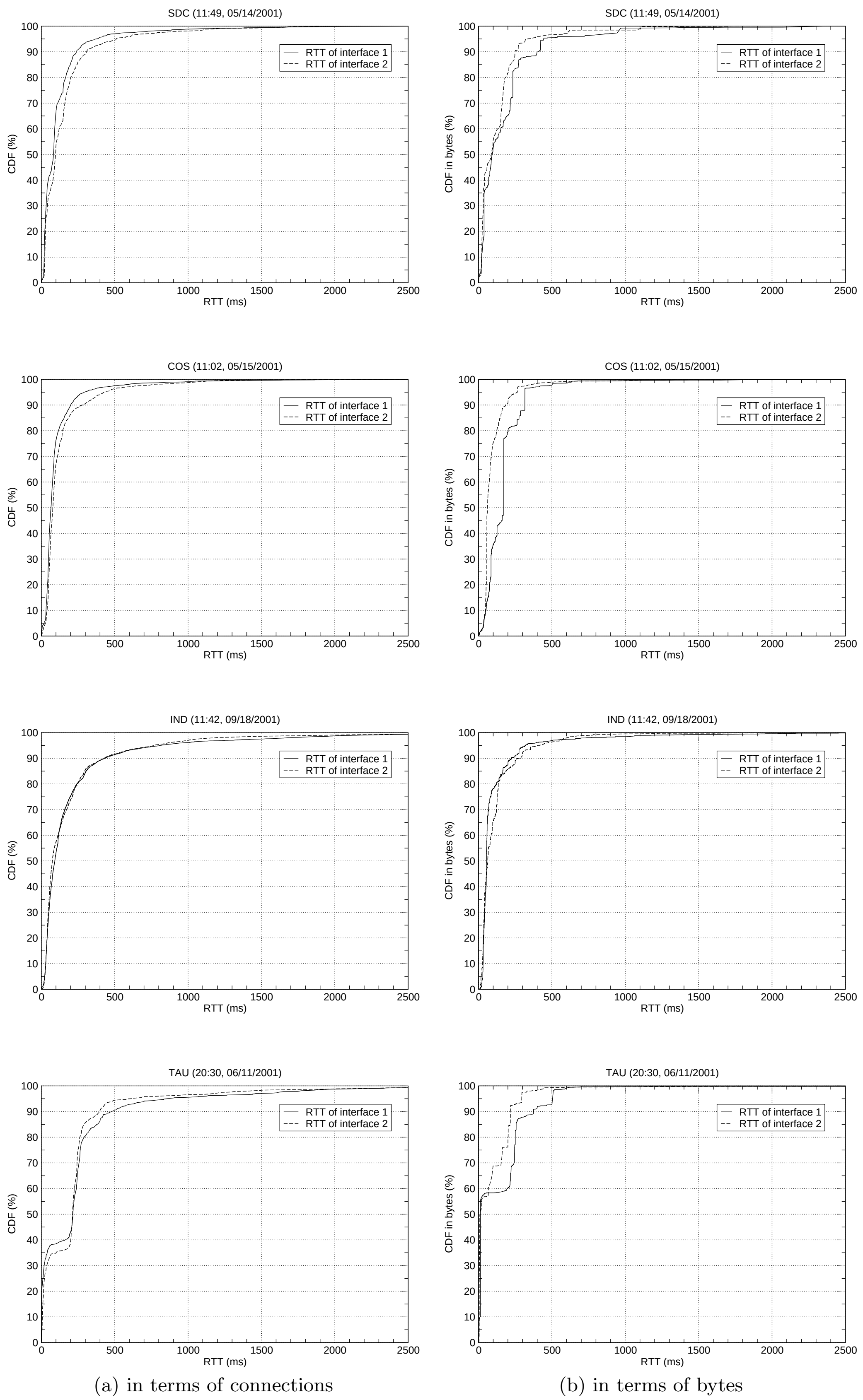
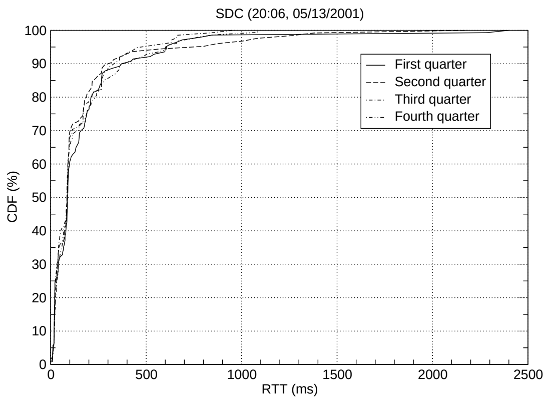
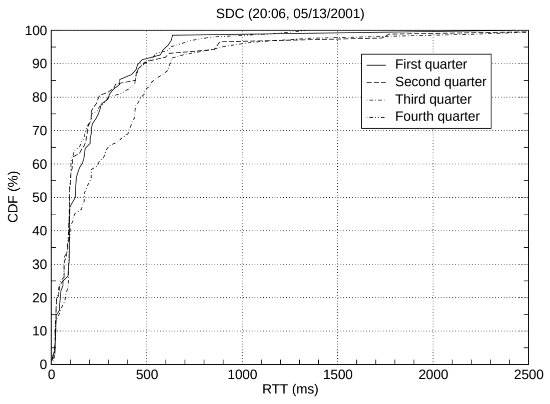


Figure 13: RTT distributions at four duplex links

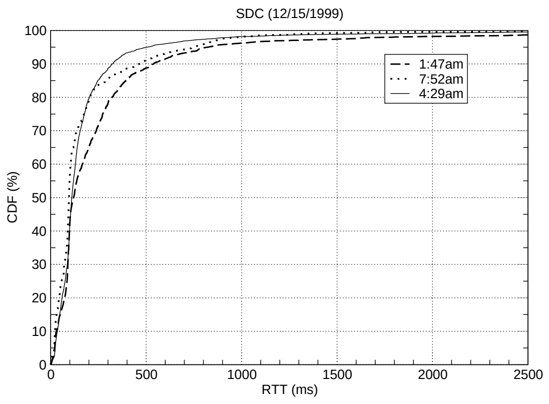


(a) interface 1

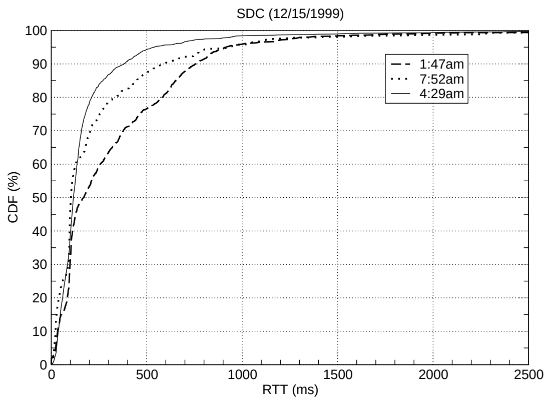


(b) interface 2

Figure 14: Variations in short timescales (tens of seconds)

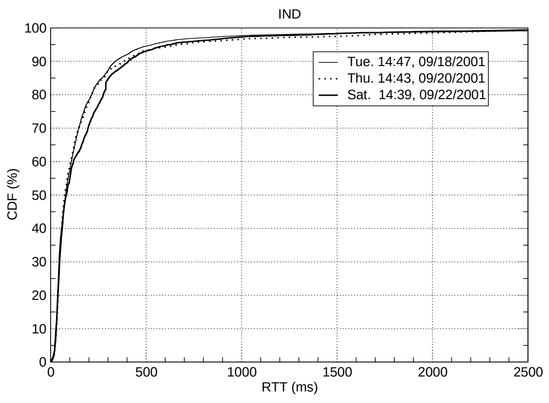


(a) interface 1

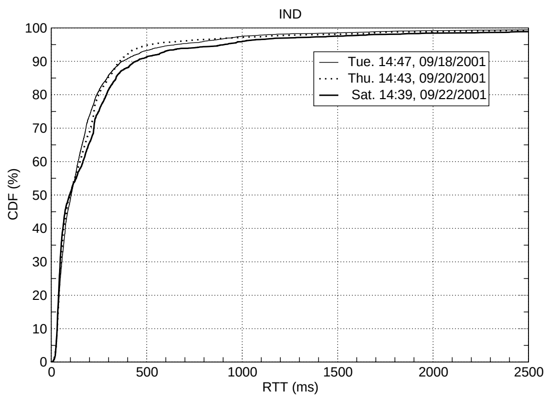


(b) interface 2

Figure 15: Variations in hourly timescales

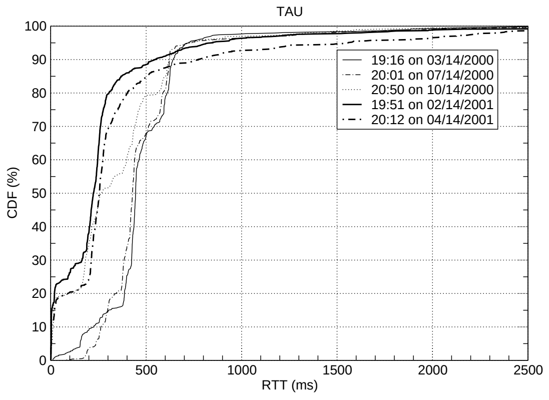


(a) interface 1

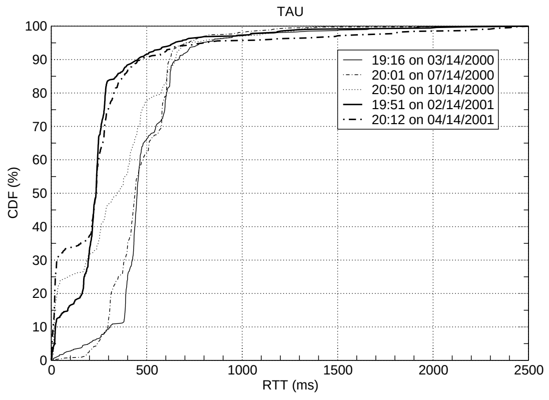


(b) interface 2

Figure 16: Variations in daily timescales

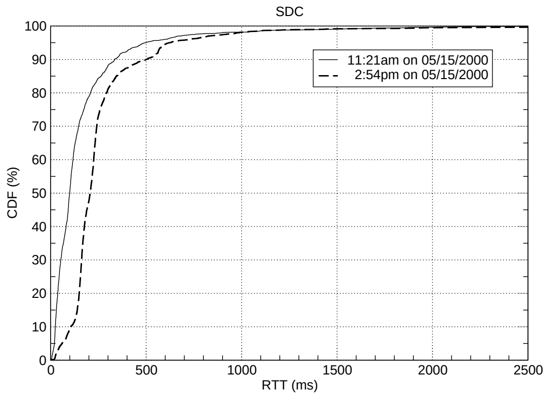


(a) interface 1

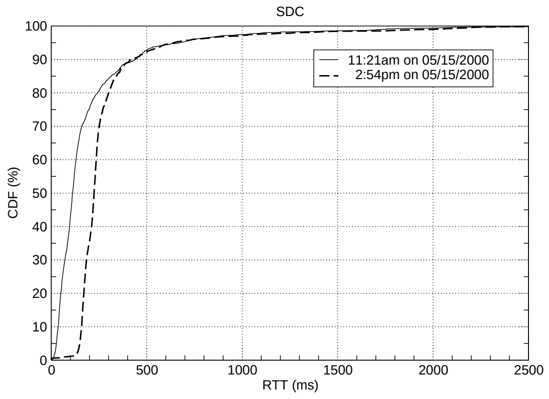


(b) interface 2

Figure 17: Variations in monthly timescales



(a) interface 1



(b) interface 2

Figure 18: Increased RTTs due to queueing delays